CONTROLS MIDDLEWARE - THE NEW GENERATION

K. Kostro, V. Baggiolini, F. Calderini, F. Chevrier, S. Jensen, R. Swoboda, 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 is object-oriented and supports two conceptual models: The Device model traditionally used in accelerator controls and the Topic model, which is widely used in the commercial world. Unlikely previous middleware solutions, developed in the 80's, this middleware is completely build upon commercial standards: CORBA for synchronous calls, JMS (Java Message Service) for message passing and OPC (OLE for Process Control) for integration of industrial equipment. The subscription paradigm is supported in both Device and Topic models. This paper presents the system architecture and an overview of the user facilities and API's. The experience with CMW, the issues of integration of CORBA, JMS and OPC and system administration are addressed as well.

1 MOTIVATION

The communication base of the control systems at CERN (PS, SPS and LEP) was the remote procedure call system developed around 1988. This system was successfully used for many years, augmented with equipment access standards.

But recently new needs become apparent: Increasing use of industrial equipment called for a better connectivity to industrial standards. Java has become popular and Java GUI's became "state of the art" supplanting Motif GUI's. In addition there was call for publish/subscribe facilities, which are commonplace in commercial world. All this was impossible or difficult to support with the existing infrastructure.

The Controls Middleware project [1] was established in 1999 to capture requirements, choose an adequate middleware and provide implementation. The goal was to create a new generation of middleware, which will be adequate for the "LHC era".

2 SYSTEM ARCHITECTURE

The CMW project started with capture of the requirements based on consultation with equipment and operation groups. As result, not only the requirements for the communication were established, but also these of naming and configuration services as well as the administration tools.

As the first step an evaluation of middleware products and technologies was carried out and two technologies were retained: CORBA and JMS. The choice of CORBA was to support multi-language and multi-platform inter-operability. The choice of JMS was to support loosely coupled system communication and hub-and-spoke architecture.

The **Device Model** is the mainstream of the CMW and it has been implemented as RDA [2] using CORBA. The **Topic Model** has been implemented as wrapper around Java Messaging Service (JMS). At the client side a thin layer of the **Device API** is defining an implementation-independent interface and unifying the subscription capabilities of both models.

A deliberate choice was made to restrict data, which is passed between the various components of the system to a generic data container: the CMW Data object. This data object is used in device access but also in database access and administration tools. It can also be transported as a topic by JMS.

The following figure shows the overall architecture on CMW. The important aspects of the system will be presented later in some detail.

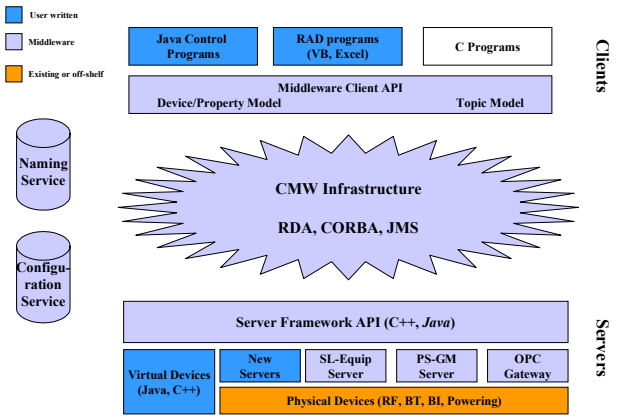

Figure 1: CMW Architecture

3 SUPPORT OF THE DEVICE MODEL

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, which 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:

```
DeviceBase device = new DeviceBase("BPM1”);
cs = new DefaultCycleSelector("211e0101");
Data result = device.getData("Pos", cs);
```

The value of the property is returned as a Data object. The Data object is defined by CMW and allows to transport self-defining data in a language-independent way. The data type is checked at runtime therefore enforcing the consistency between the device implementation and the device client.

The cycle selector used in the example demonstrates the possibility of specifying which “slice” of the property should be returned. This reflects ability required in the accelerator control systems to work with a specific cycle or with a specific “virtual machine” of the accelerator. In this example the timing event of SPS is specified, but CMW does not assume anything about the nature of cycle selectors, which are transported as strings.

The object-oriented design of the Device API has the consequence that functionality of classes such as DeviceBase or CycleSelector can be easily extended with more specific device or accelerator behaviour.

Other methods which can be called on a device: set() to set value of a property and monitorOn() to initiate reporting of value changes of a property, follow the same principle.

This model has been implemented as the Remote Device Access (RDA) package. Currently Java and C++ implementations of RDA exist for the server and Java implementation exist for the client. C++ client implementation is being developed. We will not describe RDA here it as it has been already described in detail in [2].

RDA provides basic features for the server construction and a CMW server can actually be build using RDA library. However, a typical server implementation requires additional features, mainly to support subscription updates. For instance device has to be polled at a given time in the cycle, driven by timing events. It should be avoided to poll the device more then once for the same property/cycle combination. The classes, which implement these facilities, form the Device Server Framework (SFWK). Figure 2 shows typical components of a CMW server on the example of the SL-Equip server.

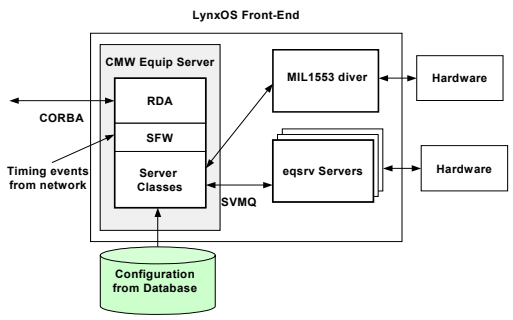

Figure 2: CMW SL-Equip Server

4 SUPPORT OF THE TOPIC MODEL

The Device model assumes that clients have knowledge of existing devices and properties. For some systems the Topic Model is better suited. In this model the clients do not know all possible suppliers of information but can tune to a *topic*. The CMW project has chosen Java Messaging Service (JMS) to support this model.

The example of the Alarm System shows that the Device Model is not always adequate. The Alarm System does not necessarily have knowledge of all devices, which can possibly issue alarms and therefore cannot subscribe to alarm properties of these devices. Devices can be added, removed or modified without the alarm system knowing about it. In addition the condition of alarm can depend on the mode of the accelerator so that the entity, which issues alarms, should have knowledge of alarm categories, which are active. All this does not fit well into the Device Model but it fits into the Topic Model: Alarms can be published as a *topic* according to the alarm category tree spontaneously. The alarm library can subscribe to the machine mode so it knows which alarms should be suppressed. And finally any number of alarm consoles can be connected without altering the performance of the alarm system. Alarms can be subscribed to by the console, according to a hierarchy used in alarm topics. Figure 3 shows the architecture of a prototype implementation.

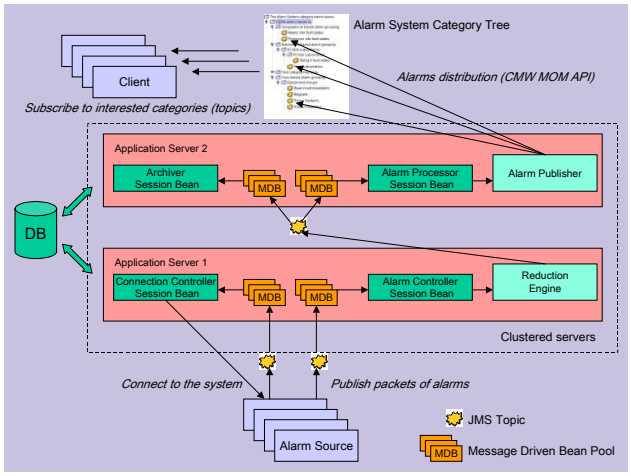

Figure 3: Use of Topic Model in Alarm System

5 SYSTEM ADMINISTRATION

In CMW the system administration has been part of the system requirements and a considerable effort has been invested into administration tools and infrastructure. One important product is the **Management Console**, which combines the tools essential for system administration.

Administration facilities have been described as CORBA *admin* interface. All CMW servers implement this interface: RDA servers but also directory and database servers. The **Server Explorer** facility of the

console is using the *admin* interface as a client. Servers are continuously monitored and the general server state is displayed, green if the server is in a good state, red if bad, etc. When a server is selected, a number of buttons can be used to display the detailed server state, statistics, configuration and information about clients. Servers can be shutdown or restarted as well. Figure 4 shows a screenshot of the Management Console.

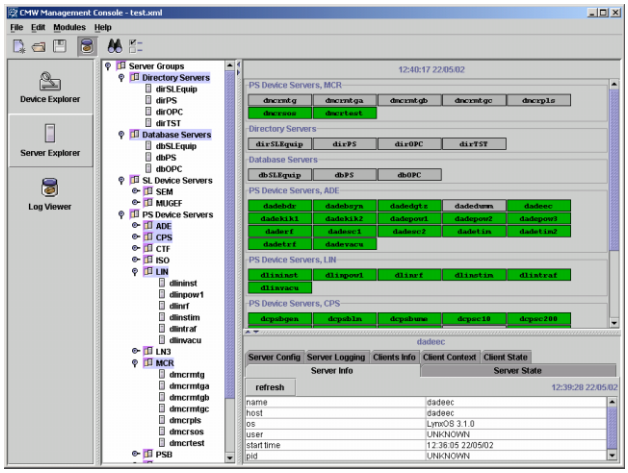

Figure 4: Server Explorer in the Management Console

When a problem is discovered in the access to a device, the system administrator often has to try to make the device access to understand the reason of the problem, which may lie in the CMW server or in the equipment layer. 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. Simple ad-hoc device display applications can also be made using the device explorer.

A distributed system is often subject to failures, which are more difficult to analyse than it is the case of isolated systems. It is therefore important to log all errors and warnings to a central place, which can be consulted in case of problems. It is also often useful to be able to enable a "verbose" mode, to trace unexpected behaviour and direct the trace to a file or to a console. In CMW this is available through the logging system.

Messages are divided into well-defined categories: error, warning, information etc. Each category can be handled separately. CMW adds information such as origin and time to each log message. Message destination is determined via the logger configuration. Messages can be sent to a central file or to a local console. This functionality is provided by Log4J, which has been adopted as the logging system infrastructure.

6 NAMING AND CONFIGURATION SERVICES

In the device/property model the control system is perceived as ensemble of named devices, which can be controlled via properties. To find resources allocated to devices, a device directory is required. Similarly servers have to know which devices to serve and how device names are mapped to hardware resources, an information typically available from a database.

In CMW both naming and database services are available as CORBA servers connected to a database. In this way database access is available from Front-End systems, which has not been the case before. Figure 5 shows the use of naming and configuration services.

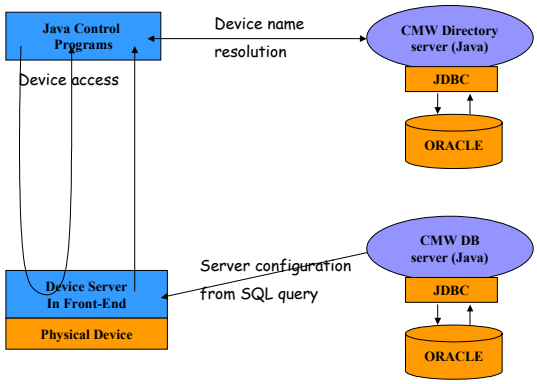

Figure 4: Naming and Configuration

7 CURRENT STATUS

The first CMW servers have been developed in Spring 2001 and the next generation of servers are in operation since Spring 2002. Currently all CERN controls equipment can be accessed through CMW. The directory and configuration services are available as well.

Recently we invested an effort to provide administration tools and facilities, which are about to be completed and integrated in the severs.

Currently the project concentrates on consolidation, developing the C/C++ client API and performance improvements, notably by grouping requests. So far the CMW servers have been developed by the CMW project itself to connect "legacy" device servers. New server developments for the LHC will have to be developed by the equipment groups. An infrastructure for access control is being developed and will have to be integrated into servers.

The experience has been positive so far and we are looking forward to exploit the full potential of the CMW in the LHC era.

6 REFERENCES

[1] http://proj-cmw.web.cern.ch/proj-cmw: The Web page of the Controls MiddleWare Project.

[2] N. Trofimow, 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.