MIDDLEWARE IN ACCELERATOR AND TELESCOPE CONTROL SYSTEM

A.Götz, ESRF, 6 rue Jules Horowitz, Grenoble 38043, FRANCE
D.Schmidt, 616E Engineering Tower, University of California, Irvine, CA 92697-2625, USA
M.Clausen, DESY, Notkestrasse 85, Hamburg 22607, GERMANY

Abstract

Middleware is an integral part of all modern control systems. In the past the term middleware was used to refer to the first abstraction layer between the lower-level hardware access layer and the higher level application layer. It was often introduced to hide the network communication protocols. Middleware has developed far beyond this. Modern control systems can now have multiple middleware layers which take care of the network but also increasingly the modelling aspects of the control sytem. New middleware component models provide the technology for modelling the control system from the highest level down to the hardware access level.

This paper will make a review of the current middleware situation in accelerator and telescope control systems. The main component models for CORBA, Java and .NET will be discussed. The paper will cover common network protocols like IIOP, RMI and SOAP and their use in control systems.

INTRODUCTION

Accelerator control systems like most other software systems follow a layered approach. Each layer takes care of a specific aspect of the control system. Telecsopes and accelerators are often physically big machines (anything from 10 m to 1 km to 30 km's). Accelerators and telescopes were therefore amongst the first to exploit the advantages of the network to build distributed control systems. The first distributed control systems used middleware only to hide the details of the network programming. The distribution of processes was however often poorly handled and control systems were tightly coupled. Middleware has not ceased to evolve and the latest generation of middleware allows building of flexible, robust distributed control systems with less effort than in the early days. The main problem today is choosing which middleware to use amongst the plethora of available ones.

After a brief look into what is middleware and where we are coming from this paper will present a review of the current common middleware solutions, examples of their use and how to choose between them.

WHAT IS MIDDLEWARE ?

Middleware was invented to make using general services like the network database, web, naming etc. easier to use. Middleware is the layer between the operating system and the applications. It handles the communication between the distirbuted nodes. As distributed computing have become ubiquitous so has middleware become essential. Problems in the middleware layer can cause a breakdown of the entire control system.

PIONEERING DAYS

Control systems for accelerators and telescopes have been around for quite a while, longer than the network. In the early days control system developers had to invent their own networks, protocols and software. These were adapted to their requirements and were quite impressive considering the fact that often everything had to be developed from scratch. In the 1980's the network came along with some standard libraries for building network connections based on sockets. The socket paradigm has served the controls community for a long time. EPICS, the most widely used control system in the accelerator community still uses sockets to communicate with clients. As the network became ubiquitous and computers have become more powerful the software built around the network i.e. the middleware, has become more sophisticated. Today the middleware layer is being driven by different standards consortiums big companies and the IT business community. There are now multiple standards and languages around and it is up to the controls system builder to make the right choice. This paper makes a review of the main choices available today.

REQUIREMENTS

A solution needs a problem to solve. It is not a good idea when the solution becomes the problem. What are the specific needs of control system concerning middleware ? Middleware needs for control systems can be broken down into general needs and specific needs.

General requirements

General needs are those needs which almost all middleware systems require. These are :

- **distributed** - the network is now a fundamental part of any control system.
- **naming** - a distributed environment needs a way of finding objects
- **database** - configuration information and data need to be stored in an easily accessible manner
- **navigation** - it is useful to be able to browse through a system without any a-priori knowledge
- **security** - users need to be protected from intrusions, viruses and themselves
- **messaging** - the possibility to send data asynchronously to one or many clients.

One middleware service which is widely used in business but which is not required by control systems is **transaction processing**.

Specific requirements

Specific needs are those needs which make control systems different from middleware for enterprises. These needs are :

- **limited resources** - control systems have to run on platforms where there are often limited resources (memory, cpu, badnwidth).
- **hardware access** - the first requirement for all control systems is to access the actuators and sensors.
- **realtime performance** - control systems need to control real hardware within very tight timing constraints.
- **reliability** - control systems run critical mission systems and downtime has to be kept to a minimum.
- **process control** - control systems control feedback loops and other typical process control systems one finds typically in a factory.
- **process attributes** - data coming from actuators and sensors have a fixed set of attributes like warnings, alarms, minimum and maximum values associated with them.

Any middleware chosen for a control system needs to enable the above control system specific needs and not disregard them.

CORBA WORLD

CORBA started off as an object-oriented language independant protocol in the early 1990s. It has since gone through a number of incarnations :

- CORBA 1 (1991-1995) defined an interface definition language (IDL) and a inter-operable TCP/IP based protocol called IIOP.

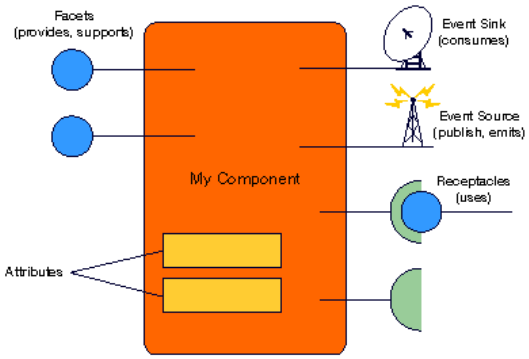

Figure 1: CORBA Component Model diagram

- CORBA 2 (1996-2000) defined a standard for implementing object creation strategies in servers (POA) and added definitions for a large number of services e.g. Naming, Notification, Realtime, etc.
- CORBA 3 (2001-) added the CORBA Component Model (CCM).

The CORBA implementations (called Object Request Brokers, ORBs) handle the marshalling of network calls between clients and servers. The IDL is defined without any reference to CORBA and has been adopted as an ISO standard. IDL can be used to define non-CORBA systems.

The CORBA Realtime extension is very interesting for controls systems because it allows an end-to-end predictability from client to server. This is done by supporting thread and client request priorities at the network (ORB) level. These then map to operating system specific calls locally. Using these primitives priority inversion can be avoided. CORBA Realtime is still relatively new and therefore not widely used in control systems. This should change in the future.

The CORBA Component Model is the most significant addition to the CORBA 3 specification. It is the OMG's response to inter-operability problems between different CORBA systems and the Java world. The CCM is based on version 1.0 of Enterprise Java Beans (EJB) component model. There is an almost one to one correspondance between CCM and EJB 1.0. The CCM proposes the component as a specialisation of the object. CORBA Objects are "dumb" objects in the sense that clients cannot know what an object can do unless it has prior knowledge about the object which can only be communicated by a non-programming interface e.g. by reading a manual. Components are objects which offer pre-defined interfaces and which can receive or send events. Clients can discover what components have to offer via built-in browsing capabilities. Components are managed by containers. The CCM Container model simplifies the way in which components can be created and made available to the external world. Whereas previously the CORBA Portable Object Adapter (POA) offered lots of different possibilities the CCM Con-

tainer model offerstwo main ones. The CCM offers standard solutions for :

- configuring attributes
- persistance of objects and their attributes
- sending and receiving events
- packaging binary components for distribution

The CORBA CCM offers a standard solution for what all control systems builders using CORBA are already doing. Very early on accelerator and telescope control system builders recognised that choosing CORBA is simply not enough for building a control system. It has to supplemented by a component model which implements the features identified in the CCM. The advantage of the CORBA CCM is that it is a standard and there is hope that by adopting the CCM different control systems can share components for controlling hardware or doing high-level tasks. There are not many CCM implementations available today. CORBA is different to most other middleware solutions because it is in many respects *software by a committee*. The Object Management Group (OMG) which manages the CORBA standard produces only a specification but no reference implementation. It is left up to the community to implement the specification. The approach of *software-by-committee* does however lead to the specification being ahead of the implementation. This means there is a time lag before published standards are converted into working implementations and get accepted by the controls community. The recent Corba Component Model is a good example. The standard is available since 2001 but partial implementations are only emerging now. For more information on CORBA and the CCM refer to the bibliography.

CORBA is turning out to be more and more widely used in new telescope and accelerator control systems. The success of CORBA in the controls community is in part due to -

1. CORBA is language independant and object oriented and therefore fits well in with the popular object oriented languages around today e.g. Java, C++
2. CORBA has defined a standard for doing realtime networking,
3. CORBA is vendor independant.

Some good examples of control systems based on CORBA in the accelerator and telescope controls world are :

1. TANGO - new CORBA based control system being built by the ESRF and Soleil (France) to control synchrotrons and experiments
2. ACS - CORBA based control system being built by ESO and Cosylab for the Atacama Telescope control system (Chile)
3. NIF - CORBA based control system for the National Ignition Facility being built by LLNL using Ada 95 and ORBExpress (USA)
4. CMW - CERN middleware project (Switzerland)
5. GTC - new 8m optical telescope control system in Gran Canarias (Spain)
6. ILL - new neutron experiments control system at the ILL (France)

The next logical step in the CORBA controls is to adopt the common CCM model and see if we can share components.

JAVA WORLD

Java is a language designed to be interpreted on any platform independant of the operating system running. It was originally designed in 1991 for handheld devices as part of SUN's vision of the future of computing. It really took off in 1995 when it was married with the internet and integrated into netscape, the main internet browser at that time. Java is very attractive for its platform independance, object orientedness, clean interfaces, internet abilities and wide range of classes. Java has become the main language for programming enterprise logic in the recent years and has replaced COBOL as the enterprise programming language for the future. Java is probably the main internet service programming language too. This has resulted in Java being a if not the cornerstone of business middleware. Accelerator and telescope control systems need to have a Java strategy in order to lever the good features of Java in their favour. Where Java has so far had little inroads is in replacing the operating system at the low-level, traditionally referred to as the frontend in control systems. The reason for this is the lack of Java implementations for frontend platforms which offer access to system features. Isolated examples exist of Java virtual machines (JVM's) being ported to hardware platforms and offering a complete frontend programming environment e.g. Accelerated Technology's Nucleus RTOS, Lejos for LEGO Mindstorms robot. We can expect this trend to increase in the future. We will restrict our discussion to Java as middleware at the higher levels, RMI and Jini.

Java implements solutions for the following middleware tasks :

- naming - Java Naming Directory (JNDI)
- messaging - Java Messaging Service (JMS)
- components - Enterprise Java Beans (EJB)
- database access - Java Database Connectivity (JDBC)
- web pages - Java Server Pages (JSP)

In contrast to CORBA there is always a reference implementation for the Java classes from SUN which is available

free of charge. This makes Java based middleware accessible to everybody and not only the big-iron players.

Why is Java middleware of interest to control system builders ? It solves many of the typical middleware problems faced by application builders, it is multi-platform, many of the Java technologies support heterogeneous environments. For example JNDI supports CORBA naming, Java 2 supports IIOP and CORBA IDL. This makes Java a good integrating middleware. On top of this Java offers a complete graphical library (Swing) for building graphical applications. Java is well-suited as a middleware and language to building the upper levels of control system. The compatibility between the EJB and CORBA CCM component model means both can be deployed together in a transparent manner i.e. EJB components look like CCM components and vice versa. Add to this the fact that Java is often being used by the enterprise resource management systems in many companies one can imagine an integrated solution which spans the control systems and management information systems. However up to now no control systems are using Java as middleware in this way. Most of the use of Java is restricted to graphical users interfaces, the web and as an asynchronous communication mechanism.

Java offers it own remote method invocation (RMI) for distributing objects. RMI offers a naming service, firewall tunneling and a security system. Although RMI is interesting for pure Java based applications it is slower than CORBA and not adapted to non-Java distributed objects. For this reason CORBA between Java and C++ is a better choice for heterogeneous systems. This is typically the situation in control systems where the hardware is controlled via non-Java programs.

Jini is SUN's proposal for controlling hardware with Java. It is intended for managing a group of distributed objects over the network. It provides a naming service, discovery service, and proxy service. Jini works together with JavaSpaces, Jini's persistent store and naming space. Although Jini is aimed at control systems so far very few devices are delivered with Jini support. Consequently Jini is still in its infancy stage in accelerator and telescope control systems yet and as such has not been heavily tested for scalability, robustness, performance yet. Jini

Examples of accelerator and telescopes control systems using Java are :

1. Cosybeans - graphical widgets specifically developed for accelerator control from Cosylab

2. CERN alarm system - a CORBA to JMS bridge

3. TSRF synchrotron beamlines - a Jini-Javaspaces based control system

4. JoiMINT - a Java based monitoring tool which is web enabled

MICROSOFT WORLD

The Microsoft world refers to the range of Microsoft operating systems and products running on them. It is an important world because of its size and financial clout. Initially the Microsoft world was limited to the desktop but recently it has entered the embedded world and the server world. Microsoft based control systems have been rare in the accelerator and telescope control systems. The reasons for this are that the Microsoft systems are not designed to run in controls environments where resources are limited. Microsoft products did not used to integrate the notion of distributed computing very well and there are problem with reliability and recently viruses. Add to this the closed nature of Microsoft platforms and it is not surprising that many control systems restrict their use of Windows to operator consoles. Nonetheless some control systems have been built successfully using only Windows based products. The recent arrival of Java has opened up Microsoft to multi-platform projects. All this does not mean that Microsoft does not have a strong development strategy and it should be ignored. On the contrary the large base of programmer's using Microsoft products creates an important player in the middleware arena.

Microsoft's latest strategy is based on .NET. .NET is as it names indicates based around the Internet. It is a framework for developing applications and distributing them as web services. .NET is completely object oriented. All objects in a .NET application are derived from the class `System.Object`. This permits all objects to offer the same minimum services. Objects are packaged as .dll's or .exe's and shared amongs .NET applications. All .NET objects contain a description of what interfaces they support. This means they do not need to be registered in a repository and can be immediately made available as shared objects without any further action. This is in contrast to the IDL approach of COM and CORBA which needs an extra step. The advantage of .NET is that all objects can be made available as web services. This is a useful service but it is not the main requirement for control systems. Web services are often too slow for doing controls and are therefore mainly used for supervision. .NET is based on SOAP and XML. These are also "slow" protocols. It is therefore unlikely that .NET becomes a major player in control systems unless there is a major change in the way control systems are build and the web becomes pivotal in control systems. It is more likely that .NET gains acceptance for providing a web services as an add on feature for control systems but not as the main framework for doing hardware access and process control.

PROTOCOLS

Protocols are a formalisation of what is sent between two distributed components. Most of the time the protocol is what is sent out on the physical medium (wire) but it can also be used when there is no wire i.e. locally. There is sometimes a confusion between protocols and middleware.

Let it be clearly stated that *a protocol on its own does not represent a complete middleware solution*. A protocol provides the possibility to call a method on a remote object or to pass it data. A protocol usually does not offer a naming service, security, a database interface etc. Here is a list of the common protocols in use today.

RPC

Remote Procedure Calls (RPCs) have been around since the late 1980s. Their principal role was to make a remote call to a different machine look like a local call. They take care of preparing the call parameters to pass them over the network, and returning the data to the caller. They represent an evolution over raw sockets because they take care of the different data representation between machine. Traditionally RPCs are procedure based as opposed to object based. Some well known RPC's are :

- ONC/RPC - the SUN RPC which is part of the NFS file protocol, very widely available and therefore widely used
- DCE/RPC - the RPC from the OSF which was also adopted by Windows for DCOM. Since the demise of OSF the DCE/RPC is not used on Unix platforms anymore and is restricted to DCOM users.

IIOP

The IIOP is the main protocol for CORBA. It is supported in Java 2 too. A low level protocol which is used for marshalling and unmarshalling data on the network. Largely transparent to CORBA users.

RMI

The remote invocation protocol for Java.

SOAP

A protocol based on XML which can be used for doing RPC as well. The RPC usage of SOAP is poorly defined. Usage of XML allows exchanging structured data types as well as simple types it is possible to send SOAP messages via http. SOAP is defined by the W3C consortium and is supported by Microsoft and IBM amongst others. Everything in SOAP is done in ascii and it is therefore a slow protocol mainly being used to exchange information on a peer to peer basis via web servers. SOAP is not really suited for accessing hardware in control systems whereperformance is required.

XML/RPC

XML/RPC is a variation of SOAP. It is XML based and has been developed mainly because SOAP did not implement RPC calls. It has similar features to SOAP but simpler to use.

EPICS/CA

EPICS Channel Access is mentioned here only because it is the main protocol for EPICS control systems which are widely used in the accelerator and telescope community. Channel Access is based on sockets. It runs on multiple platforms, is very efficient, uses a subscribe and publish mechanism to inform multiple clients simultaneously and is implemented in C.

THE FUTURE

The main trend emerging from the middleware technology evolution is that *objects* are being replaced by *components*. Components are specialised objects with a minimum set of interfaces in order to support the specific needs of each system. Most systems including control systems have been doing this implicitly for years already. The standard component models which are appearing are very interesting for control systems builders because they open the opportunity for more software reuse and sharing. In the future it might even be possible to buy software components off the shelf like we buy hardware today.

The next step in middleware is towards adopting common models. The OMG is working on MDA (Model Driven Architecture) to setup standards for models in different domains. These models are defined in standard platform independant languages like UML. Tools are being written to convert these Platform Independant Models (PIM) to source code. This is a very ambitious vision of the future. If it succeeds it would bridge the technology difference between the various middlewares and platforms once and for all.

CONCLUSION

Which middleware to choose then ? We have tried to show in this paper that middleware is evolving fast and there is no clear winner and might never be one. For the moment heterogeneous systems still rule in the control system environment. CORBA and J2EE are the best choice if you want to stay open and multi-platform and multi-language. .NET is the most obvious choice if you want to run only on Windows.

REFERENCES

[1] J.M.Myerson, *"The complete book of middleware"*, Auerbach Publications, 2002.

[2] N.Wang, D.C.Schmidt, C.O'Ryan, *"An Overview of the CORBA Component Model"* in "Component-Based Software Engineering: Putting the Pieces Together", (G.Heineman, B.Councill, eds.) Addison-Wesley, Reading, MA, 2001.