EXTENDEND APPLICATION FIELDS FOR THE RENOVATED GSI CONTROL SYSTEM

K. Höppner, L. Hechler, K. Herlo, P. Kainberger, U. Krause, S. Matthies, GSI, Darmstadt, Germany

Abstract

The current GSI control system uses a very monolithic approach that made it difficult to extend the system to other than the original platforms (VME front ends and OpenVMS on the application level). For the present renovation project of the communication layers, flexibility was a major design criterion. Front-end and application levels are connected via CORBA middleware, giving free choice for using various system architectures and programming languages on both levels. While most of the current front-end software will be ported to the existing VME front-end environment, now running Linux, the new system can integrate devices running on various architectures and operating systems into the new GSI control system. To model equipment functionality as independently as possible, generating adapter code from a well-defined XML description of device models is now under development. This will make the task of porting the existing 65 device models (including around 3000 properties) to the new modular approach easier.

INTRODUCTION

The GSI control system was commissioned nearly 20 years ago. Since then it was extended step by step to cope with the continuously refined usage of the GSI accelerators. In regular operation, three ion sources serve up to five experiments in pulse to pulse time sharing mode, allowing varying setups according to the different experiment requests as well as providing a set of fixed beams for cancer irradiation.

The high degree reached in operation demonstrates the capabilities of the system outline. However, fulfilling the growing demands becomes more and more difficult. Rigid structures severely hinder to keep up with the technological progress. Future work therefore has to spend special attention to provide sufficient flexibility in the system.

GSI CONTROL SYSTEM

Outline of Control System

The GSI control system is designed as a decentralized distributed system, according to the well established standard model (fig. 1): Operation level workstations for man machine interaction and data generation, connected by network to VME front-end controllers for device handling.

A characteristic of the GSI system is to use two hardware layers in the front-ends. Real-time Equipment Controllers (EC), synchronized by the timing system, assure precisely timed equipment handling with an accuracy down to several microseconds. Remote requests via the network are handled in separate Supervisor Controller (SC) boards which don't need any real-time capability.

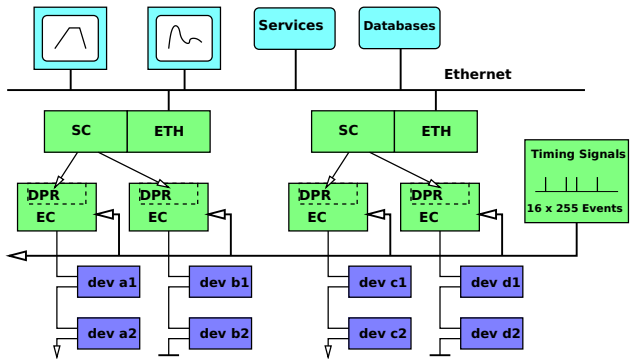

Figure 1: Outline of GSI control system.

Status

Not unusual at that time when the system was developed, focus was on over-all functionality only. To provide a tuned system, most of the software, even the network protocol, and the equipment interfacing hardware, were developed in-house. Tailoring to the underlying platforms has lead to a system in which the components of the the operation level and the front-end level are tightly interwoven.

Modifications of the core components are very cumbersome because of the coupling and had to be avoided in the past. As a result the system is implemented only on the platforms used during original development: OpenVMS workstations and specific M680xx VME boards.

The SC boards are no longer available and need to be replaced urgently. Porting to similar boards would be possible, in principle. However, this would conserve the rigid structures further on. Therefore a more general renovation was started: Completely rebuilding the communication layers in the control system.

RENOVATION PROJECT

Strategy

The new communication layers have to implement a similar functionality as before, to allow usage with the existing operation's applications. Using nowadays available middleware, this can be done with much less effort compared to the existing implementation with its proprietary components.

Decision was made to use CORBA since it straightforwardly links applications in different programming lan-

guages including scripting languages like Python, on different operating systems. Being well standardized, many commercial CORBA implementations are available as well as OpenSource developments.

Implementation Outline

The system software on the SCs was developed newly. It offers a device model similar to the former one: Independent devices with properties reflecting the characteristics of the equipment. Access to equipment means calling a property for a specific device.

A central device manager establishes the device objects. Installation is on PowerPC VME boards running Linux.

Presently, about 65 different device types have to be supported, implementing about 3000 different properties. Being the major part of the SC software, the code of these properties can be re-used in the new environment with small adaptations. The existing EC installation, soft- as well as hardware, can be integrated nearly unchanged.

The existing procedural client interface is extended so that it can also handle the new device implementation. Both device front-end installations, the old and the new, can be used in parallel to allow a smooth step by step migration on the front-end side. For future use, an OO client interfaces is provided. Presently C++ and Python are supported, while a Java interface is under development.

Status

The implementation has reached the state were it can be used under real-life conditions. It is installed in a first location in the accelerator and showed the usability of the underlying approach in a first experiment's beam time. With this new SC software the control system overcomes the former limitations to OpenVMS and dedicated VME boards on the front-end side.

EXTENDED APPLICATION FIELDS

Non-VME Devices

The new device presentation layer was initially developed for usage on the PowerPC SC boards. It sits atop of the existing ECs which handle the equipment control, and the interfacing to the equipment. Test of the system requested installation on a VME crate, equipped with at least one EC.

To provide a simpler test environment the SC software was ported to a standard x86 Linux PC. This was achieved in short time by using several quick and dirty short cuts to remove the interaction with the ECs .

Success with this fast implementation encouraged to rework the central SC software, which is device manager and device model. Dependencies on the underlying EC software were separated. Equivalents for stand alone operation, without underlying ECs, were developed.

By simple makefile switch, the device manager can be build for the GSI VME environment as well a for a standalone Linux installation. This was already helpful in an early stage where prototypes of Java GUI applications were developed by Cosylab and could be tested easily in Ljubljana.

M-Box

The x86 implementation in first stage provided only dummy properties. Because up to now all equipment in the control system is connected via EC controllers, the x86 version was not used for equipment handling.

However, interfacing to equipment can be integrated easily into the properties. This allows using non-VME controllers as front-end node in the GSI control system.

First usage is for a stepper motor driver, to position the new septum in the GSI synchrotron [1]. Because of its capability of driving several axes simultaneously, Cosylab's M-Box was selected to drive the motors. It integrates a PMAC stepper motor controller onto an MicroIOC, which is an embedded Linux computer.

Since the device manager depends on rather general resources only, it could be straightforwardly build for the MicroIOC. Driver functions provide access to the PMAC stepper motor controller. The septum actuator is presented in the control system identically to the devices installed on the traditional VME crates.

Windows

For a long time, Windows operating systems were not used in the control system. However, for many experts it is their preferred working environment, and in the commercial area many drivers are available for Windows systems only. It is therefore desirable to support Windows in the control system.

When a new team member joined us, who had worked mainly in the Windows environment before, we started with the first steps towards an integration.

The client interface, which is only lightly depending on operating system functionality, could be ported and build with Microsoft Visual C++ in short time. Work was done by a newbie in control system technology. By this port, Windows clients have access to devices in the control system.

Encouraged by this fast success, porting of the device manager itself was started. Again a re-work of the code had to be done, now by identifying system dependencies. These were caused by the different approaches on interprocess communication (e. g. message queuing) and system logging in Windows and Linux. We created a library with a common layer for operating system dependent calls, providing the same interface for message queuing, signal handling and system logging for Windows and Linux. Threads are based on omniORB threads which define a common wrapper for thread programming on several systems. The

library contains a set of functions providing some Unix functions that were non-existing under Windows.

After about two months work, the device manager is functional on Windows, too.

CODE GENERATOR

While the device manager can be installed easily, writing the device specific implementation can be cumbersome. A set of properties has to be implemented to represent the specifics of the device.

On the other hand, clients must use the properties correctly. To assure that both sides fit, the implementation should be derived from a common interface definition.

We invented a XML Schema Definition for GSI equipment models. In present, C++ header files and source skeletons for device servers are created from the XML representation of the equipment models (cf. listing 1 for an example) using a XSLT style sheet processed by XalanJ. The class definitions for the device properties and the prototypes for the action methods (read, write and call) are created automatically (fig. 2). This includes the in-code documentation of property data and parameters using Doxygen, supporting the generation of API documentation in HTML and LaTeX format.

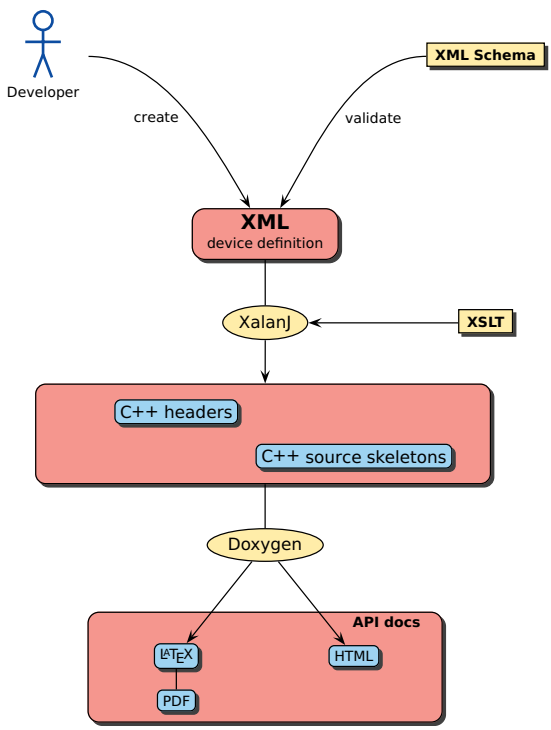

Figure 2: Workflow for the generation of C++ sources and API docs from XML device definition.

Listing 1: Example of a XML device definition

```
<eqmod>
  <name>MX</name>
  <creator>KlausHoeppner</creator>
  <version>09.01.01</version>
  <description>Multiplexed Dipole
                    Magnet</description>
  <variant id="1">PERMANENT_SIS</variant>
  <variant id="2">SHARED_SIS</variant>
  <variant id="3">PERMANENT_UNI</variant>
  <property category="master">
    <name>INIT</name>
    <description>Initialize</description>
    <action type="call" medlock="all"/>
  </property>
  <property category="slave">
    <name>CURRENTS</name>
    <description>Current Set
                      Value</description>
    <action type="read" medlock="none"/>
    <action type="write" medlock="vrtacc"/>
    <data type="Float32">
      <value name="current">Current (Set)</value>
    </data>
  </property>
</eqmod>
```

CONCLUSION

The renovated front-end software with its new network communication is now ready to be used in GSI's accelerator controls. Intentionally developed for Linux PowerPC VME boards to replace the outdated SC controllers, it can be easily used in other environments, too.

With the new device manager a flexible controls framework is available. Being restricted to VME front-ends in the past, the GSI control system now allows integration of other front-end platforms, too. The M-Box stepping motor controller is a first step towards well adapted solutions for future demands in the control of accelerator equipment.

REFERENCES

[1] J. Dedič, J. Bobnar, I. Križnar, R. Šabjan, R. C. Bär, G. Fröhlich, K. Herlo, U. Krause, M. Schwickert, "Customizable Motion Control Solution Supporting Large Distances", ICALEPCS'07, October 2007, Knoxville, WPPB18, http://www.sns.gov/icalepcs07/.