HARD REAL-TIME CORBA (HRTC) FOR PROCESS CONTROL SYSTEMS [2]

Santos Galán, Manuel Rodríguez, Ricardo Sanz [1]

ASLAB, Universidad Politécnica de Madrid, Spain

Abstract: Control systems for process plants are complex applications running in several interacting computers with varying degrees of integration. The construction, deployment and maintenance of the software system is a difficult problem and distributed object oriented technology offers a good way to deal with it. The open standard CORBA provides flexible middleware capable of integrating complex applications in heterogeneous environments, but was originally designed with large business applications in mind and is not perfectly suited for the construction of control systems. Even with recent advances in the real-time specification for CORBA, it is only suitable for soft real-time applications and do not deal with the tight requirements of closed control loops. In this paper, the building of a process control testbed to identify requirements for CORBA control systems, with both predictable and event-driven transports, is presented. The benefits of such technology are discussed.

Keywords: Real-Time, CORBA, Process Control, Distributed Object Computing

1. INTRODUCTION

Most present-day plant-wide control systems are very complex, constituted by diverse hardware and software components which interact with each other. With the incorporation of intelligent sensors, the computers reach even the lower level of the control hierarchy. They are also distributed systems, different tasks run on different processors (computers, networks interfaces, PLC's...) and common resources are shared between processors. Distributed systems are designed to improve performance and increase system reliability in order to meet timing, resources and concurrency constraints on each node.

Control systems have been traditionally separated into several levels:

(1) Field level. This level is dedicated to the instruments (sensors and actuators) and basic regulatory control. It is communicated via fieldbus.
(2) Process control level. This level takes over the advanced and supervisory control, including local optimization. It is communicated via an Ethernet based protocol.
(3) Business level. The upper level is dedicated to global optimization, scheduling and planning. It is communicated via Ethernet.

Although these levels have been always present in the process industry the control implementation has been evolving along the years. From the first direct digital control where all the devices were connected separately to the control room where the control was centralized to a single computer; to the traditional Distributed Control System (DCS) implementation where several devices are linked to a controller and there are several distributed controllers that are connected to the DCS

[1] corresponding author, ricardo.sanz@aslab.org
[2] The project is funded by the European Commission as IST-37652, "HRTC, Hard Real-Time CORBA".

console; to the future where the control is totally distributed to field with the loops in individual devices. Nowadays we are still in the traditional DCS but migrating slowly to the future

To implement these coming distributed systems efficiently and with enough flexibility, middleware seems to be the most appropriate tool to simplify the task. The construction, deployment and maintenance of the software system is an extremely difficult problem. Even though there are no silver bullets, object oriented technology offers a good way to build complex systems and when they are running in several, networked computers, distributed object technology has been demonstrated as a feasible way to cope with this complexity while keeping costs under control.

CORBA (Common Object Request Broker Architecture, (OMG, 2000; OMG, 1999; OMG, 1998)) is an open standard which provides developers of distributed systems with a flexible middleware capable of integrate complex applications in heterogeneous environments. It should not matter the programming language or operating system chosen to be part of the system, CORBA makes it possible through a feature called interoperability.

In the global Distributed Object Computing (DOC) landscape, CORBA is a well known framework for the construction of modularised, object oriented, distributed applications. It was designed from the perspective of surpassing heterogeneity barriers and provide support for modularity and reuse. CORBA, however, was originally designed with large business applications in mind and is not perfectly suited for the construction of embedded control applications. This has changed recently because the RT (Real-Time) SIG (Special Interest Group) inside the OMG is very active in the development of specifications for this field: Real-time CORBA has found its place into mainstream CORBA specifications. This makes CORBA a specification that deals with real-time issues from the very core (a real difference from other distributed objects technologies).

Some people in the process industry consider that CORBA is an alternative replacement of OPC, but this is based on a lack of understanding of both technologies. CORBA is not a service but a middleware technology that happens to be better than COM (the software under OPC). Particular services —like OPC— can be built and delivered atop of it. In fact there is an OMG specification that provides a complete replacement with enhancements of OPC servers (it is called HDAIS). CORBA is much wider in scope than OPC and technologically more sound and powerful than COM. For example, CORBA specifies mechanisms for real-time behavior or fault tolerance that is basic for the construction of control applications.

CORBA is used by the process industry. RiskMan (Sanz *et al.*, 2000) (based on the ICa (Sanz *et al.*, 1999*a*) broker) is a system for emergency management in a chemical complex with nine plants (see Figure 1). The system supports the whole life-cycle of emergencies: prevention, detection, firing, diagnosis, handling, follow-up and cancellation. The application is composed by a collection of CORBA objects running on heterogeneous platforms (VAX/VMS, Alpha/UNIX, x86/Windows NT) performing an heterogeneous collection of functions: expert systems, user interfaces, wrappers of real-time plant databases, data filters based on fuzzy rules, predictors based on neural networks, etc.

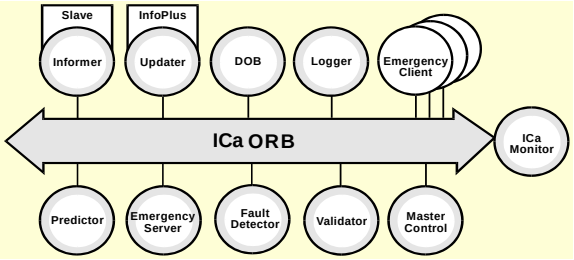

Fig. 1. Some of the CORBA objects that compose the RiskMan application. Informer and Updater are wrappers of external systems.

HRTC uses a Real-time Object Request Broker (developing a prototype implementation of a hard real-time network transport), to build a Process Control Testbed (PCT) that address issues of hard real-time composability in heterogeneous applications, identifying the requirements for hard real-time distributed control systems using CORBA technology. The testbed experiments should be able to prove some of the benefits that such distributed object computing can bring to the process control field.

The paper is organized as follows: Section 2 summarizes the basic concepts of Real-Time CORBA. Section 3 discusses the changes proposed in CORBA needed to achieve hard real-time performance. Section 4 presents the Process Control Testbed and the experiments to identify requirements for HRTC. Section 5 concludes the paper discussing the expected benefits of the technology.

2. CORBA, REAL-TIME AND REAL-TIME CORBA

2.1 What is CORBA?

CORBA is an open standard that allows programmers to specify interfaces as contracts between

servers and clients (these roles are classic in distributed applications). These interfaces are specified using a language called IDL (Interface Definition Language). IDL is used naturally with object oriented programming languages, which map IDL types to their native types after passing IDL files through an IDL compiler. Basically, an interface is an object service contract, implemented by a server, and a way to decouple it form its implementation so that changes to an implementation do not involve a whole re-compilation of the system.

CORBA's key entity is called Object Request Broker (ORB). An ORB is a software bus capable of transmitting messages through a network, from clients to servers, in a transparent way. Clients invoke server methods through a proxy or stub; the ORB locates the server, transmits the invocation from client to server and after the server executes the operation, brings back the results to the client. The servant is a server object which implements an IDL interface and is plugged to the ORB via an object adapter; the most common being the Portable Object Adapter (POA).

2.2 What is Real-Time?

A good definition of the field of hard real-time systems is provided by Douglas Locke (Locke, 2000) from TimeSys:

"**What is real-time?** A real-time system (as defined by IEEE) is a system whose correctness includes its response time as well as its functional correctness. In other words, in a real-time system, it not only matters that the answers are correct, but it matters when the answers are produced. Note that by this definition, systems requiring a defined Quality of Service are usually real-time systems, although they might not use those words to describe themselves.

What is hard real-time? Hard real time means that the system (i.e., the entire system including OS, middleware, application, HW, communications, etc.) must be designed to GUARANTEE that response requirements are met. It doesn't matter how fast the requirements are (microsecond, millisecond, etc.) to be hard real-time, just that they MUST be met EVERY TIME."

Some applications like cellular phones, web servers or digital television need real-time behaviour but in most cases they do not need hard real-time. Other applications like aircraft or process control are presently built as soft real-time but in its very nature, they pose hard real-time requirements to systems developers.

2.3 Real Time CORBA

RT CORBA is an extension of the CORBA standard whose intention is aiding the design of real-time distributed applications. RT CORBA defines CORBA priorities which have corresponding native priorities on each operating system. The interface PriorityMapping is responsible of this conversion. There are two priority models of distributed priority handling:

- Client Propagated: The server honours the priority requested by the client, who sends it along with the invocation.
- Server Declared: The server establishes its own priorities and ignores client requested priorities.

What a client can do using RT CORBA is:

- Set a priority or band of priorities for a given connection.
- Obtain a private transport (non demultiplexed connection) to a servant, so that the connection is not shared with other clients.
- Set a timeout on an invocation.

What a server can do:

- Manage execution of threads through Threadpool interface. Threads can be preallocated (so that server is limiting the number of incoming requests with the possibility of buffering requests that cannot be dispatched) and partitioned in priority lanes responsible of managing requests with a priority bounded to a certain range.
- Select a priority model (Client propagated vs. Server declared).
- Create a Mutex so that the client can prevent other server threads to access certain piece of code of the server.

Both client and server can select a communication protocol and configure certain protocol parameters.

RT CORBA also defines a service called Scheduling Service. This service is designed to work in a closed environment, where clients and servers can be considered a static set, with fixed priorities. Scheduling service provides global scheduling policies, associating names with scheduling parameters. RT CORBA 1.0 does not provide dynamic scheduling. A new extension (RT CORBA 2.0) specifically addresses dynamic scheduling.

3. HARD REAL-TIME CORBA

The CORBA object model (and the development processes and tools associated with it) is extremely adequate for the construction of complex distributed applications and hence the interest in

extending it to be useful in the real, embedded control domain. But there is a problem. Present day CORBA specifications are suitable only for soft real-time applications. CORBA and its extension RT CORBA are not fully suitable to implement these systems because:

- They have only been designed to build systems with soft real time requirements.
- CORBA lacks of a real-time interoperable protocol, necessary to integrate control and real time systems. Neither GIOP nor IIOP are reliable or predictable enough.
- The Scheduling Service is incomplete, can not be dynamically reconfigured and does not provide a wide range of scheduling algorithms.
- Most real time systems are also embedded ones. There is an effort called Minimum CORBA to build a small ORB, tailoring it to fit in embedded systems, but this seems to exclude RT CORBA which increases ORB size.
- Interface specification needs to be extended to express temporal issues.

The analysis of hard-real time requirements posed by CORBA-based distributed control systems shows the necessity to develop theory and technology for hard-real time applications, extending the set of CORBA specifications with interfaces that deal with hard real-time issues.

3.1 What does CORBA need to be Hard Real-Time CORBA?

Some authors claim that advances in real-time distributed object computing can be achieved only by systematically pinpointing performance bottlenecks; optimising the performance of networks, ORB endsystems, common services, and applications; and simultaneously integrating techniques and tools that simplify application development. We believe that a sound engineering approach to system design is also necessary.

Building hard real-time systems with stringent constraints requires the election of an appropriate environment which includes:

- Choosing real-time operating systems for critical nodes: with real-time I/O subsystems and with real-time scheduling.
- Choosing predictable (usually high-speed) network interfaces, communication protocols or industrial backplanes suitable for real-time applications like ATM, CAN, VME, switched fabric, fieldbuses, etc. They must be highly predictable and provide flexibility of control (because TCP/IP, GIOP or IIOP are not very suitable).

RT CORBA 1.0 is thought to be used with static systems, where processes, clients, servers and tasks are perfectly known and let us determine the best policies for our system. This not flexible enough and does not provide needed reconfiguration capabilities. RT CORBA 2.0 includes dynamic scheduling but it is still not enough.

In our opinion, CORBA needs to be extended in certain aspects because:

- CORBA requires a deterministic transport and a reliable and interoperable RT protocol, whose QoS parameters can be modified through CORBA interfaces.
- RT Scheduling need support for dynamic algorithms and support for advanced feedback scheduling.
- CORBA interfaces must be specified not only in the value domain but also in the temporal domain.
- Another problem is global time synchronization. Deployment over time triggered platforms can provide enhancements in distributed time.
- Meeting hard real time requirements includes validating them. This can be done with interceptors, but it is a time consuming way. Maybe some other method should be used to do this.

4. PROCESS CONTROL TESTBED

In order to identify (mainly hard real-time) requirements for distributed control systems and perform experiments in conditions of systems heterogeneity and legacy integration a Process Control Testbed is used. Experiments will be done using conventional IIOP and a new real-time protocol.

Figure 2 shows the complete topology of the proposed testbed. This final structure should be reached in several stages of increasing difficulty where different experiments are run.

The PCT tries to represent the basic characteristics of a process plant control system network with advanced features not found in current designs, like the flat two control networks (Ethernet and TTP/C (TTTech Computertechnik, 1999; Kopetz, 1997)) where all the elements are linked. Several instruments (sensors and actuators) are connected to a (actual or simulated) process plant in three different ways:

(1) Trough a typical industrial distributed control system (DCS), in this case the TPS from Honeywell that constitutes a legacy system in this context, with its own controller and user interface. The TPS communicates with the Ethernet control network.

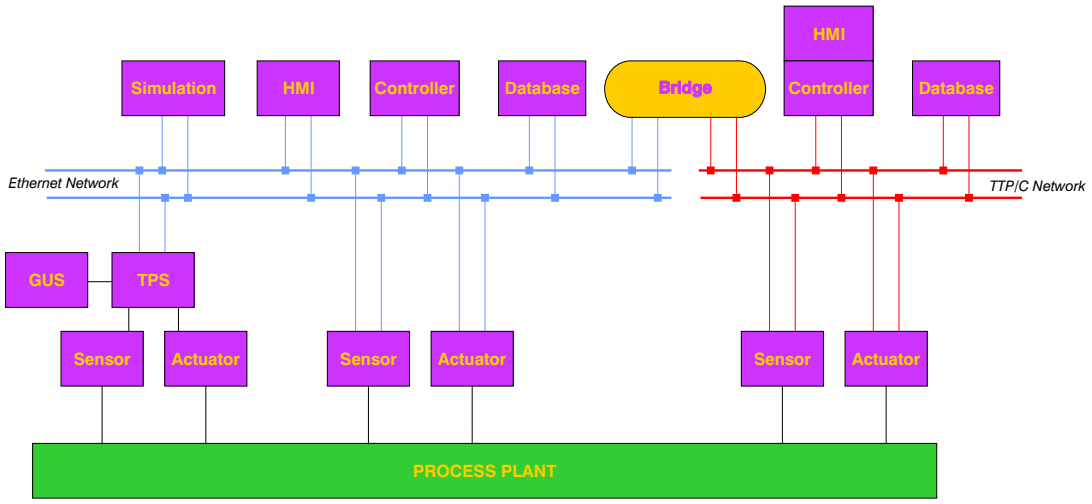

Fig. 2. Schema of the Process Control Testbed

(2) Directly connected to an Ethernet control network.
(3) Directly connected to a time-triggered network (TTP/C).

Apart from the TPS monitoring and controlling devices, both networks include controllers, human-machine interfaces (HMI) and history databases. This is not the typical configuration in industrial practice, where separate networks are used. Finally, one or several simulation nodes are included on the Ethernet network. The Ethernet and time-triggered networks communicate through a brigde.

4.1 Closing control loops through networks

A simple regulatory control loop with three components: Sensor, Actuator and Controller, built as independent nodes connected through the Ethernet and the TTP/C networks are tested (figure 3). Also there should be two additional nodes: HMI and History Database. For the TT network the HMI and the controller are in the same node.

In the experiment, an operating elemental process plant, such as a neutralization tank with a pH sensor is controlled by the adition of a reactant with a volumetric pump. The time series of values of the process variables are recorded on the history database and shown on real time on the HMI. The operator can change the setpoint through this HMI node.

This is apparently a simple experiment but its success will demonstrate the use of CORBA for control systems. When designing a distributed real-time system, scheduling of common resources is a key problem. For distributed control systems where control loops are closed over a communication network or a field bus, the network can be a bottleneck.

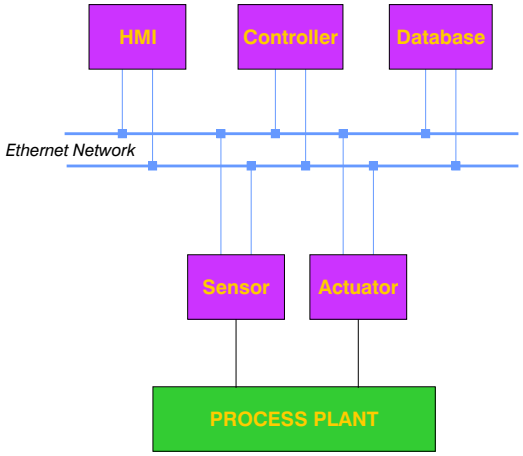

Fig. 3. Control loop with CORBA components over Ethernet

A similar experiment will be run over a TTP/C network, comparing both results.

4.2 Legacy system integration

This experiment is aimed to demonstrate the possibility of integrating legacy systems in a CCS (figure 4). As indicated above, a Honeywell TPS, with its graphical user interface (GUS) is used.

4.3 Interaction of simulation objects with control agents

A test for the integration of heterogeneous components over the same network is the connection of simulation modules that interact with other objects in the system (figure 5). The use of mathematics models in control and monitoring functions is continuously increasing and control systems should accommodate easily this components.

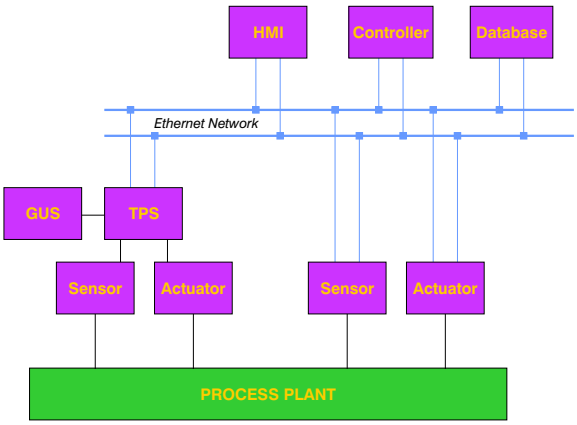

Fig. 4. Schema of the PCT for integration of DCS (legacy system)

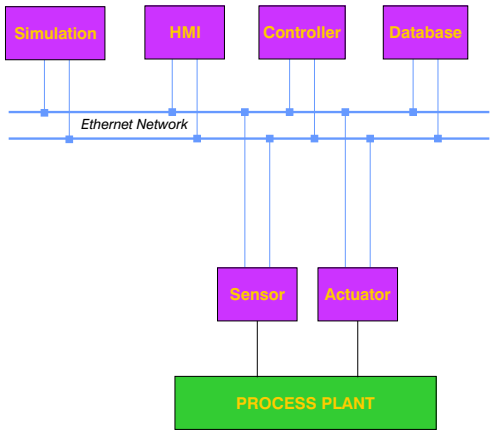

Fig. 5. Integration of simulation objects in control network

5. CONCLUSIONS

CORBA provides the middleware capable of integrating complex applications but it needs to be upgraded to hard real-time to be applied in process control systems.

Perhaps the main question is: Why do we need the integration provided by HRTC in process control systems? Beyond many obvious answers (simplicity of flat network, use of heterogeneous components like optimization or simulation, vendor independence, reduction in cost, etc.) we would like to stress one: The modular approach fostered by CORBA will let us develop true modular control systems.

The second point we want to mention is design freedom. Design freedom is necessary in the complex control systems domain to explore alternative controller designs. Excessively restrictive technologies will collapse - unnecessarily - dimensions of the controller design space (Shaw and Garlan, 1996). This is, for example, the case of some fieldbus technologies that support several slaves but only one master. While design restrictions (in the form of prerequisite design decisions) simplify development, they sacrifice flexibility. Can we get both, simple development and flexibility? The key are frameworks where design dimensions are still open even when pre-built designs are available. To continue the example of the fieldbus, the one-master/several-slaves approach is one type of pre-built, directly usable, design; but the underlying fieldbus mechanism should allow for alternative, multi-master designs. This can be done by means of the development of agent libraries that provide predefined partial designs in the form of design patterns (Sanz *et al.*, 1999*b*), and a transparent object-oriented real-time middleware like the one proposed in HRTC. This approach will let developers construct their own agencies to support their own designs.

REFERENCES

Kopetz, Hermann (1997). *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers.

Locke, Douglas (2000). Real-time linux – what is it, why do you want it, how do you do it?. At6090565653.html. LinuxDevices.com. http://www.linuxdevices.com/articles/.

OMG (1998). A discussion of the object management architecture. Technical report. Object Management Group. Falls Church, USA.

OMG (1999). Real-time CORBA adopted specification. Technical Report orbos/99-06-02. Object Management Group. Falls Church, USA.

OMG (2000). Common object request broker architecture and specification. Release 2.4. Technical report. Object Management Group. Falls Church, USA.

Sanz, Ricardo, Miguel J. Segarra, Angel de Antonio and José A. Clavijo (1999*a*). ICa: Middleware for intelligent process control. In: *IEEE International Symposium on Intelligent Control, ISIC'1999*. Cambridge, USA.

Sanz, Ricardo, Miguel J. Segarra, Angel de Antonio, Fernando Matía, Agustín Jiménez and Ramón Galán (1999*b*). Design patterns in intelligent control systems. In: *Proceedings of IFAC 14th World Congress*. Beijing, China.

Sanz, Ricardo, Miguel Segarra, Angel de Antonio, Idoia Alarcón, Fernando Matía and Agustín Jiménez (2000). Plant-wide risk management using distributed objects. In: *IFAC SAFEPROCESS'2000*. Budapest, Hungary.

Shaw, Mary and David Garlan (1996). *Software Architecture. An Emerging Discipline*. Prentice-Hall. Upper Saddle River, NJ.

TTTech Computertechnik (1999). *Specification of the TTP/C Protocol*.