Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

PCT: Component-based Process Control Testbed

Ricardo Sanz, Rafael Chinchilla, Manuel Rodriguez, David Perez and Carlos Martinez*

Abstract— **In the last years industrial control systems have increased their complexity, using a great variety of hardware and software elements of all kinds. In larger systems a big number of different protocols, communication networks and architectures can be found. System components are not centralized but distributed over the plant and they communicate with each other through several channels. Industrial control systems are usually organized in layers, and information used within/among them needs to be integrated. The general trend is to develop heterogeneous and distributed control systems, where system integration is a hard issue. CORBA is an open standard that provides a flexible middleware capable of integrating complex applications in heterogeneous environments, and could be a good choice when developing a control system. Nevertheless, CORBA has some limitations regarding to hard real-time applications, so it is only suitable for developing soft real-time systems. This paper describes the Process Control Testbed (PCT) of the IST Hard Real-time CORBA project (HRTC), which has been used to analyze and evaluate CORBA advantages and disadvantages in distributed real-time control systems development. The PCT system has been designed and implemented using CORBA and RTCORBA. A new version is being implemented using CCM Technology in the IST COMPARE project.**

I. INTRODUCTION

A typical control system in a modern plant is composed by a heterogeneous collection of hardware and software entities scattered over a set of heterogeneous platforms (operator stations, remote units, process computers, programmable controllers, intelligent devices) and communication systems (analog cabling, serial lines, fieldbuses, LANs or even satellite communications). This HW/SW heterogeneity is a source of extreme complexity in the control system regarded as a whole. Apart from the platforms that provide support to the different control system components, the technologies used in control system implementation are quite heterogeneous and provide functionalities that go well beyond the classical sensing/calculating/acting triad. Examples of this heterogeneity is the use of software systems for controller auto-tuning, advanced monitoring, filtering and estimation, adaptation and learning, plant-wide optimization, or real-time, in-the-loop simulation. Interception software systems are playing a wide variety of roles in complex controllers acting as interfaces between preexistent systems (plants, controllers and humans). Classical hierarchical layering overcomes some of the difficulties of complex systems construction. While hierarchies encapsulate low level behavior, simplifying the deployment of higher level controllers, they do not necessarily solve the problem of the conceptual integrity of the system. Layers can be difficult to match if they lack a common view of structure and responsibility distribution. Conceptual integrity is seen as the core factor affecting systems constructability, and can be achieved by using Distributed Object Computing solutions. This paper describes past and ongoing research work on the Process Control Testbed in the IST HRTC project.

*Ricardo Sanz is with the ETSII, Universidad Politecnica de Madrid, Madrid, Spain Ricardo.Sanz@etsii.upm.es

Rafael Chinchilla is with the ETSII, Universidad Politecnica de Madrid, Madrid, Spain rchinchilla@etsii.upm.es

Manuel Rodriguez is with the ETSII, Universidad Politecnica de Madrid, Madrid, Spain mrod@diquima.upm.es

David Perez is with the ETSII, Universidad Politecnica de Madrid, Madrid, Spain dpgonzalez@alum.etsii.upm.es

Carlos Martinez is with the ETSII, Universidad Politecnica de Madrid, Madrid, Spain carlosm.upm@gmail.com

II. DISTRIBUTED CORBA-BASED CONTROL SYSTEMS

An approach widely used in distributed control systems development is Distributed Object Computing (DOC) (see [8]). DOC is a software model based in the use of services provided by objects that are possibly running in different hosts. DOC is a ”natural” way of modeling distributed systems because it hides implementation details (OS protocols, languages) behind ”interfaces”. Encapsulation, abstraction and inheritance are valid and very useful concepts to model distributed control systems. There are many benefits of using DOC for control systems engineering: object collaboration through connectivity and interworking, performance through parallel processing, reliability and availability through replication, scalability and portability through modularity, extensibility through dynamic configuration and reconfiguration, cost effectiveness through resource sharing and open systems, maintainability through hot swapping, and design flexibility through transparency.

DOC technology addresses particularly well one of the main problems of complex systems construction: *integration*. If we consider the interaction between two pieces of code (the client and the server) we can identify four relative positions (i.e. four coarse types of integration mechanisms): *in-thread*, where client and server are parts of the same thread and interaction is done by method call; *in-process*, where client and server are parts of the same process but in different threads and requests are usually based on ITC (Inter-Thread Communication) mechanisms; *in-host*, where client and server are in different processes and requests are based on operating systems IPC (Inter-Process Communication); and *in-net* integration, where client and server are in different hosts and requests use some form of remote procedure call (RPC).

Middleware is a generic name used to refer to a class of software whose sole purpose is to serve as glue between separately built systems. It tries to simplify the implementation of clients and servers for different relative locations;

0-7803-9568-9/05/$20.00 ©2005 IEEE

for example making possible the implementation of clients that are unaware of server locations. A large simplification is achieved by using the same interface to be used by client and servers independently of the underlying integration mechanism; i.e. the same interface is used to wrap an IPC and an RPC. However, the really large step is when this interface is independent of the relative location of the other object, i.e., location transparency. Brokering middleware is based on the use of an intermediary entity between the client and the server: *the broker*. The process of remote invocation is decomposed in eight steps:

1) The client makes a call to the client stub (the client plug to the broker).
2) The client stub packs the call parameters into a request message and invokes a wire protocol.
3) The wire protocol delivers the message to the server side stub (the server plug to the broker).
4) The server side stub then unpacks the message and calls the actual method on the object.
5) (6,7,8) The response - if any - uses the same process to reach the client.

There are many contenders in the object-oriented middleware arena. The three main technologies are Microsoft's COM+, Sun Microsystems' Java RMI and Object Management Group CORBA. We use the last one due to its better suitability.

A. CORBA

CORBA is the acronym for Common Object Request Broker Architecture, OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks (see [1]). Using special protocols, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network. As we will see, using CORBA appears to be a good approach when developing control systems, because of its inherent modularity and transparency. There are several ways in which CORBA can be used to implement control system components (see [2]). Each of them has its advantages and disadvantages.

CORBA is designed with the following goals in mind: *(1) object-orientation*; remote operations are grouped into interfaces, similar to classes in object-oriented programming languages. An instance of an interface is known as a CORBA object. *(2) Location transparency;* a client does not need to know the location of the object (local or remote). Operations are always invoked with the same syntax. *(3) Programming language neutrality;* CORBA, in contrast to, e.g., Java RMI, is not dependent on any single programming language. *(4) Support for bridge interoperability;* the core specification of CORBA contains an internetworking architecture that allows CORBA to operate in conjunction with other distributed computing technologies, e.g., DCE Remote Procedure Calls and Microsoft's DCOM.

The CORBA technology consists of three main parts: the CORBA distributed object model, CORBA services and facilities, and the CORBA component model.

1) CORBA Distributed Object Model: The distributed object model enables the implementation of distributed object-oriented client-server applications. The Distributed Object Model is based on the following parts:

Interface Definition Language (IDL): used to define the interface of a CORBA object, independent of programming language, but mappeable to all of the popular programming languages (C,C++, Java, Ada, Lisp...) via OMG standards.

The Object Request Broker (ORB): contains the necessary infrastructure that enables clients to invoke operations on CORBA objects.

Object references: object references are the basic entity for encapsulating the type and location of a CORBA object. Are represented as runtime objects.

Object adapters: the object adapter is the part of the ORB that is responsible for providing the necessary mechanisms for associating a CORBA object implementation with a particular IDL interface.

Inter-ORB protocols: CORBA Inter-ORB Protocols (IOP)s define interoperability between ORB end-systems. The General Inter-ORB Protocol (GIOP) is the basis for all IOP's. The mapping of GIOP onto TCP/IP is known as the Internet Inter-ORB Protocol (IIOP). This is the default protocol used by commercial ORBs.

CORBA Messaging: CORBA 2 supports three communication models: synchronous two-way communication, one-way communication and deferred synchronous communication.

2) CORBA Services and Facilities: CORBA Services provide pre-built functionality for the construction of applications from CORBA building blocks. A large number of services have been defined, e.g., Collection Service, Concurrency Service, Enhanced View of Time, Event Service, Externalization Service, Licensing Service, Life Cycle Service, Naming Service, Notification Service, Persistent State Service, Property Service, Query Service, Relationship Service, Security Service, Telecom Log Service, Time Service, Trading Object Service, and Transaction Service. CORBA Facilities are similar to services (but coarser). They include facilities for Internationalization and Time, and Mobile Agents. Two of the services that are of particular relevance to real-time communication are the Event Service and the Notification Service.

3) CORBA Component Model: The CORBA Component Model (CCM) (see [9]) and the "Lightweigth CCM" (see [10])are OMG specifications that give the basis to create *component-based systems*. A *component* is a CORBA meta-type which is an extension of the *object* meta-type. It is a specific collection of features and services that can be described by an IDL component definition, which acts as a "black box" that can be plugged *as is* in a particular system.

Representation and implementation of a component functionality are encapsulated, so that reusability and retargetability are guaranteed. This is achieved by defining a set of enti-

ties such as Containers, Homes, Executors, *etc.*, and a set of procedures and programming models such as the Component programming model, the Container programming model or the Package and deployment procedures. In figure 1 a schema of what CCM implies can be seen.

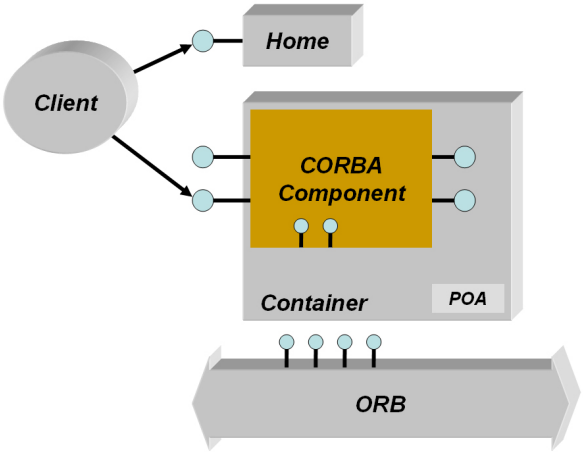

Fig. 1. CORBA Component Model

From the point of view of the component user, the CORBA Component Model technology makes it possible to develop "software bricks" that can be used to build complex and large systems in an easier way. From the point of view of the component developer, CCM provides an useful infrastructure to implement CORBA objects faster. To give an example, by using CCM the component developer does not have to write any CORBA initialization code any more.

III. Real - Time CORBA

There are some issues in CORBA that are especially relevant for distributed control systems engineering: predictable behavior, fault tolerance and execution in an embedded environment. The Real-time platform task Force of the OMG is addressing all these topics and focuses their activities on real-time systems, which often are also embedded and have fault tolerance requirements. The Real-time PSIG goal is the recommendation of adoption of technologies that can ensure that OMG specifications enable the development of real-time ORBs and applications. To achieve this goal, the Real-time PSIG gathers real-time requirements from industry, organizes workshops and other activities and involves real-time technology manufactures to elaborate Requests For Information and Requests For Proposals for these technologies. The main results of this work can be organized in the three categories: the Real-Time CORBA (RT-CORBA) specification, the Fault-tolerant CORBA specification and the Minimum CORBA specification.

RT-CORBA (see [3]) standardizes the mechanisms for resource control (memory, processes, priorities, threads, protocols, bandwidth, etc.) and handling of priorities in a distributed sense (for example forwarding client priorities to the server). The RT-CORBA 1.0 specification defines standard features that support end-to-end predictability for fixed priority CORBA applications. Standard interfaces and QoS policies are defined and allow applications to configure and control (1) processor resources via thread pools, priority mechanisms, intra-process mutexes, and global scheduling; (2) communication resources via protocol properties and explicit bindings, and (3) memory resources via buffering requests in queues and bounding the size of thread pools. The following are the most import parts of RT-CORBA:

Priorities. Two types of priorities are defined: CORBA priorities and native priorities, as well as the mapping in-between. This allows consistent global priorities in distributed applications with heterogeneous nodes with different priority bands. Two priority models are defined: server-declared priorities, and client-propagated priorities. When using the former model the server decides the priority at which an object invocation should execute on the server-side; with the latter it is the client that declares the invocation priorities which the server then must honor. A server is permitted to define priority transforms which set the priority at which a particular invocation is performed based on e.g., external factors. This can be used to define different types of priority ceiling protocols. Inbound transformations are applied on incoming invocations after reception by the ORB core, but before dispatching to the servant. Outbound transformations are performed when a servant invokes an operation on an object.

Thread pools. The thread pool model allows pre-allocation of thread pools and the setting of thread attributes, e.g., default priorities. To handle request bursts the number of threads is allowed to grow through the creation of dynamic threads. Using thread pools with lanes the threads in a thread pool are partitioned into subsets, each with different priorities or priority bands.

Mutex. In order to avoid priority inversion and ensure consistency between the synchronization mechanisms used within the ORB and the synchronization mechanisms used in the application part of the code RT-CORBA defines a mutex.

Global scheduling service. The global scheduling service allows application developers to express QoS requirements using a higher level of abstraction than what is provided by traditional OS mechanisms. Using the scheduling service it is possible to specify the processing requirements of the operations in terms of, e.g., worst-case execution time or period. This is only an optional part of RT-CORBA 1.0.

Protocol properties. transport-specific protocol properties can be specified, that control various communication protocol features.

Explicit binding. In standard CORBA connections (bindings) between a client and a server are established on-demand. RT-CORBA allows an explicit binding model that allow pre-establishment of connections to servers, and makes it possible to associate priorities with the connections.

Leveraged CORBA 3.0 features. RT-CORBA also leverages a number of real-time relevant features in ordinary CORBA. CORBA Messaging provides policies to control roundtrip timeouts. It also supports reliable one-way com-

munications and type-safe asynchronous method invocation. The Enhanced Views of Time Service defines interfaces to control and access clocks. The RT Notification Service is a planned rt-extended notification service.

CORBA and RT-CORBA contain a number of features that are useful also for hard real-time applications (see [5], [6]), e.g. timeouts, asynchronous invocations, one-way invocations, private and pre-allocated connections, avoidance of priority inversion within ORBs, and consistent global priorities. However, several important issues are not addressed or are lacking.

Deterministic transports. The major source of non-determinism in current CORBA/RTCORBA is the transport protocol. Although CORBA allows the use of other transports, which also may be pluggable, most ORB manufacturers only support IIOP, or only support additional transports that have similar timing characteristics as IIOP/TCP. In order for CORBA to be applicable to hard real-time applications it is necessary to support transport protocols with higher levels of determinism.

Periodic activities. In order to support periodic real-time communication CORBA needs to support the description of periodic invocations from a client to a server, i.e., it must be possible to model information that concerns both the client and the server object as a single entity, and to associate information to this entity, e.g., the period, the amount of data that will transferred, and what the maximum allowed communication latency is. RT-CORBA briefly defines the concept of an activity.

Scheduling. In order to be able to guarantee any communication timing constraints it is necessary to schedule the access to the network. Scheduling requires global knowledge of all network accesses. The need for global information about distributed object invocations does not fit into CORBA very well. One possibility could be to have a special CORBA scheduling object that resides within some node and that is defined using IDL. Another approach would be to introduce the scheduling support as a CORBA service.

Small footprint. CORBA has a reputation of being resource-intensive. RT-CORBA increases complexity rather than decreases it. In order for HRT-CORBA to be applicable to embedded systems, e.g., used in sensors, actuators, and intelligent controllers it must have a small footprint. Hence, it is necessary for HRT-CORBA to build upon the Minimum CORBA specification rather than the RT-CORBA specification. Several of the features of RT-CORBA, e.g., the multiple thread lanes, dynamic thread creation, and thread borrowing are probably not necessary in embedded HRT-CORBA applications.

IV. The PCT Testbed

A. Objectives

The main objective of the distributed Process Control Testbed is to identify (mainly hard real time) requirements for CORBA, RTCORBA and CCM-based distributed control systems, and to perform experiments in conditions of systems heterogeneity and legacy integration. The results of the experiments (including the negative ones) will identify the features needed in CORBA to be used in control systems. The PCT design is derived from the current industrial process control systems: it has to be representative of the basic characteristics of a process plant control system, so the result should be significant for a real industrial environment. The type and number of components as well as the network topology/ies are designed based on this idea. Because of this, the PCT must can change easily and adopt different configurations, architectures and topologies. Finally, it is necessary that the PCT allows mechanisms to make some experiments and to measure the results. Secondary requirements are the cost and safety conditions of the testbed. The PCT design base is a set of use cases selected in order to satisfy the mentioned general requirements. PCT main use cases are: package and deployment, control loops, performance test, interoperability, retargetability, sclalability, intensive data traffic, concurrent access, component replacement, priority management, event generation, PLC integration, legacy systems integration, simulated process plant, physical model based control and distributed simulation. These use cases are the startpoint for the PCT experiments design and for the HS/SW PCT components specification.

The PCT mimics a chemical process plant control system, what can be seen as a redundant network where the nodes are monitoring and control elements. The PCT is essentially a redundant network where components are connected. The hybrid characteristic of current industrial control systems, with distinct networks at different levels, is intentionally eliminated to test the viability of a flat network in control environments. The idea is to try to build such a control system using CORBA components and check whether it is possible to perform the tasks that current systems usually do and to accomplish the tasks that future systems are expected to achieve.

Monitoring tools are used for the measurement of the relevant variables in tests. For distributed real-time systems it is necessary to observe inputs and outputs, but also the timing and order of the executing and communicating. Thus, a global synchronized time base with known precision is needed. In the tests, the behavior of the system is monitored to judge whether the system comply with the requirements. The monitoring tools observe the system behavior at different levels: at network level, at host level and at process level.

A first version was developed in IST HRTC project using CORBA objects. A new, componentized version in the IST COMPARE project.

B. Description

Figure 2 shows the complete topology of the testbed. Testbed instruments (sensors and actuators) are connected to a (actual or simulated) process plant trough a typical industrial distributed control system (DCS), in this case the TPS from Honeywell that constitutes a legacy system in this context, or directly connected to an Ethernet control network. Human-machine interfaces (HMI), engineering station and history databases are included. Sensing and acting devices

are wrapped into CORBA compoments, as well as database, simulator (based on ABACUSS II simulation library), PLC and TPS. Controllers are pure CORBA components. The engineering station is the heart of the testbed, since component deployment is controlled from there. The HMI is not a CORBA component but a common CORBA client, and it is used to control and supervise all PCT component operations.

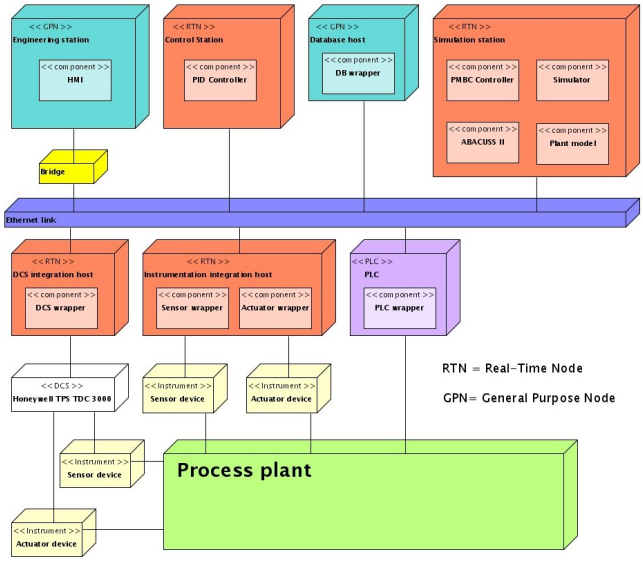

Fig. 2. Process Control Testbed elements

The physical plant design has been choosen to be simple. The nature of the chemical process is not relevant for the PCT objectives, so an elementary acid-base neutralisation process is selected. The physical plant is constituted by a set of three positive displacement pumps, a pH sensor, a temperature sensor, three tanks (acid, base and product), a reactor, a heater module and the corresponding tubing and wiring. Two control loops are needed for the plant: pH control loop, and temperature control loop.

C. Experiments

Package & Deployment: the aim of this experiment is to test the CCM package and deployment procedures when setting up a small system constituted by a sample component and a CORBA client. The component is deployed from the engineering workstation and is tested by using a plain CORBA client. From this experiment we obtain the main requirements for CCM components deployment.

CCS control loops: the purpose of this experiment is to demonstrate the use of CORBA components for the implementation of control loops. From this experiment, basic requirements for CORBA Control Systems (CCS) are elucidated. A simple regulatory control loop is implemented, with three components: sensor, actuator and controller, each of them running on indepentent nodes through the Ethernet network. Two aditional nodes are used: HMI and historical database. The neutralization process is controlled through the pH sensor and by adding the needed reactant at each time. Time series of values of the process are logged in the database, and there are both offline and online versions of the experiment.

Performance test: this experiment is designed to find out similarities and differences among a system developed with plain CORBA and another system with the same design but developed with CCM. Operation times, usability and quality of service are analyzed.

Interoperability: the purpose of this experiment is to demonstrate the possibility of use different operating systems and architectures when setting up a CORBA-based system, in particular the CCS. Several components of the testbed are deployed and executed in different platforms.

Retargetability: the possibility of deploying a component to a different host without the whole system being too much affected is an important issue regarding the CCS maintenance. This process of re-deployment of a component is analyzed in this experiment.

Intensive data traffic: the purpose of this experiment is to check capacity limits of the system when the number of information and control signals, and system nodes, increases significantly. During the test, an increasing number of virtual (simulated) instances of sensors, controllers and actuators are launched. Network overhead is measured meanwhile.

Concurrent access: this experiment shows the concurrency issues in the CCS. Several controllers (increasing in number) access one single node, a sensor, simultaneously. The node performance is monitored in order to detect possible variations.

Scalability: although scalability should have been proven in both the two previous experiments, this experiment is designed to analyze the scalability of the CCS in a more detailed way. The system complexity is increased progresively and effects to the system performance are registered.

Component replacement: the purpose of this experiment is to analyze system maintenance capabilities, regarding those cases when a component of the system shall be replaced in order to be revised or updated. The component replacement should not affect the whole system operation.

Priority management: this experiment is designed to test the priority management of CORBA. Priority inversions must be avoided.

Event generation: chains of event consumers and event generators are implemented within the CCS.

Legacy systems integration: the purpose is to demonstrate the possibility of integrate a legacy system, the Honeywell Total Plant Solution TDC 3000, into the CCS. The TPS is wrapped with a CORBA component to become a node of the CCS.

Simulated process plant: in this experiment the controller node interacts (in several ways) and controls a simulated plant instead of the physical one. The simulation model is created with ABACUSS II, which is also wrapped as a CORBA component into the CCS. The experiment is performed offline.

Physical model based control (PMBC): plant control

is achieved by using a PMBC controller implemented with ABACUSS II, and integrated into the system.

V. Experimental Results and Conclusions

The experiments of CCS control loops show that the timing properties of the control loop are sufficient for process control, where reaction times go from 5-10 ms in the field level to 100 ms in the control network level. The loop cycle of the experiment is around 10 ms in both cases (hub and switch). The overhead imposed by using the CORBA middleware is low and non significant. In the actual process industry CORBA should go embedded in the instrument itself, so the footprint should be quite small as the memory of these devices is low. CORBA calls should be non-blocking (oneway) in order to avoid additional latency and to get stalled when an instrument fails. CORBA implementation should allow that a client be alive even when the server goes down, and to automatically detect when the server goes up again and connect to it.

For the experiment for legacy systems integration, the possibility and characteristics of the integration of legacy systems in CCS are fundamentally determined by the facilities provided by vendors of that system, not CORBA. The integration of the Honeywell TPS in a CCS system has been demonstrated as possible but it is very constrained in capacity and scan period. Additionally, there is uncertainty in the temporal behaviour.

The integration with ABACUSS II has been easy using CORBA. This was possible due to the availability of the simulator as a library; the use with commercial simulators is not so straightforward. The Cape Open initiative could be a way to achieve a more wide and generic integration between CORBA/CCM components and COTS simulators. The use of real time simulation online needs to extend CORBA to handle the notion of time to interact with the simulator. One approach is to use the standard RTI (HLA) for distributed simulation and extend it to real time.

In the intensive data traffic experiment, tests performed on the Hub network show that the loop performance degrades under a heavy load on the network. The single collision domain makes that the latency increases as well as its jitter. The switched Ethernet can cope with the heavy load of the network but there is a limit which is set by the capacity of the switch buffers. A Switched Ethernet could be used then for process control without further consideration. The use of CORBA with a standard widely used network like Ethernet is appealing for the process control domain as the control layers can be flattened and homogenised. Costs (first installation and maintenance) can be reduced and real-time plant information can be made be available to any node in the system. This obviously poses a security problem (and possible network degradation) so it is critical to control the information flow between the control and the business layer.

In the experiment about concurrent access to resources system operation is also affected by the concurrency access in both hubbed/switched networks, although results are still good for process control. A priority policy is needed for process (and any other) realtime complex control systems. Priorities tend to grow as going down to regulatory and safety control loops. But for large and complex control systems where predictability (or at least a bounded worst case) is a must it is advisable to use deadlines instead of priorities. This is something that has to be implemented in CORBA. CORBA has proved to handle very well requests at a very high rate as all the elements performed quite well in these experiments.

The PCT testbed evaluation criteria depends on the experiments done because this is what enable us to identify new CORBA requirements for distributed control systems. CORBA is a potential element to incorporate to process control systems. Many features make it really attractive but there are features missing as deadlines for requests. The overhead imposed is not significant for the loop timing properties, it can cope with concurrent requests and it works well with multiple objects (around two hundred objects and 6000 thousand signals were alive in the intensive traffic experiment). A CORBA/CCM feature that was very useful in the implementation and testing process was location transparency. This is extremely valuable as enables some dynamism in the allocation of objects to nodes.

Due to the additional complexity, and limitations of the platforms fault tolerance mechanisms have not been implemented in the PCT. Control system configuration is analysed through CCM package and deployment procedures, from the engineering station node. Real-time is not very exigent in most of the process control applications. Lag times in instruments and equipments are in the order of, at last, hundreds of milliseconds and the networks used up to day are much less than what we have in Ethernet. So, real time Ethernet is the best solution (of the two alternatives considered) to use with CORBA/CCM in process control systems as it can provide a predictable but more flexible environment and the use of a widely used technology as it is Ethernet.

References

[1] OMG, 2002, *CORBA, CommonObject Request Broker Architecture Specification* 3.01, Object Management Group , Needham, MA, USA.

[2] Sanz, R. and Alonso, M. 2001, *CORBA for Control Systems,Annual Reviews in Control*, vol 25, pp. 169-181.

[3] Schmidt, D. And Kuhns, F. ‚June 2002, *An Overview of the Real-Time CORBA Specification*, IEEE Computer.

[4] Astrm and B. Wittenmark, *Computer Controlled systems*. Third Ed. Prentice-Hall. New York, NJ 1997.

[5] H. Kopetz, *Real Time Systems: Design Principles for Distributed Embedded Applications*. Boston, MA; Kluwer, Academic Pub. 1997

[6] J. W. S. Liu, *Real Time Systems*. Upper Saddle River, NJ: Prentice Hall, 2000

[7] D. Schmidt, *Overview of CORBA*, www.cs.wustl.edu/ schmidt/corba-overview.html

[8] Shokri, Eltefaat and Philip Sheu, *Real time Distributed Object Computing: An emerging field*. IEEE Computer (pp 45-46) 2000

[9] OMG, 2002, *CORBA Component Model Specification* 3.0, Object Management Group , Needham, MA, USA.

[10] OMG, 2002, *CORBA Lightweight Component Model Specification* 1.0 (adopted), Object Management Group , Needham, MA, USA.