Object Oriented API Layers Improve the Modularity of Applications Controlling Accelerator Physics

T. Birke, R. Lange, R. Müller

Berliner Elektronenspeicherring-Gesellschaft für Synchrotronstrahlung m.b.H. (BESSY), Lentzeallee 100, D–14195 Berlin, FRG*

Abstract

With EPICS the architecture of core software and standard control programs is largely defined by the given tools. The installation of EPICS at BESSY II focuses attention on configuration and improvement issues. This does not cover the applications measuring and controlling accelerator physics. Despite similar problems at different sites these types of programs are seldom portable due to the low degree of modularity or encapsulation. At BESSY dedicated API layers written in C++ have proven to be a useful means to hide implementation details: An event-driven interface to a graphics server allows one to write programs with a graphical user interface on a windowing system (now: X11/Motif) without detailed knowledge of that system. Device objects on top of the equipment access layer (now: CA/*cdev*) implement hardware specific control methods. This facilitates device independent coding of measurement sequences.

I. Introduction

The 'Berliner Elektronenspeicherring-Gesellschaft für Synchrotronstrahlung m.b.H.' operates a 0.8 GeV storage ring dedicated to the generation of synchrotron light in the VUV and soft X-ray region, BESSY I.[1] The successor BESSY II has been designed as a high brillance synchrotron light source in the VUV/XUV region with 1.7 GeV nominal beam energy. Sixteen DBA cells provide alternating dispersion-free straight sections of high and low horizontal beta function. This machine is presently under construction at Berlin-Adlershof and scheduled to start routine operation in 1998.[2]

The replacement of the original control system of BESSY I [3], [4] was completed in 1993. [5] The architecture of the new system embodies the 'standard model' of distributed control system design. Due to the restrictions imposed by the replacement process it contains some peculiarities. As a novel feature full advantage of embedded controllers and the properties of the CAN fieldbus is taken.[6], [7]

Apart from an attempt to save investments and know-how where appropriate there are no intrinsic boundary conditions on the control system design for the new machine BESSY II. We decided not to proceed with our own development but to base the system core on EPICS.[8] As part of the overall architecture the extensive use of the concept 'embedded controllers as CAN field bus nodes' remains.[9]

II. Context of Application Development

The control system applications needed for the operation of BESSY II are coupled to the underlying control system core by the 'application programming interface' (API) and the data model involved. That is sufficiently universal to be able to profit from the experience and investments made during the project BESSY I. A new restriction is that developments now have to suit the environment of EPICS tools.

A. Experiences and Investments at BESSY I

Concepts or solutions that have proven their usefulness are planned to be taken over at least partly.

- The data model at BESSY I is very simple. It requires polling, only synchronous 'get' and 'put' operations are implemented. Therefore the callback mechanisms of 'channel access' (CA) provides a big increase in functionality. On the other hand the API for the BESSY I equipment access is already based on exchanging messages with devices. This abstraction level of 'devices' and their data structures is not provided by CA.
- The 'graphical user interface' (GUI) has been implemented by a graphical server encapsulating complete representational aspects. It provides synoptic views, control panels and window hierarchies. Clients communicate with the graphical server on an event driven API exclusively about requests concerning application variables.[4] The most versatile client is an

*eMail: birke@bessy.de, lange@bessy.de, mueller@bessy.de

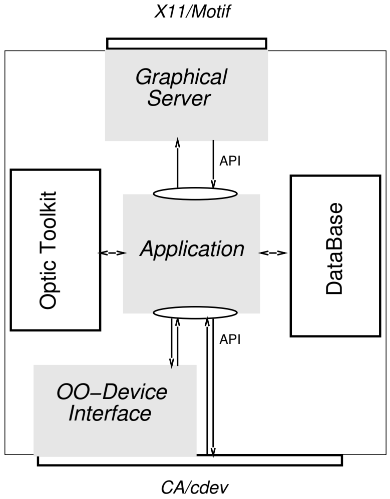

Figure. 1. Application Program Support

interpreter that allows for any equipment access call. Simple tasks can be easily implemented in a scripting language embedded in the representation description file of the graphical server–for the end user somewhat comparable to tcl/Tk scripts.

• A multi purpose hardware driving utility mainly for backup/restore and magnet conditioning has been developed. It implements device classes according to their data sets and meaningful driving methods. Handling of lists of devices, lists of lists and machine data files (snapshots, references) are supported. For the needs of BESSY I macro programming of a single application program that parses command input [10] has been sufficient (Fig. 3).

B. Framework of EPICS

• The *cdev* API [11] provides access to devices by using asynchronous operations and monitor callbacks (features present in CA). At the same time *cdev* hides implementation details of the underlying package(s). *cdev* has been chosen by several major laboratories (EPCS SOSH Workshop, CERN, 5/95) as the candidate for a common calling convention that promises to facilitate sharing of accelerator control applications. Wherever possible applications will be based on the *cdev* API.

• With the EPICS extensions (*medm*, *km*, *dp*, *alh*...) the standard requirements for accelerator operation are covered by sophisticated and high performance tools. However since they are solutions to specific problems on their own the construction of a homogeneous and concise GUI is a challenge. With *medm* it is easy to build fancy screens but there is no support for global variables that affect process variables (e.g. sector number/corresponding corrector power supplies). Support for the construction of hierarchies is minimal (only by the related display button). Resolution of device names into functional parts and the assignment of control panels corresponding to the appropriate device class is impossible. There is no major problem in building synoptic displays with tcl/Tk scripts and in launching the different EPICS tools or *medm* screens graphically in the appropriate context. The main deficiency of this solution is the rapid decrease of maintainability as the GUI evolves.

III. Conceptual Issues

Even if subsets of application programs fulfill the requirements for modularity there are important areas that are usually hard to maintain (Fig. 1):

• Consistency of relatively static configuration data is quite easily provided by making it mandatory for the programmers to use the database (possibly simply stored in central filesets). For BESSY II the reference data will be supplied by the Oracle DBMS.

• Consistent results of model calculations or frequently used standard algorithms can be achieved by providing a centrally maintained optics toolkit and high level software library. Optics calculations will be performed in the context of Goemon [12], a C++ class library. Other utilities are to be developed as they are needed.

• A GUI based on the Motif toolkit requires a considerable amount of understanding. The success of tcl/Tk shows that solutions that do not presume a knowledge of Motif are a reasonable goal. Furthermore even with the support of a

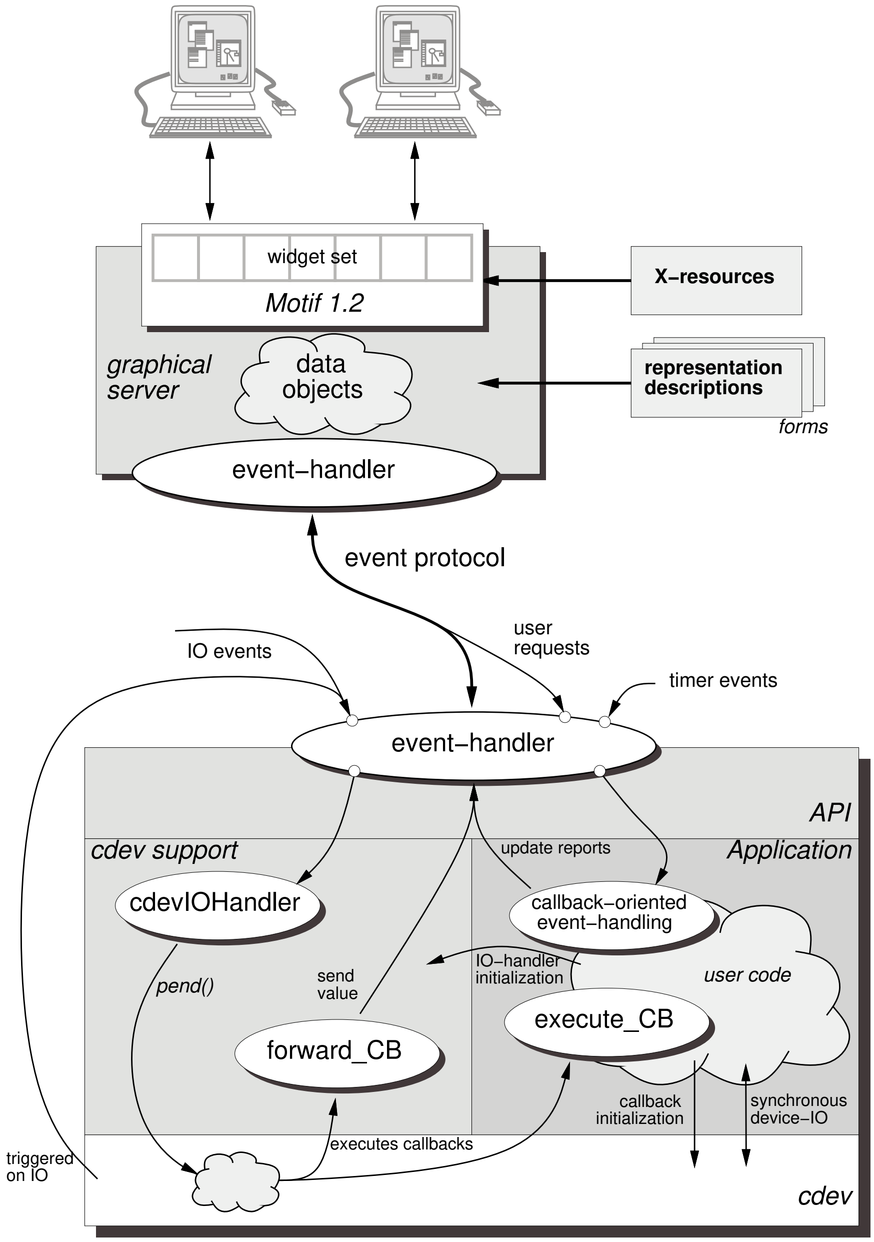

Figure. 2. Graphic Client Event Flow

commercial GUI builder it takes a considerable effort to keep all aspects of the graphical representation localized. The solution of designing an API free from presentational elements and providing network access to the graphic server of an X–display is presented in this paper.

• Hardware control and conditioning programs usually have to deal with all the hardware peculiarities and exceptions. It is described below how the treatment of these hardware specifics is encapsulated in a device object library.

IV. Graphical User Interfaces

The graphical server has been described elsewhere [4] in detail. The structure of this program can be summarized by three blocks: graphical entities based on some windowing toolkit, dynamically created data objects that link presentational and application variables and an event driven interface to the client programs (Fig. 2).

As a result of developments carried out by an external company the presentational layer of the graphical server is now constructed with Motif widgets (formerly InterViews). This improves dramatically the potentialities of this tool: external

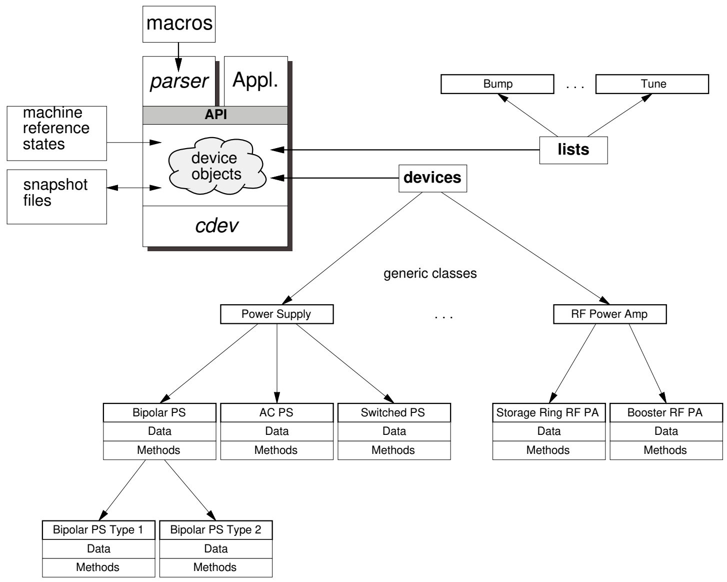

Figure. 3. Sketch of Device Object Environment

widgets are easily introduced and control of specific behaviour is possible with the mechanisms of Motif resource handling. To this is now added the possibility to switch between different presentational entities according to the content in the actual representation description file (form). The forms are requested and evaluated by the graphical server at run time – a feature already present in the previous version of the server .[4] Clients of the graphic server are classically event driven and relatively simply structured. The API calls allow the building of a connection to the server that initiates transfer of application variable data structures and mapping of the window(s). Code is executed according to the incoming events, results are reported by generating events. All client events are grouped into the basic types:

- elementary connection handling (sanity of communication objects)
- user requests (interaction with the GUI)
- update reports (results, changes)
- *cdev* callbacks (frequently simply converted to update reports)

With this API application programmers can take advantage of a powerful GUI with literally no idea about Motif, as long as the widget resources are centrally maintained. This is in contrast to (professional) Motif GUI builders or other toolkits where at least the Motif architecture has to be known to the end user. The generic client of the graphical server named *action language interpreter* [4] is presently adapted to the new EPICS environment by implementing the full *cdev* functionality. It is planned to merge the advantages of GUI elements generated by the graphic server and the generic client programmed with forms, with the fancy screens easily produced and modified by *medm* to an operator interface of high functionality. The former will provide synoptic views and flexible window hierarchies, embed the different *medm* screens and launch the other tools (*km*, *dp*, ...) with the proper parameters.

V. Device Objects

Originally this tool was developed from the need to write an 'intelligent' store-and- load utility for snapshot files that performs conditioning, state modifications of devices etc. According to experience running programs based on the plain equipment access API, with otherwise nicely structured code, are frequently ruined by the exception handling of specific devices. Device objects have been created that adress these problems. They consist of device data sets (status, set point, read back...) and control methods. These methods perform not only the downloading of values or commands, but also complex operations: checks of task success, attempts to put the device into a proper state, possibly the untertaking of repair or surveillance operations. More complex device objects are constructed from simpler objects through inheritance. Collections of these device objects form nested lists. We plan to implement specific lists (e.g. 'bump') inherited from the base list. Devices in these lists are not simply controlled sequentially but ‘simultaneously’ in ‘small steps’. Action is only performed when all the devices in the list are in a proper state. Today we find similar data structures elsewhere in the EPICS environment: the data sets and lists correspond to the *cdev* device attributes and compound devices. The principal application ‘backup/restore’ could be implemented by BURT tasks embedded in shell scripts. The device-specific driving methods still offer additional functionality to the application programmers that cannot be implemented in a site- independent data handling layer like *cdev*. With the API to the library of device objects we plan to ease the coding of measurement and correction programs and enhance maintainability at the same time.

VI. Summary

The rapid installation of the EPICS tools provides immediately basic functionalities for remote control and surveillance of the connected pieces of equipment. Saving of manpower is invested into the development of tools that improve modularity. The graphic server provides a reasonable balance between coding simplicity, representational flexibility, performance power and demands on system resources. It shields the application programmers from the painful task of learning Motif. Specific hardware control methods and device failure condition handling is encapsulated within the library of device objects . The simple and uniform API will enable the programmer to write conditioning or measurement programs without knowing the hardware specifics.

References

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

[2] E. Jaeschke for the BESSY II design team, Conference Record of the 1995 IEEE Particle Accelerator Conference, Dallas, to be published

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

[4] R. Müller, H.–D. Doll, I. J. Donasch, H. Marxen, H. Pause, Conference Record of the 1991 IEEE Particle Accelerator Conference, San Francisco, pp. 1311–1313 (1991)

R. Müller, Proceedings of the 3rd European Particle Accelerator Conference, Berlin, pp. 1167–1169 (1992)

[5] G. v. Egan-Krieger, R. Müller, J. Rahn, Conference Record of the 1993 IEEE Particle Accelerator Conference, Washington, pp. 1887–1889 (1993)

[6] Road Vehicles – Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication, Document Iso/DIS 11898, International Standardization Organization (1991)

[7] G. v. Egan-Krieger, T. Stein, J. Rahn, Nuclear Instruments & Methods in Physics Research A 352, pp. 204–206 (1994)

[8] L. Dalesio, J. Hill, M. Kraimer, D. Murray, S. Hunt, M. Clausen, C. Watson, J. Dalesio, Nuclear Instruments & Methods in Physics Research A 352, pp. 179–184 (1994)

[9] J. Bergl, B. Kuner, R. Lange, I. Müller, R. Müller, G. Pfeiffer, J. Rahn, H. Rüdiger, these Proceedings (ICALEPCS , Chicago, 1995)

[10] R. Lange, Diploma Thesis, FB 20, TU-Berlin, 1993

[11] J. Chen, W. Akers, G. Heyes, D. Wu, C. Watson, these Proceedings (ICALEPCS , Chicago, 1995)

[12] H. Nishimura, Nuclear Instruments & Methods in Physics Research A 352, pp. 379-382 (1994)