EMBEDDED LINUX AND CORBA IN THE GSI CONTROL SYSTEM

K. Höppner, L. Hechler, P. Kainberger, U. Krause, GSI, 64291 Darmstadt, Germany

Abstract

The current GSI control system is based on VME single board computers with M680x0 CPUs at front-end level and OpenVMS at operating and configuration level. Communication is done by a proprietary in-house networking protocol.

A renovation project started to migrate to more actual components. Commercially available PowerPC boards replace the M680x0 boards which had been specially manufactured for GSI. The core software is build newly, implementing a new object oriented device model. It allows, however, to re-use the code of the existing device implementation which contains the specifics to handle the GSI accelerators.

Rather than integrating our in-house networking protocol in the newly build system, communication will be based on CORBA in the future.

MOTIVATION

The control system for the GSI accelerators is in operation since 1989. Since then, according to the continuously extended operation of the accelerators, the controls were more and more refined to handle a greater spectrum of devices and to provide greater flexibility, but no substantial modernization could be done up to now. As a result, the system depends on many components dating from the time when the system was developed. Purchasing spare parts became a problem, and functionality is no longer adequate for state of the art developments.

A renovation of central parts of the system has become indispensable [1]. After preparing the basis for future software developments [2], the most crucial parts of the system can be modernized now: The communication network and the access points at the front-ends, the device presentation controllers with their network interface.

ACTUAL CONTROL SYSTEM

Hardware Layout

The control system was designed as a decentralized distributed system (Fig. 1), according to the nowadays well established standard model. The equipment to be controlled is connected by a field bus to front-end controllers which communicate by Ethernet with operation workstations. M68020 VME boards are used at the front-end level while the operations level uses OpenVMS Alphas.

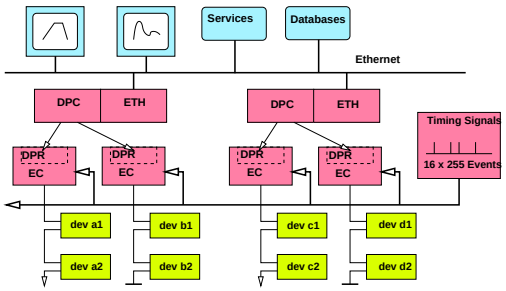

Figure 1: Hardware layout of GSI control system.

Different to many other systems; the front end side is implemented as two layers, device presentation computers (DPC) as access points for operations requests, and equipment control (EC) computers to service the devices. Assigning the tasks to different processors eases ensuring precisely timed device control: Only the ECs have to implement real-time functionality. Synchronization is achieved by signals from the timing system.

One DPC serves up to nine ECs which are all located in the same VME crate. Connection to the network is by special network controllers, also located in the VME crates, which closely interact with the DPC. ECs and DPCs communicate by common memory on the ECs, and by VME interrupts from the ECs.

Software Characteristics

The control system models the equipment as devices with *properties* instead of independent channels. A property is an action to be executed on a device, generally associated with data: Setting or reading values, or executing a function like reset.

Devices are identified by unique names, the nomenclatures. Names are also used to indicate the properties. Tab. 1 shows some properties of a magnet's power supply.

Each property is implemented as a function on the DPC

Table 1: Properties of a Power Supply

Name	Access	Data	Description
CURRENTS	read/write	1 float	set value
CURRENTA	read	1 float	actual value
POWER	read/write	1 int16	mains power
VERSION	read	48 char	version of software
RESET	call	–	reset

level, a user service routine (USR). When a property of a device is accessed, the corresponding USR is executed. A central manager on each DPC handles all devices with all USRs and implements the mechanism for property execution, which is calling USRs. The USRs don't interact with the equipment of the accelerator directly. They provide the data for the EC level, or initiate actions on the ECs.

A general interface is provided which is used by all operation level applications. It implements synchronous as well as asynchronous commands. A special kind of asynchronous access, called connected access, activates multiple property execution, either periodically or on selected machine events.

Several data types are supported in device access: Single values or arrays of different types, covering various sizes of signed and unsigned integers and floating point data. Declaring data type and count by descriptors allows using a single interface for all kinds of data.

Commands are specified by the device's nomenclature and the name of the property. The access interface converts the command to an intermediate format and delivers it for execution to the DPC by which the device is handled.

Status of the Control System

The GSI control system, like other software at that time, was implemented for a dedicated environment only: One type of VME boards with M68000 processor, running a real-time kernel on DPCs and ECs, and VMS on the operation level. Except for the operating system, all software on the VME level was developed at GSI. Even the network communication was build from scratch, establishing a proprietary protocol which is implemented only on the VME network boards and for VMS.

Primary focus was not on portability and sharp interfaces between components. As a result many dependencies between the parts of the system exist which make modifications difficult. Therefore, up to now newer components were only introduced when they are very similar to the original ones: VME boards with M68020 processors and OpenVMS Alphas.

Now the limitation to M680x0 VME boards and to OpenVMS, and the low functionality of the proprietary network, providing UDP functionality only, has become a heavy burden in maintenance and a severe obstacle to further extensions of the system. The DPC boards, manufactured specially for GSI, are no longer available, and the limitation to OpenVMS workstations does not cope for technological progress in the Unix and Windows area. A significant modernization cannot be delayed further, where most urgently the DPCs have to be replaced.

SYSTEM UPGRADE

Step by Step Migration

Instead of modernizing the old system it might look easier to switch to another one. But this is not a realistic option. The multi-beam operation of the GSI accelerators in which up to 5 experiments are served, on a pulse to pulse basis, with up to 3 different ion beams, needs specific handling in the control system. Interfacing the broad spectrum of equipment specifically developed for the GSI pulse to pulse switching could only be done by customized USRs and their counterparts on the ECs.

The effort to implement similar adaptations in another system would be far too high, especially since the accelerator is operated continuously with only four short shut-down periods every year. Only stepwise migration can be done, modifying in every step only small parts. This holds especially for the device specific adaptations, the USRs. They implement the specifics for the GSI devices and the GSI accelerator operation. To assure the existing functionality of the controls, the code of the USRs has to be kept. Even small modifications may have unpredictable effects.

Many hardware dependencies, as well as the close interaction with the network processor, would make porting the system core of the old DPC software extremely cumbersome. Rebuilding the central parts, including the network, was therefore preferred. Using a TCP/IP based communication allows to use the operating system support for the on-board network port and to skip the extra network board needed today. CORBA will be used as high level protocol.

Effort for rebuilding central parts of the DPC's software is acceptable. Many components which had to be build tediously in the old system are now available ready to use. Today's level of software support, like the C++ STL or CORBA, eases development further.

Architecture

A board with an up-to-date processor should be used to replace the old DPCs. The processor should use the same byte order as the EC's M68020 because it has to interpret the common memory on the EC, used for communication between the boards. Decision was for a PowerPC board[1]. Since all time critical actions are handled on the ECs, no real-time capability is needed. Therefore Linux could be chosen as the operating system. It allows comfortable software development and is nowadays well established in embedded applications.

Object oriented development is used for the newly build software. The object-like device representation in the GSI control system suggests handling devices as objects on the DPC level. The specific adaptations to the GSI environment, the USRs, can then be reused as the object's methods. A general interface to call the USRs is part of each device. Access to this interface is, by CORBA calls, possible over the network.

Reusing the existing USRs, with minor modifications, reduces the newly build parts to a general device which can be specifically extended by the given USRs, and the access mechanisms to execute a USR by calling a property. Addi-

[1] CPU 86, Microsys Electronics GmbH, 82054 Sauerlach, Germany, `http://www.microsys.de`

tionally a manager has to be provided to handle the device objects. According to configuration lists it creates device objects and surveys them during their lifetime. This device manager has to interact closely with the system core on the ECs.

Device Access and Networking

In contrast to the existing system with its central managers on the DPCs, the renovated system implements distributed command execution as part of the device objects. Each device object has a CORBA access interface which allows communication directly with the devices instead of a central device manager.

The access interface provides the functionality of the existing operating level interface. Synchronous and asynchronous commands as well as multiple property execution are supported. Properties are specified by their names. Asynchronous calls are handled in separate threads, returning the result by callback object.

Several formats of data have to be supported for command execution. Rather than explicitly using error-prone descriptors, a general data container is used. It holds data types actually supported in the control system, either single values or arrays. The container provides automatic data conversion. Items can be read from the container in any format, as long as conversion from the internal format is possible. Using dedicated methods for storage and retrieval ensures type safe handling of the various types, at least at runtime.

Access to the devices is available, via CORBA, from anywhere in the network. Locating the device, given its nomenclature, is easily achieved by name service which is generally provided with a CORBA implementation.

The new interface implements the characteristics of the old access interface. This allows to build the old interface as a wrapper around the new one. Existing operation level applications don't have to be modified, only the extended interface has to be linked. Internally the old proprietary communication may be used in parallel. It is not necessary to install the new DPC software at the same time in all 45 VME crates of the facility. Smooth stepwise migration is possible.

Connection to ECs

The new DPCs have to operate with the existing ECs. Communication between a DPC and its ECs is by a common memory located on the ECs which is accessible via VME bus from the DPCs. ECs cannot access the VME bus, information exchange is from the DPC always. ECs only can signal VME interrupts to their master DPC.

The old DPC hardware maps VME addresses in the processors address space. All software on the DPC can manipulate all resources freely, also on the VME bus. In a multi-user operating system like Linux, access to central resources is restricted to kernel level to prevent corruption of the system by faulty applications. A driver for the VME bus, as well as a kernel module to handle interrupts, are provided with the board. To provide EC interrupts to the applications, a driver was developed.

A rather obscuring part of the existing front end system is the connection between DPC and ECs. Used by many units, it is implemented as one big structure where each application uses its own region, which in turn is a specific structure. Each unit which uses the common memory has to know the full declaration to be able to access the part it uses. The layout must be interpreted identically on DPC and EC. Since both boards use the same processor, this was achieved by using the same declaration and the same compiler for both.

Using identical declarations and the same GNU C/C++ compiler for EC and DPC, the PowerPC variant introduced extra padding bytes to improve memory access. An identical layout could be achieved by forcing both compiler variants to generate naturally aligned memory layout. Nevertheless, dependencies from using complex structures which have to be interpreted identically in all software components cannot be handled in the long term.

To decouple the modules, the common functionality of the EC is encapsulated in classes. Only theses classes are used by the new system core on the DPC. The memory layout is hidden completely.

Unfortunately, similar dependencies are implemented in the USRs. Here, at least, working with a reference of the device's specific data only instead of the common memory as a whole reduces the dependency.

Device Implementation

Devices on the DPCs are characterized by their USRs which implement the properties, and specific data to hold the device's state. Device specific data are stored in the common memory area on the ECs which is accessible from the DPCs. The common memory acts also as link to the USRs counterpart on the ECs which handle the equipment to be controlled.

All USRs are implemented as functions with identical parameter list. To re-use the existing code it is transferred to a standard method, either read, write or call, of new Usr classes, one for each USR. Each device object holds a list of its Usr objects, which is filled when the device object is created. Each Usr object stores the name of the property it represents. When a command is executed, the responsible Usr object is searched according to its name, and its read, write or call method is executed. No specific knowledge about the device is needed to determine and execute a read, write or call method of an Usr class. Command execution is, for all devices, inherited from one base class VmeDevice. Handling the list of Usr objects, and retrieving a specific one is easily implemented using the set-template from the STL.

Modifications in the old USRs mainly affect the parameter list. USRs worked on the common memory as a whole. This memory area contains an array of device data, one el-

ement for each device handled by an EC. The data for the specific device is referenced by an index in this array, provided when the USR is called. Each Usr object in contrast operates on the device's specific data only via a pointer to this data, set when the device object is created. Instead of interpreting the whole common memory, Usr classes need only the knowledge of the device's specific structure. This is another step to decouple the implementations on DPC and EC.

A second modification affects the data associated with the property, e. g. the set value for the current. To support different types in the unique parameter list, data are exchanged by an unsafe void pointer. Downcasts inside the old USRs restore the original types. Usr classes operate with the data container used also in the operation level interface. Explicit methods for storage and retrieval ensure correct usage of data types.

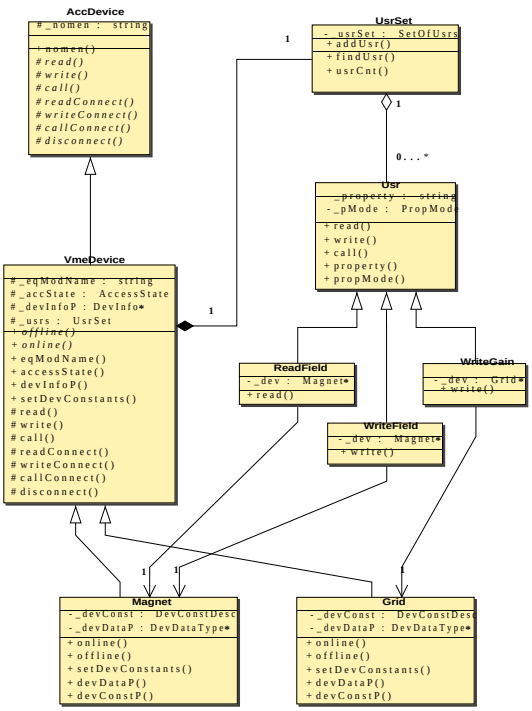

Figure 2: Object oriented device implementation.

FIRST EXPERIENCE

The new board is supplied with a pre-configured Linux operating system. Even with modest experience with embedded Linux it was possible to get the board operational in very short time. Also a development environment, with all board specific adaptations including VME driver and interrupt handling, was delivered [3]. After becoming familiar with the new tools, development of the specific applications could start rapidly. All software development, and boot and file service for the new DPCs is done on standard PCs under RedHat Linux.

To integrate the new boards in the old environment, a lot of details had to be solved. It took some time to become familiar with new technologies, like CORBA, and the Linux environment. After passing the learning curve, progress was rapid. Working software to study special aspects of the new system could be developed fast. By now solutions for all critical points could be demonstrated. Putting all building blocks together, a first implementation in a test environment could be completed. The new DPC components are connected to the existing EC level, and the operations level interface offers old and new front-end installations uniformly.

The full functionality of the old system is not yet implemented, and not all error conditions are handled properly. But the feasibility of the proposed renovation could be demonstrated.

OUTLOOK

The base version of the newly build system will be successively extended to full functionality. It has to be tested extensively, with a final test by replacing an old DPC board in the accelerator installation. When the system proves to be stable, the new DPC boards will replace the old M68020 boards step by step in all 45 VME crates installed in the facility.

With the new DPC board spare parts will be available again for the next years. A main design objective of the new system is to encapsulate dependencies of hard- and software. It will be therefore much easier than before to introduce a then modern hardware when the actual DPC board is no longer available.

An even more valuable result of the renovation is the replacement of the network software. After using a proprietary protocol for a long time, the network will be based on a widely used standard in the future. This is an important step to overcome the limitation to OpenVMS for the operation level software. Integrating Linux, and Windows, and any other system for which CORBA is available will be possible easily. A broad range of software can then be made available for the operations of the accelerator.

On the front-end side, integrating other standards than VME will be possible too. As long as the systems provide the CORBA device access interface, they can be accessed directly. Additionally, it will be much easier than today to connect other control systems to the GSI system by bridges or gateways.

REFERENCES

[1] U. Krause, V. R.W. Schaa, "Re-Engineering of the GSI Control System", ICALEPS'01, San Jose, Nov. 2001, `http://icalepcs2001.slac.stanford.edu/`

[2] L. Hechler, "Converting Equipment Control Software from Pascal to C/C++", ICALEPS'01, San Jose, Nov. 2001, `http://icalepcs2001.slac.stanford.edu/`

[3] Embedded Linux Development Kit, ELDK, Denx Software Engineering, `http://www.denx.de`