THE CONTROLS MIDDLEWARE (CMW) AT CERN STATUS AND USAGE

K. Kostro, J. Andersson, F. Di Maio, S. Jensen, CERN, Geneva N. Trofimov, IHEP, Protvino, Moscow Region

Abstract

A new Controls Middleware (CMW) for the "LHC era" has been recently designed and implemented to serve the CERN accelerator sector. It has now been used for almost two years in the operation of the PS accelerator complex and is being introduced for the control of all the upcoming LHC equipment as well as for the existing SPS equipment.

This paper presents the architecture and capabilities of the system and shows how it has been integrated in the existing controls environment. The use of publish/subscribe paradigm, the performance of the system, and administration facilities are described as well. Based on the experience with CMW we also discuss the validity of choices, which were made almost four years ago.

INTRODUCTION

As part of the preparation to control the LHC it has been requested that a new middleware is put in place to replace the existing communication systems, notably the remote procedure call system developed around 1988 and successfully used for many years, augmented with equipment access standards.

More specifically this new middleware shall support OO development, particularly in Java; it shall offer publish-subscribe facilities in addition to synchronous equipment access and a better connectivity to industrial systems.

Originally two separate technologies were selected to serve as the base for the controls middleware: CORBA and JMS. The choice of CORBA was to support multilanguage and multi-platform inter-operability. The choice of JMS was motivated by the availability of commercial products and strong involvement in Java. JMS is today mainly used internally within the J2EE platform and we will not discuss it here.

ARCHITECTURE AND COMPONENTS

CMW is structured as a client/server model. At the heart of CMW is the Remote Device Access (RDA) system [1], which defines the client and the server API and provides the communication on top of CORBA.

The Java control programs constitute the main category of CMW clients. A C++ client API is provided as well and used mainly to build gateways. A VB/Excel API is also available for rapid prototyping.

At the server level a significant effort was made to connect existing controls equipment. This required multiple server developments due to heterogeneous equipment access methods, which had to be covered. Naming and Configuration services were developed within the CMW to support the device name resolution and to allow server configuration from existing databases. Finally, administration and diagnostics utilities were developed to be able to survey the status of servers and to rapidly diagnose any faults.

Figure 1 gives an overview of CMW components.

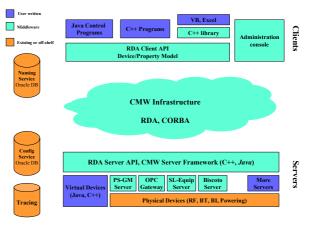


Figure 1: CMW Architecture and Components

The device model and the RDA

The Device Model has been traditionally used in the PS and SPS control systems. Within this model the control system consists of **named devices**. The devices can represent actual physical device such as Position Monitor or can represent virtual entities such as Beam Line. Each device belongs to a **Device Class** and it is the Device Class that defines the properties, which can be used to access the device. By invoking a get() on the device with the property name, the value of this property will be read. The following sequence of Java code illustrates this:

```
DeviceHandle bpmDevice =
    rda.getDeviceHandle("BPM1");
Data result = bpmDevice.get("Position");
```

Similarly by calling a *set()* method on the device, the value of the property (for instance the gain) can be set.

In addition to the get and set operations, CMW allows a property to be **monitored**. When a user invokes *monitorOn()* on a device, the updates to the value of the property specified in the call, but also any exceptional events will be delivered to the listener:

```
BpmHandler listener = new BpmHandler();
BpmDevice.monitorOn("Position", listener);
```

Similarly by calling a *set()* method on the device, the value of the property (for instance the gain) can be set.

In addition to the get and set operations, CMW allows a property to be **monitored**. When a user invokes *monitorOn()* on a device, the updates to the value of the property specified in the call, but also any exceptional events will be delivered to the listener:

BpmHandler listener = new BpmHandler();
BpmDevice.monitorOn("Position", listener);

The BpmHandler class *implements the ReplyHandler interface* i.e. it provides the actual implementation of the method, which will handle updates to the value of the *Position* property. Methods that handle device I/O errors and any exceptional events such as disconnections are also part of this interface.

The Device Access Model has been implemented as the Remote Device Access (RDA) system. RDA provides both synchronous and asynchronous versions of the *get()* and *set()* methods. On the server side the developer has the possibility of implementing only the synchronous versions with RDA providing default synchronization. Although the device model does not explicitly provide the possibility for a set-and-get operation, the get call can carry a filter by which get condition can be specified. We are trying to standardize and limit utilization of filters to preserve the generic aspect of the device access model.

RDA is implemented on top of CORBA with *two-way* calls being used for synchronous and *one-way* calls for asynchronous get/set operations and for monitoring. RDA provides an elaborated mechanism for detection of connection failures and automatic reconnection. The description of RDA, the rationales behind it and the details of the implementation are described in [1].

All calls can take *cycle selector* as a parameter. The cycle selector restricts the applicability of the operation to a specific cycle of cycle type. This reflects ability required in the accelerator control systems to work with a specific cycle (e.g. first proton cycle) or with a specific "virtual machine" of the PS accelerator complex. Thus the cycle selector constitutes a sort of filter, especially useful to specify subscription conditions. CMW does not assume anything about the nature of cycle selectors. In the monitoring a *polling period* can be specified instead of a cycle selector.

Narrow API and self-describing data

A deliberate choice was made in CMW to use a *narrow API* for the device access. As it has been shown in the example the property name is specified in the *get()* method. The alternative would be to offer a *wide API* i.e. provide *getPosition()* method to get the value of the Position property. There were several reasons for this choice. First of all the previous experience with the Remote Procedure Calls (RPC) has shown that the diversity of methods to access equipment creates more problems than benefits. Later on a generic device access was created on top of the RPC. Secondly the wide

interface would imply a huge initial effort required to generate the code required to access the existing devices. And last but not least it is much easier to develop generic applications and gateways based on a narrow API.

To nevertheless enforce the maximum of consistency checking, the types of properties are checked at runtime. For instance setting the gain as an *integer* will generate a runtime error if the expected value was a *double*.

A similar decision was made for property values. The value of the property is encoded as a *Data object*. The Data object is defined by CMW and allows transport of self-defining data in a language-independent way. The Data object serves as a container for one or more DataEntry objects. Each DataEntry can hold a scalar value, a string or an array of these. The data object carries the names of DataEntry tags with it. This generates a small overhead but it is very useful for interpretation of data in generic clients.

System administration and diagnostics

Good diagnostics and administration facilities are essential in a distributed system. In the CMW the system administration is part of the system requirements and a considerable effort has been invested into administration and diagnostics support in the servers as well as the accompanying tools.

Administration facilities have been defined in CORBA as a dedicated *admin* interface. All CMW servers implement this interface (RDA servers and also directory and database servers). The admin interface allows to interrogate the status of the server, collect server statistics, set tracing levels and even to restart the server.

Based on this interface, the **CMW Management Console** has been developed. In the survey mode the console displays the overall status of the server, decoded as the colour of the corresponding button. When a server is selected, detailed information about the server, its configuration, client information and statistics is available. Trace levels can be remotely changed to enable server diagnostics. Figure 2 shows a screenshot of the Management Console.

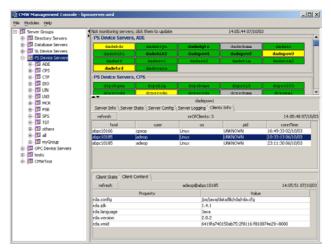


Figure 2: CMW Management Console

When a problem is discovered in connection with a device access, it is useful to be able rapidly to access the same device to verify the existence of a fault and study the behaviour. For this reason a **Device Explorer** has been developed, which allows browsing the device name space, discovering the available properties and exercising the access to device both in get and in subscribe mode.

To diagnose equipment access problems, the CMW logging and tracing system can be used as well. All servers are able to log the trace of what they are doing with various level of detail. Log levels for areas of concern (get/set calls, subscription) can be enabled from the Administration Console and results can be consulted in the log file. Errors are normally logged all the time.

DEPLOYMENT AND USE OF CMW

One of the prerequisites for replacing existing communication methods was the possibility to access all the existing equipment via the CMW. This has been accomplished and today virtually all accelerator equipment can be accessed this way. Currently the CMW is used mainly from Java. All new developments at the application level are today in Java and the device access is made through the Java RDA API.

This process has started with the control of the Antiproton Decelerator (AD) in March 2002. Recently the commissioning of the beam extraction from SPS to the LHC was performed almost exclusively with Java applications and equipment access through the CMW.

Use of the subscription facilities

Use of the publish/subscribe paradigm is new in the CERN controls environment and users as well as providers of this facility had to learn how to use it effectively. First of all the existing equipment access methods and equipment servers were not made to support subscription so that the CMW servers have to poll equipment first, to be able to push updates to the subscribers. RDA offers the possibility of subscribing "on change" i.e. updates are delivered only if the property value has changed. This facility is not available in existing device access libraries and has to be implemented in CMW servers. The first implementations of the monitoring were not delivering the initial property value, a facility, which is essential for effective use of subscription on-change.

Today the update on-change and initial value delivery are systematically supported in all CMW servers. All properties are systematically delivered with the timestamp (UTC time in nanoseconds) and/or the unique cycle identification. This cycle identification makes it possible to correlate different updates as belonging to the same cycle.

In the future a better support for the subscription will be built into the equipment control software so that it will be the real-time task, which will notify the CMW when a property has changed in a significant way.

Developing and deploying CMW servers

RDA provides basic features for the server construction and a CMW server can actually be build using the RDA library. However, a typical server implementation requires additional features, mainly to support subscription updates. For instance the device has to be polled at a given time in the cycle, driven by timing events. Polling the device more then once for the same property/cycle combination must be avoided. The classes, which implement these facilities, form the Device Server Framework (SFWK).

Typically equipment control software runs on a realtime platform (LynxOS) as a high priority process (realtime task). The CMW server communicates with realtime tasks via an internal mechanism – shared memory or message queue. To develop a CMW server an adapter has to be developed, which is specific for a class of equipment (for instance Beam Instrumentation equipment access). Once the adapter is available, equipment-specific servers can be generated more or less automatically. Typical server components are shown in Figure 3.

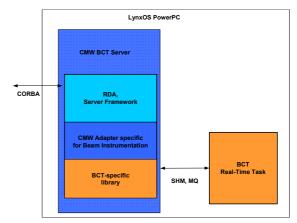


Figure 3: Beam Curent Transformer (BCT) Server

In the past a number of different CMW servers were developed to cover access to existing equipment. Recently an effort has started to design and develop the universal equipment server framework for the accelerator controls equipment servers [2]. We are participating in this effort to define the optimal architecture for communication via CMW, notably for subscriptions.

Table 1 shows the platforms on which CMW servers are or will be deployed.

Table 1: Deployment of CMW servers

Platform	OS	compiler
PowerPC, VME	LynxOS 3.1/4.0	gcc 2.92
Intel, PCI	LynxOS 4.0	gcc 2.95
Intel, PCI	Linux	gcc 2.96
Intel, PCI	Windows 2000/XP	MSVC++ 6.0
Java	JDK 1.3/1.4	JDK 1.3/1.4

Naming and configuration services

In the device/property model the control system is perceived as an ensemble of named devices, which can be controlled via properties. Device names are unique within CERN. To find resources allocated to devices, a device directory is required.

CMW servers can often be developed as generic servers configured with equipment information already available from the database, which greatly simplifies maintenance and deployment.

In CMW both naming and configuration services were developed as CORBA servers in Java, connected to a database via JDBC. These servers are running directly on the database computer or on a dedicated Java server. They also implement the administration interface mentioned before and can be surveyed as any other CMW server.

CMW clients

Most of the CMW client access is from Java, either directly from Java GUI applications or from the middle tier, the *accelerator business logic* being developed in Java. But Java cannot be easily integrated with other, non-Java systems so that the availability of a C++ client interface is also essential. The C++ client API was used to develop a Visual Basic and Excel interface to the control system (the so-called *passerelle*). This facility is very much appreciated by the operators and used for rapid prototyping and machine development.

CMW performance

Given the modern and complex technology, which we are using in CMW (CORBA, Java, multithreading, etc), we had some doubts about the final performance of the equipment access. It turned out that the performance is absolutely adequate, often better than the previous equipment access methods. In table 2 we give some performance figures for a synchronous get() call returning a single scalar value. The first row shows the actual access time, including the network overhead, which we experience on operational servers and which include the reading of the equipment property. The two other measurement results were obtained with a test server and give a reasonable estimate of what can be expected in the LHC era in terms of communication overhead.

F F F F F F F F F F F F F F F F F F F		
Client	Server	Synch. get()
800 Mhz Intel,	LynxOS, 175 Mhz PPC,	4.5 ms
Java	PS Equipment Server	
2400 Mhz Intel,	LynxOS, 400 Mhz PPC,	0.5 ms
Java	RDA test server	
2400 Mhz Intel,	Linux, 2400 Mhz Intel,	0.16 ms
Windows, C++	RDA test server	

Table 2: RDA performance of a simple get() call

CONCLUSIONS

In summary the experience with developing and using CMW is very positive. The use of Object-Oriented programming allowed us to have a very stable product from the very beginning. The execution performance is

more than adequate. We were able to offer the publish/subscribe mechanism, which is being increasingly used. The CMW has been deployed rapidly and integrated with the existing controls infrastructure.

The implementation and use of the publish/subscribe paradigm required some adjustment, notably grouping of updates was introduced to satisfy the performance requirements.

The main difficulties, which we encountered in development and maintenance of CMW, are related to the use of C++. The CORBA implementation, the RDA and the CMW servers require a recent, good quality C++ compiler. Especially on our base Front-End operating system – the real-time LynxOS system, the deployment of RDA was only possible on one of the recent releases of the system. C++ executables, which use complex libraries, tend to have a large footprint. This is not a problem on modern PCs but it is limiting the deployment possibilities on VME systems introduced in the 90-ties. Since shared libraries are still not available on our LynxOS systems, the deployment of CMW servers is often limited to one per CPU.

Apart from the memory problem, the experience with CORBA was very positive. Thanks to CORBA we could use an OO approach across the communication layer. When additional services, such as database access, are required we can develop them as CORBA servers, which greatly simplifies the development task.

FUTURE WORK

Although the CMW infrastructure itself is ready, additional developments are necessary to support the upcoming LHC infrastructure and to continue the deployment on the new LHC platforms.

To complete the connectivity we have to provide a bridge between CMW and PVSS – the major SCADA system used at CERN. The new Front-End equipment framework [2] will require the development of a new CMW server. Some consolidation work is still necessary, notably in the area of Naming Services where we would like to establish a common device directory for all controls equipment and in the area of access control, which requires the establishment of access rights.

The use of CMW can be improved by introducing standards for server behaviour. For instance a standard for data reduction filters would allow to specify which subrange of array has to be transmitted, without the need for defining a specific property for this.

REFERENCES

- [1] N. Trofimov, V. Baggiolini, S. Jensen, K. Kostro, F. Di Maio, A. Risso, "Remote Device Access in the New Accelerator Controls", ICALEPCS '01, San Jose, USA, 27-30 Nov. 2001.
- [2] M. Arruat, A. Guerrero, J-J. Gras, S. Jackson, M. Ludwig, J-L. Nougaret (CERN), "CERN Front End Software Architecture for Accelerator Controls", This conference.