EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
CERN-SL DIVISION

SL-Note-2000-008 (CO)
Also at http://cern.ch/proj-cmw/documents/whitepaper.pdf

The Technology and Techniques for the PS/SL Middleware

The Whitepaper

The Middleware Project Team

V. Baggiolini, F. di Maio, K. Kostro (Project Leader),

A. Risso, M. Vanden Eynden

Introduction

The PS/SL Middleware project was launched in November 1998 to provide a modern middleware communication infrastructure for the CERN accelerator controls. The management plan for this project stipulates that the project will be partitioned in three phases with the first phase (planned to finish end of January 2000) devoted mainly to the study of technology, requirement capture and prototyping.

Following the technology study and the requirements capture, the middleware technology has been selected and the base architecture has been proposed. A detailed evaluation of the available products, design and prototyping is currently being performed. By publishing the whitepaper we want to share our point of view and provide the opportunity to react for those who have not been following the project very closely.

We present the essential objectives and requirements in chapter 2 and discuss the available technology in chapter 3. Then in chapter 4 technological choices, which have been made, are explained. In chapter 5 we propose the raw architecture for the middleware and chapter 6 describes the ongoing and planned activities and the strategy for coming months.

This document is deliberately kept general and it does not reflect all the discussions, which led to the conclusions presented here. To help the reader with the large number of acronyms, a glossary is provided at the end. The style may be slightly varying throughout the paper, which is due to its teamwork character.

Geneva, Switzerland
January, 2000

1. Table of Content

2. Important objectives and requirements.

The User Requirement Document (URD) [1] provides a detailed specification of the middleware requirements. Some of the requirements as well as the objectives of the project have a big impact on the choices and we will discuss them in this chapter. Some of the objectives will be used to justify choices we will make in chapter 4.

2.1. Object-Oriented technology

There are several aspects of using Object-Oriented technology. When the Middleware project was launched, inter-object communication was put forward as one of the objectives. Even if there is little OO software in the accelerator controls at the moment, this is bound to change in future, following the current industry trend. The URD requires the Middleware to support invocation of methods on remote objects (OO-01 to OO-03), which implies use of Object Request Broker (ORB) technology.

Another reason for promoting object orientation is the proliferation of Java. Java is the language which experienced the biggest growth in the last couple of years, and software industry reports that the productivity is increased when Java is used - especially in the debugging and maintenance phase. The dominant environments for software development today, especially for GUI, are Microsoft Windows (Visual C++, Visual Basic, etc.) and Java.

2.2. Device/Property Model.

The Java API project, which is another project between PS and SL control groups to modernise the accelerator controls [2] has defined a simple model to access accelerator equipment - the **Device/Property Model**. The objectives of the Middleware project stipulate the support for this model as mandatory. Many of the requirements in the URD [1] refer directly to the capabilities of the Java API and the Device/Property Model. In the following we give a short overview of the model.

2.2.1. Model overview.

The Java API depends on a device-oriented view of the control system. In this view the system consists of named **devices.** A device may represent a physical device in the control system (such as a magnet or a beam position monitor), but it may also represent an abstraction of control entity (e.g. a ring with associated tune and orbit measurements). Conceptually, devices are composed of **device properties**, which constitute the state of the device. By **getting** the value of a device property the device state can be read. Accordingly, a device can be controlled by **setting** one of its properties with the required value. For instance, a magnet may have a “current” property. The getting or setting of this property respectively delivers the actual value of the magnet current or brings the current to the required level. Although devised independently, this model is very similar to the Java beans model with its get/set attribute assessors. Properties which support get/set operation also support **subscribe/unsubscribe** operation.

Devices are organised into **device classes** that describe the device interface i.e. the available properties and their type. Devices of the same class have the same set of properties. Device classes constitute the meta-description of devices.

Properties may have **characteristics** such as units, resolution, etc. A property’s characteristic is either a single value, common to all members of the same device class, or a reference to another property, which holds the value. Among others, characteristics can be used to describe the purpose of a property or to make relations between properties.

2.3. Subscription facility.

This facility is one of the key goals of the project. The requirements DSS-001 to DSS-016 [1] all describe the subscription capabilities. Essentially there is need for two types of publishing:
1. When the value of the published value change, and
2. When an accelerator timing event occurs.

It is important that the subscription system is reliable i.e. with the subscription-on-change the client process is notified when the distributor has failed. Another requirement was to be able to group subscriptions and be able to subscribe to a group of devices or to a "subscription subject", notably to facilitate development of alarm applications.

2.4. Interoperability with industrial systems

When the Middleware project was launched, one of the objectives was the capability of data exchange and control interface with LHC subsystems, ST services and LHC experiments. Meanwhile this has been reduced[1] to interoperability with the industrial SCADA systems, which offer an OPC interface (requirement IOP-001 [1]). (See also chapter 3 for explanation on OPC).

2.5. Timing and Cycle Support.

As timing is an essential part of the PS and SPS controls it has to be well supported by the Middleware. It should be possible to interact with devices in a timing-synchronised way i.e. perform a setting operation at requested timing event or subscribe to a property on-change but only at a requested event time (SIOS-004, AIOS-006 [1]). The same considerations apply to the machine cycle. The recent developments in the SPS2001 project indicate that timing requirements have to be seen from the cycle point of view and that "raw" timing events should be hidden from the users.

It is also required that the received data can be stamped with the machine cycle identifier and with the acquisition time (DSS-003, SDT-002 to SDT-004) [1]).

2.6. Reservation and Security.

In the accelerator environment non-authorised operation of some devices can result in damage to equipment. This is why a security control mechanism is necessary (ACS-001, ACS-002 [1]).

It is frequent that programs have to perform a sequence of operations on devices and that during this sequence access to the device from other programs should be blocked. This means that a reservation system is necessary (RS-001 to RS-004 [1]). To avoid "forgotten" reservations, which would block devices a guard detecting such situations is necessary.

2.7. Administration and surveillance.

The Middleware should allow administrators to observe the load, detect timeouts and bottlenecks and repair broken parts easily (ADM-001 to ADM-008). It should also be possible to verify functionality and make trials or performance tests without resorting to writing programs, for instance with a scripting language.

[1] In this fast-changing world new protocols based on XML, which appeared recently, seem to be well adapted for data exchange between LHC subsystems (see chapter 3).

3. Overview of technology.

The functional requirements for the control system are not changing very fast. But the way of making software systems, familiarity with the look and feel of interfaces and "current software practices" are changing rapidly. If the Middleware does not use the technology of this decade, it will run the risk of becoming obsolete in the LHC era. In the following we describe some of the technology, the "enabling technology", which represents the current trends and which will make easier putting the new Middleware in place.

3.1. Definition of Middleware

The term "Middleware" is used in a very broad sense. The often-evoked comparison with electronic systems defines it as a "Software Bus for distributed applications". This is still rather vague but what is usually meant by "middleware" are software systems which:

- Go beyond the client-server model
- Provide an additional layer which allows to interact with server (or service) abstraction such as:
  - Distributed objects (Object request brokers).
  - Message queues (Message Oriented Middleware).
  - Unified database access (Most commercial middleware have been made to address this).
- Provide added value such as location service, reliability, authentication, transaction semantics etc.

3.2. Object Request Brokers and Message Oriented Middleware

In inter-process communication the trend is to move from client/server systems, which are based on sockets and RPC (Remote Procedure Call) technology to ORB (Object Request Broker) technology. But at the same time Message Oriented Middleware products conserved their importance and have been adapted to Object-Oriented technology. In the following sections we briefly describe both. A more exhaustive description can be found in the presentations from the Workshop on Middleware Technology [3].

3.2.1. Object Request Brokers (ORB)

ORB systems offer facilities for remote object location and invocation, which match well facilities of Java and C++. CORBA [4] is the technology usually used to integrate existing (so-called legacy) software with new Web-based applications and in multiplatform, multilanguage applications in general.

The two main software development environments today: MS Windows and Java also promote their own equivalents of CORBA: DCOM and RMI. The disadvantage of DCOM is that it is not really open to non-Microsoft platforms. DCOM evolved from COM and was originally not designed for a distributed environment. For instance naming services are virtually non-existing.

Compared to CORBA RMI offers a better integration with Java - there is no need for a separate interface description language and the full local functionality including sending objects (by value) is available. On the other hand RMI can only be used between Java programs but CORBA bindings and implementations are available for virtually any existing language and computing platform. On the negative side CORBA has been created by standard bodies and is full of compromises. The process of introduction of new facilities and services, although improved, is still very slow and it takes years.

There is some recent integration between CORBA and RMI: RMI can use the CORBA IIOP protocol and there is a mapping between the Java and the CORBA IDL, which make use of IDL for Java applications optional.

3.2.2. Message Oriented Middlewares (MOM)

The Message Oriented Middleware family is characterised by products that facilitate asynchronous, point to multi-point non-blocking communications. These systems are based on the publish-subscribe paradigm allowing “publisher” processes to push data into the system and “subscriber” processes to consume data. The complete de-coupling between the publishers and the subscribers (as opposed to peer-to-peer communication) is a major characteristic of MOMs.

In most products, the data transferred through the system can be organised into hierarchical “subjects” to which publishers send predefined messages. Facilities like flow control, load balancing, fail-over and diagnostic tools make the MOM products good candidates for building systems where information losses are not acceptable.

Some products are capable of using IP (Internet Protocol) multicast in addition to unicast to disseminate information to the subscribers. This facility makes the development of highly scalable applications possible.

Today, the emergence of the Java Message Service (JMS) developed by SUN improves programmer productivity by defining a common set of messaging concepts and programming strategies that will be supported by all JMS technology-compliant messaging systems. JMS seems to take momentum and is now progressively adopted as a standard API by leading enterprise-messaging companies.

OMG group has also defined a CORBA messaging standard - the CORBA Notification Service. It is similar in scope to JMS and backward compatible with the older Event Channel Service. The final specification has just been released but some CORBA suppliers anticipated the final specification and are already offering implementations of the Notification Service.

3.3. Software development environments

There are currently three “mainstream” development environments:

1. **Java environment** with a large number of toolkits available, including GUI beans. Java runs on any platform for which a Java Virtual Machine (JVM) exists. In practice the best JVM's are available on Windows, Solaris and Linux. The advantage of this environment is that it reportedly increases programmer productivity and that Java is an easy-to-learn and relatively fool-proof language. Java is still relatively slow in execution compared to C and C++ and Microsoft does not fully embrace it.

2. **Microsoft Visual C++ and Visual Basic** are “industry standard” in commercial software. The advantage is a good integration in the leading commercial platform environment. The development in this environment is complex when it comes down to using C++ and Microsoft Foundation Classes. We have little experience with it and it does not seem to be in line with the CERN investment in programming.

3. **C (and C++) with Motif** - this is what we are using currently. This working method is not developing anymore and often considered obsolete. In low-level software such as drivers use of C for programming is still the standard approach and on Front-End systems, especially for Real-Time all alternative methods appear exotic.

3.4. Application servers and three-tier systems

The trend to three-tier systems is mainly driven by technological changes and necessity to integrate software systems, which were not originally designed to work together. For instance the advent of Web browsers created a new class of clients which have to be integrated in the enterprise computing. Merging of companies requires subsystems such as accounting to be merged together without rewriting the whole software. This is typically done by separating the system in three layers called "tiers":

1. The tier 1 (client tier) is typically a GUI, which is responsible for the visual aspects of the application - displaying the client objects and applying the client view of the data. The client interacts with the tier 2 (middle tier) to perform any action or display data. An example of this tier would be a graphical application which displays beam position and allows tuning of the beam in a graphical way.

2. The tier 2 (middle tier) is a server, which implements the “business logic” of the application. For instance the “beam server” could implement an abstract model of a beam and include algorithms to acquire beam properties or tune the beam. The middle tier server objects can interact with many clients (e.g. to display the beam position) but they rely on tier 3 to get data from equipment or from the databases. The server objects provide a unified view of the different data sources. The middle tier software concentrates on implementing domain-specific logic and therefore its lifetime is not limited by a specific access technique or a database being employed.

3. The tier 3 (resource tier) is typically a “legacy application” - a piece of working software which can be used to extract data and perform actions but which implements it in a specific way. For instance an OPC server and a SL-Equip server can both provide access to equipment but they have little in common. It would be difficult to replace one by the other for specific hardware equipment. But if the access is done via a generic equipment object (middle tier), it can be made transparent to the clients in tier 1.

3.5. Server and component frameworks

In the previous chapter we briefly presented the typical architecture of modern distributed object-oriented software systems. To write such software system is still not an easy task. For instance some kind of authentication has typically to be performed between the client tier and the middle tier. Or access to equipment might have to be done in parallel using threads. The idea is to make these tasks easier for the programmer by providing a component framework. Within this framework programmer can place his objects in component containers, which will provide services such as persistency, security etc. to these objects. When the framework is a well-defined standard, components from different sources can be integrated into a single server.

There are currently three concurrent technologies (exactly as for ORB): Enterprise Java Beans (EJB), CORBA components from OMG and Microsoft Transaction Server (MTS) from Microsoft. There is some interoperability between CORBA and EJB components as they can be clients of each other and an EJB component can be deployed into a CORBA container (and vice-versa with some restrictions). Component frameworks for all three competitors are "under construction" and not fully defined and implemented yet.

3.6. OPC

OPC (OLE for Process Control) [5] is an industry standard for communication between SCADA systems and field devices. The rationale for this standard was mainly reduction of the number of drivers necessary to operate electronic devices and fieldbuses. When an OPC server is available for a device, this device can be easily used with any SCADA system, which employ a single protocol to access a number of devices from many manufactures. In OPC the server appears as a list of named items of which the values can be read or set. There are interfaces to find servers and discover their namespace. Clients can organise individually their access to server items by grouping them (OPC group). There are several possibilities to read item values: by a synchronous get, by an asynchronous get or by subscription. When subscribing to an item it can be specified that the update should be delivered only if the value changed by a percentage (deadband). In general OPC defines the access protocol but not the rules for implementing the servers. Although OPC can potentially be used as a middleware in practice the underlying DCOM infrastructure is difficult to use and fragile in case of failures[2].

3.7. XML

XML stands for eXtended Markup Language and is expected to relay HTML as the markup language for World Wide Web. XML is also well adapted for description of data structures and has been embraced by database makers. The importance of XML has been widely recognised and both CORBA and Java are addressing XML integration. OMG has issued an RFP (Request for Proposals) on mapping between IDL and XML. Sun has proposed a way for mapping XML documents to Java objects and vice-versa. And XML can even be used to describe protocols. In a recent development Microsoft has submitted an Internet standard for remote method execution which is based on XML and HTTP. It is called SOAP (Simple Object Access Protocol) [6] and Microsoft wants to use it as a base for component communication in the future generation of COM+. XML implies high data overhead (everything is sent as text), so that such protocol would be inefficient for closely coupled systems but it seems to be well adapted for data exchange between independent systems such as LHC subsystems. This development is new and will be followed by the Middleware project.

[2] This has been recently admitted by OPC, which is embracing the new Microsoft strategy of using XML for remote communication.

4. Choices

This section describes the preliminary choices of middleware families and technology. From the wealth of families and products, we intend to retain some and discard others. This process shall eventually lead to the identification of a small number of products that we shall install in our test beds and use for the implementation of prototypes and of the first version of the Middleware.

Making choices is an arduous task, and it is difficult to satisfy all users. In the following paragraphs, we will present our decisions and illustrate the reasons that have lead to them.

4.1. Summary of relevant requirements

The following requirements are relevant for our choices.

- Object-orientation. We plan to develop object-oriented applications; therefore we should choose a Middleware that integrates well with this technology.
- Standardisation. We will prefer middleware products with a standardised API, to avoid vendor lock-in. By using standard APIs programmers can substitute the product of one vendor with the product of another vendor fairly easily. This is not possible when using products with proprietary APIs.
- Multi-platform. We have to support a wide range of operating systems, ranging from LynxOS over HP-UX and LINUX through to Windows NT.
- Commercial products. Where possible, we shall integrate industry-strength middleware products, rather than develop software in-house. Commercial software products are generally of higher quality and have more functionality than in-house developments, and they can be backed up with maintenance contracts.

4.2. Choices of Middleware Families

Our choices can be summarised as follows. For request/reply communication (as used in RPC-style set/get interactions), we will retain CORBA as the main solution, because it is an open standard and well adapted to heterogeneous environments. RMI remains a complementary solution applicable for communication between Java applications. Therefore, all developments undertaken with CORBA shall be kept compatible with a possible migration to RMI. DCOM, the third major competitor, will be discarded from the list of products we intend to pursue, because it is a proprietary solution.

For Publish/Subscribe communication, we are currently evaluating the use of Message-oriented Middleware (MOM), in particular products that are compliant with the Java Message Service (JMS) standard.

As for OPC, we will provide strategies and solutions for integrating OPC servers (such as industrial hardware and SCADA systems) into the control system middleware. In other words, the middleware shall act as an OPC client.

In the following, each choice will be explained in more detail.

4.2.1. CORBA

We will select CORBA as main request/reply communication family for the following reasons:

- It is the best technology to integrate diverse platforms and languages, including C, C++ and Java.
- CORBA supports communication between object-oriented applications.
- It is an industrial standard, endorsed by a consortium of over 800 companies, that supply a whole range of products from which to choose.
- Products are available on virtually any platform, including LynxOS.

4.2.2. RMI

We retain RMI for the following reasons:

- RMI tends to be easier to use than CORBA.
- RMI is becoming more and more integrated with CORBA; a first sign of this is the RMI-over-IIOP specification that allows programmers to use the user-friendly RMI API to communicate via CORBA.
- RMI supports the full OO paradigm, including the transfer of entire objects (as opposed to CORBA, that can only pass data structures).
- RMI is good especially for communication between Java programs, but it also provides means for server-side integration of C/C++ programs. On the other hand, integrating C-Client with RMI is possible but quite bulky.
- It is a 100% Java solution; as such it can be used on any platform with a Java Virtual Machine. In future (mid-2000), this will include LynxOS.

4.2.3. DCOM

For the following reasons, we decided not to use DCOM as the main middleware technology:

- DCOM is a proprietary solution owned by Microsoft. It is targeted at communications within Windows-only environments. Microsoft does not provide solutions for other platforms.
- Microsoft has published the DCOM standards on a good-will basis, without any commitment to guarantee this openness for future versions. It is therefore not sure that future version of the specifications will still be available. [3]
- Although DCOM has been ported to other platforms by third party companies, these products are reportedly less reliable than the Microsoft version, and they tend to become available later.
- DCOM is not supported on LynxOS and will most certainly never be.

Note however, that we will have to use DCOM peripherally to integrate OPC (cf. below).

4.2.4. Message-oriented Middleware

We intend to explore Message-oriented Middleware products because:

- They have a number of capabilities that are highly useful for implementing the publish/subscribe paradigm, which are not present in any of the three ORB products presented above. We will therefore try to implement pub/sub functionality by using a commercial MOM (in parallel with CORBA).

[3] Actually Microsoft plans to change DCOM foundations as indicated in the previous chapter

- MOM products seem very suitable for the software diffusion of timing, as support for the Alarm system and for the transport of the information currently transmitted via Teletext.

According to the strategy of adopting standards where possible, we intend to closely examine products that implement the Java Message Service (JMS) interface[4]. The solution with CORBA Notification Service will be kept as fallback solution.

4.2.5. OPC

We consider OPC as a means of integrating industrial systems into our control system. The reasons are that OPC is the de-facto standard to connect to industrial systems, such as PLCs and SCADA systems.
For the following reasons, OPC cannot be used as a general-purpose Middleware for the whole control system:

- OPC is based on DCOM and strongly dependent on the Windows platform.
- OPC is not object-oriented, but a low-level, tag-oriented protocol. As such, it is suitable for hardware access, but not for communication between parts of an object-oriented control application.

The middleware will implement the OPC data access standard as a client to industrial OPC servers (e.g. PLC and SCADA). They shall be made accessible to a Unix-based control system through a bridge.

5. System Architecture and capabilities

As it has been seen from the previous chapters, even if some of the decisions are not clear presently (such as deployment of an adequate MOM system), the basic architecture has emerged as presented in Figure 1. This architecture represents both the current work and the vision of the future developments. We have used colours to distinguish the origin of different building blocks. If you have a paper copy you will probably miss the colours. In this case you can take a look at http://proj-cmw.web.cern.ch/proj-cmw/drawings/architecture.pdf. In the following explanation we will use ***bold italic*** when referring to boxes of this drawing.

5.1. Which parts will be provided by the Middleware project

The user-written parts are depicted in blue. In the simplest case they are user interface programs which use the generic ***Device/Property Java API*** interface and access existing device servers (e.g. ***PS Equip. Modules***) via the Middleware infrastructure. Later other device servers might be developed (***New Device Servers***), and plugged into the Middleware device server framework. The device interfaces are described by ***Contracts***, which are defined by users. (SPS2001 defined some generic contracts). Finally we believe that eventually most of the stable software developments should occur in the middle tier (***Controls Business Logic***) which will implement physics- and operation-related contracts, independent of the resource tier servers.
The two ***Domain Specific*** boxes at the right hand express the possibility of developments, which go beyond the Device/Property model, using frameworks such as EJB.

The Brick-coloured blocks are industrial or other existing parts on which the Middleware will be built. The solid ones represent commercial software while the shaded blocks are in use in the

[4] The JMS interface is part of the Enterprise Java Beans platform, which might be used for the development of the middle tier

current control system. The long bar marked "***CORBA, RMI, JMS***" represents the ORB and MOM infrastructure on which the Middleware will be based. The recently defined CDEV Java API [2] (***Java*****)** has been completed but it will have to be integrated with the Middleware protocols.

The green and yellow parts are the ones, which have to be provided by the Middleware project. The parts, which are in green, are the ones, which are currently "in works" while the yellow parts will be tackled in the later stage of the project[5].

At the bottom we have the ***Device Server Framework*** for plugging-in device servers. Initially this framework will be used by the gateways to the existing device servers: ***PS Gateway, OPC Gateway and SL Gateway***. All this constitutes the ***Simple Device/Property Server*** with the common software being usable for other servers as well.

Actually the ***Device/Property Server*** and the ***Middleware Interfaces and Protocols*** can be seen as a piece of distributed software with the glue and skeleton being provided by the CORBA/Java brick between them.

The ***Advanced Device/Property Server*** in the middle tier is similar to the one before with the exception of extra capabilities which will be offered (see also the following sections).

Finally, the ***C/C++*** API (as well as scripting API) can be offered and will follow the same style as the Java one.

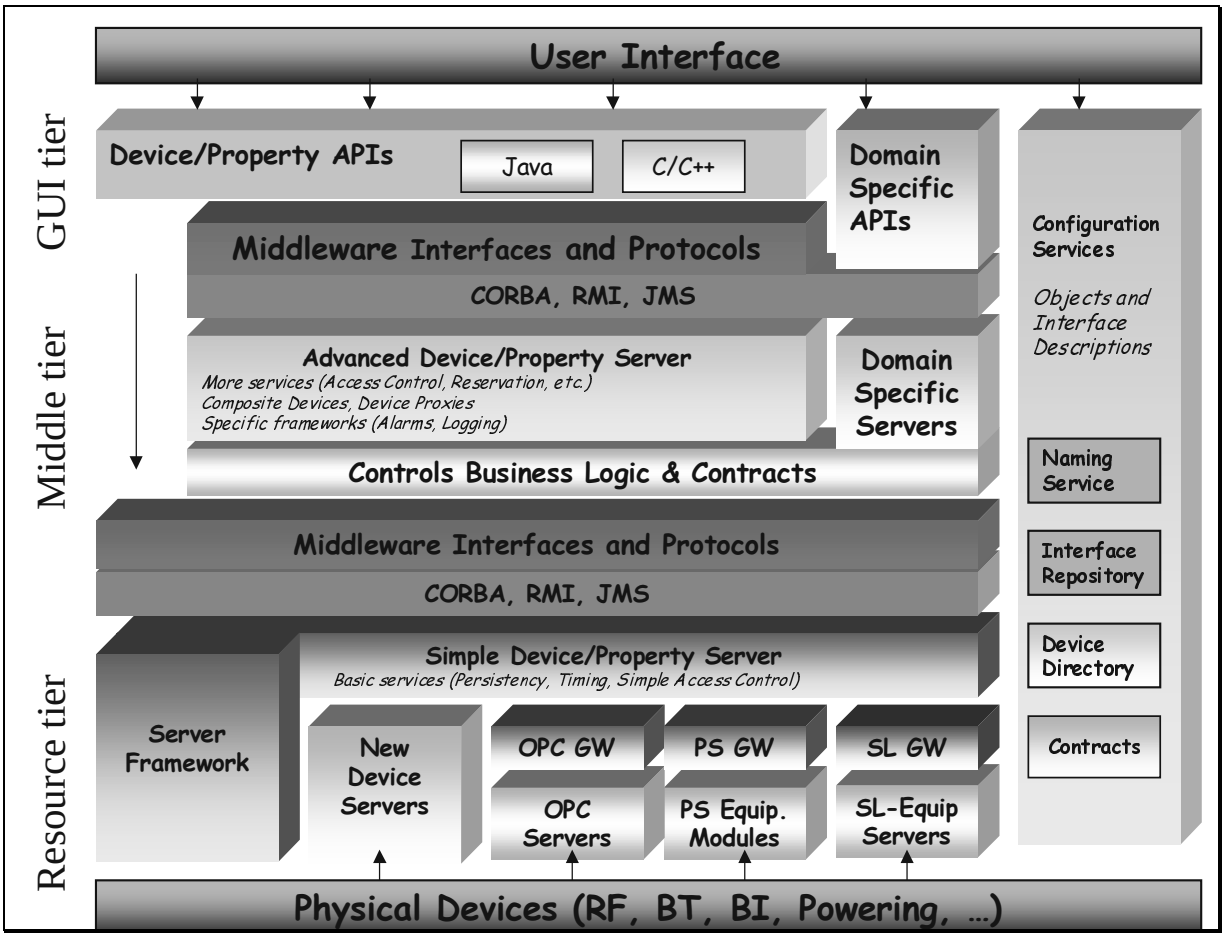

Fig. 1 Middleware Architecture and Components (should be viewed in colours!)

[5]Disclaimer: This does not reflect any priorities within the project.

5.2. Three tier architecture

The anticipated architecture is three-tier and this for various reasons: First of all this allows de-coupling of the generic device server layer (resource tier) from the part which implements device specific behaviour (middle tier), and the GUI. Secondly we anticipate that the services which will be available on workstations and PCs will be more complete as compared with what is available on the Front-End level so that the three tiers will often coincide with the different deployment platforms for the Middleware. And finally the past experience has shown that programming in the Front-End computers can be tedious, as the programming environments are always behind those available on PCs and in workstations.

The GUI tier

The GUI tier is the one, in which the user interfaces are implemented. There we will usually access middle tier objects which implement a high level object-oriented abstraction of control objects and composite devices. Some applications might want to access the resource layer directly (generic equipment control, prototypes, maintenance interface), which is expressed by the arrow going directly to the resource tier.

The GUI tier can use objects of the middle tier via the device/property API. One such API which exists today is the CDEV Java API [2]. This API can be improved in two ways: firstly by defining a set of standard properties ("Contracts") and behaviour (e.g. State Management, Cycle Management or Measurement) and secondly by providing an interface in which the get/set methods can be type-checked at compile time (so-called wide or beans interface).

The middle tier

The middle tier is the "business logic" tier. This tier will typically implement the operation view or the physics view of the equipment. It is the middle tier, which must assure that operations on equipment are meaningful and performed in the correct order. At this level devices will often be grouped to provide a higher abstraction layer (e.g. transfer line, PS working set), than a simple device instance. The separation of the business logic from the user interface part requires a more careful design but will be profitable in terms of reusability, reliability and performance.

The construction of the middle tier servers will be facilitated by server and component frameworks such as EJB, which will provide object persistency, security etc.

The business logic tier can use the objects of the resource tier via the same API as described before.

The resource tier

The main responsibility of this tier is to provide access to equipment. As at the top level we view equipment as devices with properties which can be set or read. The properties are accessible via Device Servers which themselves implement the access to physical devices.

There are two varieties of servers existing actually: SL-Equip and PS Equipment modules. In addition it has been decided that OPC will be used as the interface to industrial systems.

Other equipment-specific servers might be developed in the future, which will make direct use of the equipment server framework.

5.3. Different capabilities on different platforms

Our goal is to provide the same architecture on all the platforms to be supported. But development capabilities and availability of software libraries, tools etc. on the Front-End platforms will always be limited. For instance persistency or security will be difficult to implement, at least in the present stage. Java, which is the best OO development language for the time being is not available on all platforms. On the other hand CORBA is available everywhere and we can be certain that we can make simple variants of publish/subscribe and synchronous I/O available on all platforms via CORBA.

For this reason we anticipate the availability (and implementation) of middleware capabilities in two stages:

1. Basic capabilities, which include subscription and synchronous I/O, will be made available on all platforms. They will also be implemented in the first iteration to provide the basic functionality as soon as possible.

2. Services which are specific to controls such as timing, alarms and logging but also generic services such as persistency and security would be implemented in the second iteration and, as the first priority, on platforms which would host the middle tier servers (workstations and PC's).

To implement the more advanced services we can use Java as development language since it is available on the middle tier platforms. Some of the services such as security and persistence are available in industrial component frameworks such as EJB (Enterprise Java Beans). Other services can be implemented using this framework as support. For instance composite devices and device proxies could be supported in the middle tier.

6. Current and future developments

So far we have concentrated mainly on requirements and technology. But there were also very concrete investigations and "Use Cases" such as the feasibility of CORBA deployment on LynxOS, CORBA to OPC gateway and analysis of the publish/subscribe mechanism. There are other areas, which have not been touched yet, which we have to investigate and develop. In this chapter we will shortly describe the developments on which we will focus in the coming months and the strategy for their conclusion.

6.1. Publish/Subscribe analysis

The publish/subscribe facility has been identified as the key Use Case for the Middleware. Subsequently we described the Use Cases for the different server environments (PS, OPC, SPS2001) and started the analysis process. Recently we have made the strategic decision to use CORBA for the low-level device server integration and use a JMS-based MOM for update filtering and distribution to subscribers. We defined the interface between the two and will continue the analysis and design process. Two prototype server implementations are envisaged: one for the PS Equipment Modules on LynxOS and one for the OPC gateway on Windows NT.

Use of CORBA technology for synchronous operations and for subscription is currently being validated in the recent implementation of the CDEV Java API. The feasibility of using Java JMS and MOM technology is currently being investigated.

6.2. Using of CORBA Technology

We have already conducted a CORBA technology investigation aimed at finding products, which are available for each of the platforms we have to support, and at analysing the facilities provided by each product. As a result of this investigation, it was evident that we can freely choose the best suitable CORBA product for all supported platforms, except for LynxOS where only a few products are available (see the next chapter).

We are also gaining experience with CORBA by using it as the data transport technology for the present PS Java CDEV API implementation. This implementation will be deployed in the operational environment in the PS accelerator in spring 2000 and will give us valuable practical experience with such an approach. A prototype of subscription to OPC servers using a CORBA callback mechanism has also been implemented.

Two of the currently used CORBA products are in public domain. Once the concept has been proven we will investigate whether using a product from one of the market leaders is preferable. Commercial software has in general higher quality standards and more complete functionality and maintenance contracts help to keep it up-to-date.

6.2.1. CORBA on LynxOS

Running CORBA on LynxOS platforms is one of the major risk areas for the middleware project. The constraints on these computers are well known and the most relevant are documented in the user requirement document [1].

As part of the CORBA investigation a public domain CORBA implementation was successfully ported to LynxOS 2.5.1. Subsequently interoperability and performance tests were made with clients in Java and C++ on AIX workstation and servers in C++ on LynxOS 2.5.1. The results were positive and consistent with our requirements.

In spite of the good results we would like to avoid long-term in-house maintenance of a public-domain CORBA product. A company has recently made available a CORBA implementation, which is targeting embedded environments and running on LynxOS 3.X. This product is being evaluated as a possible alternative to the public-domain one.

6.3. Evaluation of MOM for publish/subscribe

We decided to use a MOM product for publish/subscribe. By using the standard JMS interface we expect to simplify the development and obtain filtering capabilities required by the Middleware. Using of JMS shall give us some choice of available products.

We do not have any previous experience with JMS and JMS-compatible MOM and we do not know if the performance will be adequate. As the first step a feasibility study is being performed. The SPS2001 project has defined a test case to investigate both the applicability of subscription to its concepts and the feasibility of using a MOM as the base for the Middleware subscription.

6.4. Investigation of EJB

With Enterprise Java Beans Sun has defined a standard component-based model for delivering services. The EJB framework provides services for persistency, security, transactions etc. and offers facilities for load balancing. EJB-based environments include tools for server deployment and management. There is a large offering of EJB-compliant products.

EJB seems to be a perfect framework for the middle-tier servers. To gain experience with EJB we should investigate some products and devise simple Use Cases for the Middleware.

6.5. Configuration Services

Two implementations of naming services are currently in use in the accelerator control environment: the SL-Equip name server and the PS directory service. The CDEV Java API provides an interface to the latter.

At the same time CORBA has a standard for naming and repository services and Java defines its own API with the Java Naming and Directory Interface (JNDI). It has to be investigated how to combine the standard services and the equipment directory service and adequate products have to be found.

7. Glossary

API. Application Programming Interface. The procedure or method calls which constitute the interface available for user programs.

COM. Component Object Model. Microsoft's framework for developing and supporting objects that can be accessed by any compliant application.

CORBA. Common Object Request Broker Architecture. A standard architecture, defined by the OMG, for communication between distributed objects.

DCOM. Distributed Component Object Model. Extensions to COM which allow remote access to COM components.

EJB. Enterprise Java Beans. A component specification developed by Sun for server-side components written in Java.

GUI. Graphical User Interface. The application, or part of it which is responsible for the interaction with the user.

IDL. Interface Description Language. The language in which object interfaces are described in CORBA.

JMS. Java Message Service. The API, specified by Sun to access MOM systems from Java. JMS is part of the EJB specification.

MOM. Message Oriented Middleware. A general term for middlewares, which are based on messages rather than on remote object invocation.

Object. In object-oriented programming, a single software entity that consists of both data and procedures to manipulate that data.

OLE. Object Linking and Embedding. A set of COM-based technologies now normally referred to as ActiveX.

OMG. Object Management Group. A consortium that aims to define a standard framework for distributed, Object-oriented programming. The OMG is responsible for the CORBA specification.

OPC. OLE for Process Control. An industry standard created with the collaboration of a number a leading world-wide automation hardware suppliers. OPC defines a standard common interface for communicating with diverse process-control devices.

ORB. Object Request Broker. An ORB is a middleware component which acts as an intermediary between a client and a distributed object. The ORB is responsible for delivering messages between the client and object across a network.

PLC. Programmable Logic Controller. PLCs are diskless compact computers including all the necessary hardware interfaces for the process level. They are generally used for automatic control application (closed loop control, etc.) either standalone or networked through a fieldbus such as Worldfip, PROFIBUS or most recently on Ethernet connections.

RMI. Remote Method Invocation. The ORB specified by Sun for Java. It allows method invocation on objects. Interfaces are specified in Java and not in a separate IDL.

RPC. Remote Procedure Call. The system which allows to call a procedure in a remote (server) program and receive the results of that call.

SCADA. Supervisory Control and Data Acquisition. A control system framework to supervise and control industrial processes. Normally build around a real-time database and includes facilities for GUI, trending, alarms and connecting hardware.

XML. eXtended Markup Language. The universal format for structured documents and data on the Web defined by WWW consortium. It is similar to HTML but much more flexible and powerful.

8. References

[1] CERN PS/SL Middleware Project, User Requirements Document, CERN Note SL/99-16(CO), 16 Aug. 1999, also at http://web.cern.ch/proj-cmw/documents/URD1.3.pdf

[2] http://proj-pssl.web.cern.ch/proj-pssl/projects/javapi/javapi.html

[3] http://web.cern.ch/proj-cmw/workshop/workshop.html March 26 1999.

[4] http://www.omg.org

[5] http://www.opcfoundation.org

[6] http://www.develop.com/SOAP