THE ACCELERATOR CONTROL SYSTEM AT ELSA

D. Proft*, F. Frommberger and W. Hillert, ELSA, Bonn, Germany

Abstract

To fulfill the new requirements of the post-accelerator mode of the electron stretcher facility ELSA, a new computer control system was developed during the early 1990s. Providing capabilities to control and monitor the facility, it represents the top layer of a distributed control system composed of HP workstations, VME and field bus processors as well as linux based personal computer s. In addition to regular updates and improvements, the HP-UX operated part of the control system recently was ported to linux, so the outdated HP workstations could be replaced by a single linux PC.

All reference values, for example the betatron tune or the extraction energy, can be specified using a window-based graphical user front end. They are directly computed to hardware compatible representations. Vice versa, measured beam parameters, e.g. the transversal beam emittance, are displayed for easy user access, allowing real time diagnostics. This abstraction layer allows for an intuitive approach to machine operation, requiring no detailed knowledge of the hardware implementation.

In this contribution, the design principles and implementation at different layers of the control system are presented, taking the recent changes brought by the migration to linux into account.

HISTORY

Until 1994 the ELSA stretcher ring was operated in the *stretcher mode* [1]. An electron beam prepared in the 50 Hz booster-synchrotron could be injected into the stretcher ring and extracted to hadron physics experiments at variable energy between 0.5 GeV and 1.6 GeV. In the early 1990s higher energies (up to 3.2 GeV) were asked for by the experiments, making a *post-acceleration mode* inevitable.

Going along with the energy increase, a fast ramp of the main magnets' power supplies in the stretcher ring is needed. Therefore, a considerable amount of magnetic field calculations beside long vectors containing power supply current ramps have to be processed. Additionally, a fast ramp up to 3.2 GeV within 300 ms puts high requirements on the beam diagnostics devices and the data analysis.

Neither the hardware running the existing control system, nor the control system itself had enough capabilities to fulfill these new requirements [2]. Hence, a new control system with support for the existing hardware (approx. 50 in-house developed, so-called MACS IO boards interfacing the hardware devices) was developed in-house [4]. In 1995, the old control system was successfully replaced by the new one.

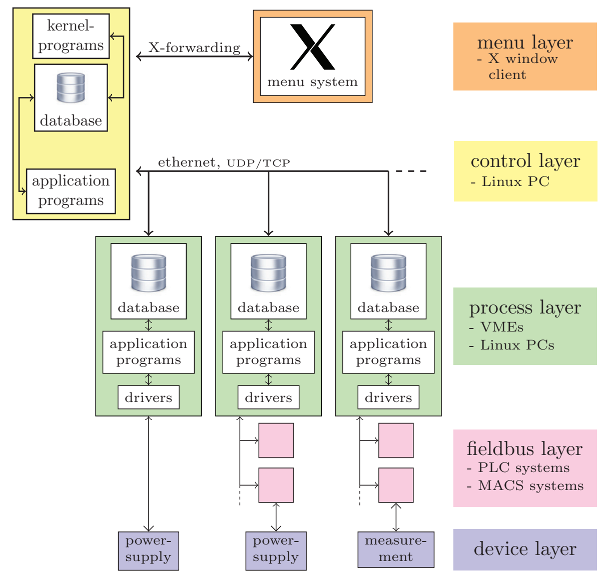

Figure 1: Hard- and software layers of the control system.

Besides continuous improvements of the software components over the last years the system was ported from HP-UX (running on three HP workstations) to linux in the end of 2012, so it can now be run under any linux operating system.

DESIGN PRINCIPLES

Some basic design decisions have been made before the development took place [3]. The main features include a completely event based data handling model and a separation of core functionality (database and event handling by the *kernel*) from userspace applications. It combines steering tasks and real time beam diagnostics in one homogeneous environment. A transparent design allows access to the X windows-based graphical user interface from any computer.

Menu System

The whole control system consists of 5 hard- and software layers (see Fig. 1). On top of them the graphical user interface gives access to all the accelerators parameters. It combines all steering tasks and the beam diagnostics in one platform. The user can choose from approximately 600 hierarchically ordered menus, reaching the desired menu in less than eight clicks.

* proft@physik.uni-bonn.de

ISBN 978-3-95450-122-9

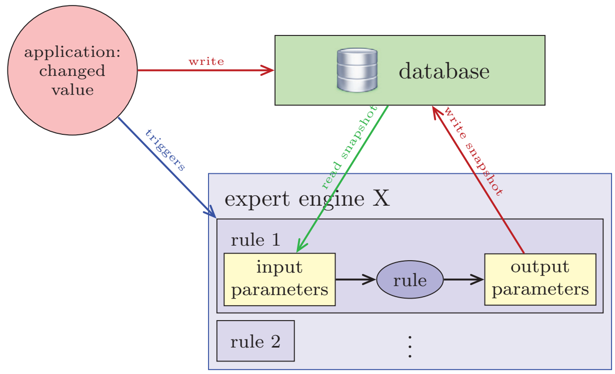

Figure 2: Expert engines.

Control Layer

The next layer — the *control* layer — consists of the kernel managing a shared memory database containing all parameter definitions and values. The database is separated into several parts, i.e. the *resource base* containing structural informations about parameters like limits, max. number of vector elements and the quantity's physical unit. The structural information is complemented by the online database filled with actual parameter values, which are updated continuously at runtime. Every parameter value administered by other control hosts is stored in a cache database to give applications easy access to this parameter as well.

All databases are stored in a shared memory. Any application wishing to access the database attaches to the shared memory and has direct and fast access to it without any wrapper. The mutual exclusive access to the online database is enforced by semaphore locking.

The core applications taking care of the database are extended by so-called *expert engines*. They represent the physical intelligence of the control system, bringing in any physical calculations needed to operate the accelerator. Each expert engine can handle a set of rules which are basically finite state machines. The rule engine is supplied with a consistent database snapshot of all parameters captured at the same time, and itself writes all computed values back to the online database (see Fig. 2).

The kernel and the application programs are not limited to be operated on one single computer. The database is ready to be distributed to several computers, each maintaining a defined range of parameters. Thus, the system can be easily extended and the overall load (currently approx. 13000 parameters) can be distributed to many machines. Since the computer CPU power increased drastically in the last years, the system is now running on a single linux personal computer replacing the three old HP workstations.

Process Layer

The hosts on the control layer are not suitable for direct hardware communication. Therefore, a new middle layer, the *process layer* was introduced. Initially, it consisted of several VME CPUs running the real time operating system VxWorks. All VME computers are diskless clients booting off from a central NFS server and are attached to the control host(s) via ethernet. Several years ago, the infrastructure was extended by a continuously growing number of computers supplied with Intel CPUs. These are running an adopted version of the original process system software. As operating system any linux distribution is suitable.

The process system hosts, regardless whether they feature VME or Intel CPUs, are equipped with several bus interface cards: HDLC, GPIB, serial interfaces (RS-232/RS-485), CAN-bus and PROFIBUS. Additionally, plug-in cards for ADCs or DACs are used within several systems.

A set of defined parameters is assigned to each process layer host during its boot sequence. This includes the structural information beside the runtime values, again organized in a shared memory database. Synchronization with the control hosts is done via network connections using RPC (TCP) and UDP.

For high level applications running on the control hosts the explicit assignment of parameter to process hosts is completely out of scope. This transparent design allows moving the parameters to different hosts without changing the corresponding application.

Fieldbus and Device Layer

Finally, the last layer includes all interfaces communicating directly with the hardware and the hardware devices themselves. The wide variety of power supplies is equipped with in-house developed interface cards. Digital and analog nominal values are directly fed into the interfaces by the MACS I/O boards. These are directly connected to the VME systems via an HDLC interface with 1.25 MBd. Furthermore, measured values (analog values beside binary status variables) are read out from the power supplies and transmitted back the same way.

In the last years, several PLCs were integrated into the system. These approximately 15 systems are interfaced via profibus and accessible like any other device from the control system.

EVENT MANAGEMENT

Like already mentioned, direct read and write operations to the database are achieved by accessing the appropriate section in shared memory. The event management comes into place when other applications need to react in case of a changed parameter. At the first level, an asynchronous notification system is implemented (see Fig. 3). Applications with interest in a particular parameter get signaled by a custom unix signal after the data got written to the database. The informed application is now in charge of reading the changed value itself and react accordingly. Following this approach an automatic load balancing is done (multiple applications running on multiple CPU cores). The only bottleneck could be the mutually exclusive access to the shared memory.

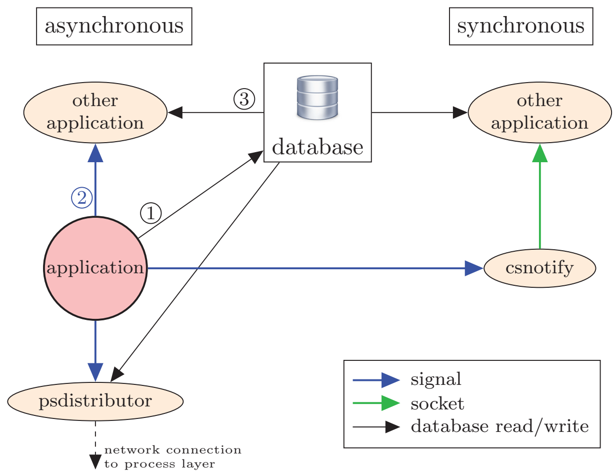

Figure 3: Exemplary event processing triggered by changed parameter. First the new parameter value gets written to the shared memory (1), afterwards all applications are notified by a signal (2). The applications read the new parameter from the database themselves (3).

Benchmarks showed up a raw[1] read/write rate of approximately 300 k ops/s using a single application. Almost the same rate is achieved when multiple applications are accessing the database. During normal operation — typically 3 k to 4 k ops/s — this is no limitation to the performance of the overall system. Taking network latency into account (because parameter values need to be transferred to according process hosts via ethernet), the maximum write rate (of a single process on the control host) reduces to approximately 500 ops/s. The same limitation is valid while transferring measured values from the process hosts back to the control hosts. This rate is too low for realtime measurement. Therefore, the measured values get stored into parameter vectors[2] and transferred at a much lower rate.

The mentioned asynchronous notification system is supplemented by a synchronous one. One core application, *csnotify*, is attached to the asynchronous notifications and gets informed on parameter updates. This way, the application can re-distribute the notification to all processes attached to the synchronous notification system. It is used by applications whose execution cannot be interrupted by signals.

Logging and Monitoring

Any process host and all attached applications are capable of sending messages with seven different logging levels ranging from *debug* to *emergency* via network to a central logger daemon, e.g. in case of failures. Reports in form of textual messages are generated and visualized within a graphical error-logger application.

[1] Measured with a parameter not defined on any process system, which otherwise would decrease the write rate due to network latency.

[2] The control system does not distinguish between single-value parameters and vector ones. A vector is simply a parameter with multiple ordered values.

SOFTWARE INTERFACES OF THE CONTROL SYSTEM

Any beam diagnostics relies on an interface to process the data. At this point, several interfaces to the control systems database come into place. Automated diagnostics are mainly outsourced to dedicated `C` or `C++` applications using the native libraries shipped with the control system. An example for that is the automated real time measurement of the beam emittance using synchrotron radiation monitors in the stretcher ring. Profile images of the beam are captured and analyzed on a process host. The beam width is transfered to the control host on which the emittance calculation takes place.

Furthermore, beam diagnostics can be easily done with very high level programming languages. Therefore, an interface to MATLAB has been created. This allows quick read and write access to any single parameter value or even vectors. Due to its matrix manipulation capabilities, it is a convenient tool to perform beam optics calculations. An example for this is the automated measurement of the optic functions in the external beamline.

Apart from the menu system, graphical tools can be developed using the well known TCL/TK interface. This offers a quick way to develop new standalone (independent from the main menu system) graphical applications with access to the control system.

EPICS based solutions can also be integrated into the existing control system. To that end, a two-way gateway interface between EPICS and the control system was set up on one process host.

Applications not running on any control host or process host can interact with the database, too. Using a single TCP connection, the whole parameter database can be accessed. Even attachment to the notification system is possible. Together with an Android port of the menu system running on mobile devices this interface can be used to steer the accelerator from almost everywhere in the world.

REFERENCES

[1] W. Hillert, “The Bonn Electron Stretcher Accelerator ELSA: Past & Future”, EPJ A s01 (2006) 139.

[2] T. Götz, “Entwicklung und Inbetriebnahme eines verteilten Rechnerkontrollsystems zur Steuerung der Elektronen-Stretcher-Anlage ELSA, unter besonderer Berücksichtigung der Anforderungen des Nachbeschleunigungsbetriebes bis 3.5 GeV”, PhD theses, University of Bonn, 1995

[3] M. Picard, “Entwurf, Entwicklung und Inbetriebnahme eines verteilten Rechnerkontrollsystems für die Elektronen-Stretcher-Anlage ELSA, unter besonderer Berücksichtigung der Extraktion im Nachbeschleunigungsbetrieb bis 3.5 GeV”, PhD theses, University of Bonn, 1995

[4] C. Wermelskirchen, “Das Kontroll- und Steuersystem der Bonner 3.5 GeV Elektronen-Stretcheranlage ELSA”, PhD theses, University of Bonn, 1988

 ISBN 978-3-95450-122-9