CORBA Control Systems Request for Information

Abstract

This Request for Information (RFI) solicits information about requirements, technologies, and software products in the area of software interface elements for implementation of networked closed loop control systems based on CORBA technology.

Release 0.1

OMG Number realtime/2003-10-01

September 19, 2003

Release Sheet (1)

Release:	0.1
Status:	Draft
Date:	2003/09/19
Revision Scope:	First draft release
Revised Sheets:	All
Comments:	This is the first release of the Control Systems RFI document prepared by the CSWG project that is distributed inside the OMG CSWG community for additions and/or corrections.

Contributors

Ricardo Sanz, UPM, Spain
Karl-Erik Årzén, LTH, Sweden
Anton Cervin, LTH, Sweden
Johan Eker, LTH, Sweden
Santos Galán, UPM, Spain
Dan Henriksson, LTH, Sweden
Manuel Rodríguez, UPM, Spain
Carlos García, Spain
Rafael Chinchilla, Spain

Send Comments to

Ricardo Sanz

Ricardo.Sanz@etsii.upm.es

Universidad Politécnica de Madrid
José Gutierrez Abascal 2
28006 Madrid

Table of Contents

1 Introduction to the RFI

1.1 Purpose

This Request for Information (RFI) solicits information about requirements, technologies, and software products in the area of software interface elements for implementation of networked closed loop control systems based on CORBA technology.

The Object Management Group (OMG) and, specifically, the Control Systems Working Group (CSWG) and the Real-time, Embedded and Specialized Systems Platform Task Force (RTESS PTF), will use this information to begin the technology adoption process which will define an OMG-standardized architecture and set of interfaces which will address safety assurance of Common Object Request Broker Architecture (CORBA)-based software components for distributed control applications.

For additional information about OMG and CORBA, see reference material at the end of this document.

The OMG encourages users, consultants, software development methodologists, and developers of software to become involved with this process by responding to this RFI. OMG non-members and members may submit responses.

The Control Systems Working Group (CSWG) and the Real-time Embedded and Specialized Systems Platform Task Force (RTESS PTF) expect to use responses to this RFI to define one or more RFPs leading to OMG-adopted specifications in the area of software components for distributed control applications. The OMG Technology Adoption Process is run according to the procedures laid down in the *OMG Policies and Procedures*.

1.2 Status

This RFI is in draft status. It has to be discussed and presented in the OMG technical meeting.

2 RFI Details

2.1 Context and Scope

Applications for CORBA-based distributed control include manufacturing, continuous processes, domotics, transportation, utilities, and defense related systems.

2.2 Objectives of the RFI

This RFI seeks information to help the Control Systems Working Group make useful and efficient decisions in software components of control systems technology adoption process.

OMG standardizes interfaces, so information with some bearing on interfaces is desired. The interfaces can be active at the time of analysis and/or design of the behavior of the software, at compile time (including compilation with IDL compiler) and/or at run (execution) time.

OMG would like responders to provide information about available technology to facilitate framing of Requests for Proposal (RFPs), and to provide a set of topics to discuss, for example domain specific infrastructure, platform extensions, IDL extensions, and/or Object Request Broker (ORB) services.

2.3 Information Being Requested

This RFI is seeking information in the categories described below. Respondents are asked to address areas in which they have expertise and/or interest. Please consider the objectives of this RFI when responding so time is spent on issues that will be helpful to reviewers. Respondents may consider areas not explicitly asked for if they feel the information provides useful guidance. In addition, OMG is requesting descriptions of trends of which the Control Systems Working Group (CSWG) and the Realtime, Embedded and Specialized Systems Platform Task Force (RTESS PTF) should be aware as OMG prepares for the technology adoption process.

2.3.1 Existing Implementations

OMG requests information on availability, maturity, and importance of any existing models, products, methodologies, etc. which support the closed loop control concept.

2.3.2 Standards

OMG requests information on relevant standards, both *de facto* and *de jure*. Where multiple standards exist, respondents are asked to compare significant differences among them. It is also important to identify problems with current standards that prevent their acceptance or cause problems in their implementation.

2.3.3 Requirements

OMG requests information on user requirements on control systems performance and software-based control technologies.

2.3.4 Other Information

OMG requests that respondents furnish any other information they think may be relevant.

2.3.5 Topics of interest might include but are not limited to[1]:

1. Executables
   - 1.1. Architecture
     - 1.1.1.Software architectureç
     - 1.1.2.Control architecture
   - 1.2. Infrastructure
     - 1.2.1.Operating System
     - 1.2.2.Middleware (e.g., ORBs, or Messaging Infrastructure)
     - 1.2.3.Services
     - 1.2.4.Containers, Components
   - 1.3. Applications
     - 1.3.1.Control
     - 1.3.2.Monitorisation
     - 1.3.3.Supervision
     - 1.3.4.Visualisation
     - 1.3.5.Storage systems
     - 1.3.6.Sensors and actuators
     - 1.3.7.Simulation
     - 1.3.8.Interoperation

[1] It is not necessary to address all of them in in a response.

2. Processes and methodologies that can be related to OMG specifications (technologies)
    2.1.1. Process steps
    2.1.2. Documentation and other artifacts
3. Design and Verification Techniques
    3.1. Core design
    3.2. Temporal verification
    3.3. Composability
4. Tool Support
    4.1. Code Generation
    4.2. Design rule checking
    4.3. Scheduling Support
    4.4. Language Profiles
    4.5. Support for Certifiability
5. Human Factors Aspects
    5.1. Work flow for operators
    5.2. System ensuring pre and post conditions
6. Related standards and reference documents

2.4 Who May Respond to this RFI

Any person or company may respond to this RFI, including both OMG members and non-members. OMG especially encourages industry, civil and military aviation organizations, users, consultants, software development methodologists, and developers of software to respond to this RFI. However, only Contributing Members of the OMG will be eligible to respond to any follow-on RFPs issued as a result of this RFI. Any company may join OMG at the contributing member level and respond; see the OMG website (http://www.omg.org) for membership information.

2.5 Instructions for Responding to this RFI

Reponses are available to the public.

Organizations responding to this RFI shall designate a single contact within that organization for receipt of all subsequent information regarding this RFI. The name of this contact will be made available to all OMG members.

Responses to this RFI must be received at OMG no later than 6 November 2000(arbitrary) to meet the *OMG Policies and Procedures* requirement that all responses to RFIs and RFPs be available at least three weeks prior to the meeting where the responses will be considered. (See below for more details on receipt dates and addresses). This Documentation submitted in response to this RFI will be available to all OMG members. This reduces the risk that Technical

Committee and Task Force members will arrive at the meetings to review proposals without having seen them and provides time for the OMG to send papers to its members.

NOTE: According to the Policies and Procedures of the OMG Technical Committee, proprietary and confidential material may not be included in any response to the OMG. Responses become public documents of the OMG. If copyrighted, a statement waiving that copyright for use by the OMG is required and a limited waiver of copyright that allows OMG members to make up to fifty copies of the document for review purposes only is required.

2.5.1 Format of RFI Responses

The RFI response can consist of pre-existing product documentation, but should preferably be organized and presented in accordance with this RFI.

The following outline is offered to assist in the development of your response. You should include:

- A cover letter -- the cover letter should include a brief summary of your response such as indicating which areas you are responding to and indicate if supporting documentation is included in your response.
- Your response to any or all of the areas of information requested by this RFI.
- If required, a glossary which maps terminology used in your response to OMG standard terminology. (See the Appendices to the OMA Guide and the CORBA Specification for OMG's standard terminology.)

Although the OMG does not limit the size of responses, you are asked to consider that the OMG will rely upon volunteer resources with limited time availability to review these responses. In order to assure that your response receives the attention it deserves, you are asked to consider limiting the size of your response (not counting any supporting documentation) to approximately 25 pages. If you consider supporting documentation to be necessary, please indicate which portions of the supporting documentation are relevant to this RFI.

2.5.2 How to Submit

OMG requests that submitted material be attached to an email cover letter and sent to our process manager at juergen@omg.org. The preferred formats are ASCII text and Postscript. PDF, MS WORD RFT and Frame MIF may additionally be used. In addition, one paper copy is required (as a backup).

Transportation Task Force plans to review submissions at the December 2000 meeting in and/or at the January 2001 meeting. Responses to this RFI (and other communication regarding this RFI) should be addressed to:

OMG:

Control Systems RFI
Object Management Group Inc.
Framingham Corporate Center
492 Old Connecticut Path
Framingham, MA 01701-4568
USA

Phone: +1-508-820 4300
Fax: +1-508-820 4303
Email: juergen@omg.org

Email responses to this RFI must be received at OMG no later than 5:00 PM US Eastern Time (22:00 GMT) XX April 2004 and the confirming paper copy must arrive at OMG shortly thereafter. The outside of packages/envelopes containing submissions or any other communication regarding this RFI should be clearly marked "Control Systems RFI Response".

2.6 Reimbursements

The OMG will not reimburse submitters for any costs in conjunction with their responses to this RFI.

2.7 Response Review Process and Schedule

2.7.1 Process

RFIs such as this one are issued with the intent to survey the industry to obtain information that provides guidance which will be used in the preparation of RFPs. The OMG membership, specifically the Realtime, Embedded and Specialized Systems Platform Task Force (RTESS PTF), (expand the list) will review responses to this RFI. Based on those responses, the PTF (expand the list) will consider revision / extension of its domain architecture, a corresponding roadmap, and one or more RFPs. In accordance with the OMG technology adoption process, each issued RFP will ultimately result in the specification of a portion of the architecture. See below for a brief description of the OMG technology adoption process and consult the OMA Guide for a complete description of the OMG's requirements, policies, and procedures for technology submissions.

2.7.2 Clarification of Responses

To fully comprehend the information contained within a response to this RFI, the reviewing group may seek further clarification of that response. This clarification may be requested in the form of brief verbal communication by telephone; written communication; electronic communication; or a presentation of the response to a meeting of the CSWG or RTESS PTF.

Therefore, the CSWG and the RTESS PTF request that submitters attend the CSWG and/or RTESS PTF meetings, prepared to present their responses.

2.7.3 Schedule

The schedule for responding to this RFI is as follows. Please note that early responses are encouraged.

PTF recommends issuing the RFI	Nov 18, 2003
RFI issued	Nov 21, 2003
RFI responses due	Mar 21, 2003

The tentative schedule for the RFI evaluation process and subsequent RFPs is:

Review of RFI responses	May 21, 2003
PTF recommends issuing the initial RFP	Sep 21, 2003

NOTE: This schedule is subject to change based on the number of RFI responses received and the information acquired from the responses.

3 Overview of Technology

3.1 Objectives

The objective of this document is to investigate the usage of CORBA technologies in an industrial control system setting. We believe that the potential benefits from using CORBA, which is a mature, well spread, and standardized technology, are enormous with respect to increased design flexibility and better system integration. The introduction of CORBA would enable independence from both hardware/software manufacturers and implementation languages. Using CORBA throughout a whole factory would allow integration between systems at all levels; from high-level administrative and business systems down to individual control loops.

The Common Object Request Broker Architecture (CORBA) is a middleware specification for the development of interoperable, distributed object systems. This report provides a general overview of the topics in distributed object systems, focusing on CORBA aspects that are critical for control systems engineering. Object Management Group (OMG) technology is summarized and some sample applications are presented. CORBA was originally created as an object-oriented component technology for non real-time systems. The main objectives were to create a framework that was flexible and powerful, rather than predictable and light-weight. This makes CORBA a less appropriate tool for time-critical systems, and as a remedy for this, a real-time extension, called RT-CORBA, was introduced in the late 1990s. RT-CORBA addresses the problems pertaining to priority inversions and unpredictability in resource allocations, such as memory reservation and thread mapping. However, RT-CORBA does not address the major source of non-determinism in CORBA, the transport.

Control systems are a wide area. This report and the entire HRTC project focuses on three different types of control systems:

- **The basic control loop.** The basic control loop consists of a single-input, single-output (SISO) loop containing a sensor, a controller, and an actuator. These three parts are assumed to be distributed over a communications network. Hence, the control loop is networked. This type of control loop is at the heart of all types of distributed control systems.

There is also a large amount of theory available for analyzing the effects of communication delays on the control performance.

- **Industrial automation systems.** By this we mean the large, distributed, and hierarchical control systems typically used in process and manufacturing automation. Other names for this type of system are PLC (Programmable Logic System) or DCS (Distributed Control System). The control loop is central also in these systems, but they also contain support for a large amount of additional functionality, e.g., discrete logic control, operator interface, supervisory control and monitoring applications, database access, production planning and scheduling, etc. These systems are typically also programmable, i.e., the user can program different control applications using domain-specific programming languages, as defined by IEC 61131-3. The resulting programs are typically compiled, and the generated code is downloaded to different distributed control systems. The use of CORBA in this type of systems will be illustrated in the Process Control Testbed (PCT).
- **Robotics control systems.** Control systems for industrial robotics must combine flexibility (with respect to new tasks and unforeseen application demands) and high performance (to accomplish productivity and profitability). The control of the manipulator motions has demanding real-time requirements and is typically distributed in nature. At the end-user level special robot programming languages are used or the robot programs are generated directly from CAD data. On this level the real-time requirements are less hard. The use of CORBA in robotics control will be illustrated by the Robot Control Testbed (RCT).

This report studies the following topics:

- What are the potential uses for *real time* distributed object and component technology in automation and automatic control applicationsY Engineering control systems on a systematic, sound way means addressing not only the limited scope of a single controller but also considering a whole family of controllers. A particular control system may be seen just as a single element of a stream of control products.
- Identifying the shortcomings of the current CORBA standards. The fact that RT-CORBA does not guarantee any hard deadlines makes it, possibly, not a very suitable platform for low level feedback loops. However, it is not clear that 100% determinism is needed in order to achieve good control performance. This report shows a few simulations of networked control systems with stochastic timing variations, i.e. jitter and latency.

3.2 The role of object technology

The very nature of control systems is object-oriented (OO) because a control system couples virtual entities with real ones. A controller correlates control design issues and software implementations, which are very conceptual in its nature with sensors, actuators and external world entities, which are very physical objects.

Control software makes a continuous mapping between external and internal entities and hence, object-oriented software is a natural way to build these systems. During the last decade OO technology was relegated from mainstream real-time software because OO implementations introduce computational overhead to support some aspects of OO computation (for example, dynamic binding). While this is usually the case, the computational power of today reduces the influence of this overhead, and OO technology is becoming the technology of choice for building complex real-time systems *because it provides better mechanisms for complexity handling*. An example of grave importance for us is the case of real-time distributed systems, where OO technology is a clear winner [Sho00].

Industrial plants are *Seas of Objects* and software-intensive controllers for them reflect this nature. The natural plant-modeling mechanisms are object-oriented and dealing with preexisting software systems, for example legacy controllers, is best done using object wrapping. Advanced controllers are designed using clear responsibility distribution between control objects.

3.3 Scope

The process of incorporation of information technology (IT) into industrial processes is making profound modifications to production systems. Control and monitoring technology is leaving the *islands-of-automation phase*, entering a new phase of complete systems integration. While enterprise integration architectures (EAI) are hot topics in advanced business engineering, at the production level where controllers live, the incorporation of new technology and designs is confronting difficult problems.

In most cases the problems are mainly due to classical barriers posed to innovation in production systems: lack of predictability, need for non-stop operation, lack of reliability and availability, less than ideal market maturity, exploitation manager's resilience, etc.

Two main objectives are being pursued in this effort, namely Complete Horizontal Integration (CHI) and Complete Vertical Integration (CVI). CHI deals with the integration of business units, business-to-business integration or supply chain integration.

In this report we will address more the topic of CVI. It is time to start thinking in plant-wide integration reaching even the lowest levels in production plants: sensors, actuators and basic controllers.

Complete vertical integration means that integration paths are available from sensors to management information systems (and back). This eliminates some of the limitations that the underlying information technology poses to the design space for monitoring and control systems.

Distributed object computing (DOC) is gaining an increased audience in the IT sector and is the technology of choice for new system implementation. From global experience in last years, it is pretty clear that + besides other advantages - DOC technology enhances systems integrability, simplifying the construction of complex information applications. We will see in this report how a DOC technology, namely CORBA, can supply us with some tools needed for better development of complex, integrated control systems.

3.4 Control engineering processes

Control systems complexity is increasing at a very fast pace in this days. New needs and new capabilities (nobody knows what came first) are driving control systems development into mainstream systems engineering. Integration capabilities are getting progressively critical as system size increments, because modular development is the only known practical way for complex systems engineering.

From this perspective it is surprising, to some extent, the limited role that software technologies play in control engineering journals and symposia. It looks like this technology does not have relevance enough to be considered a research discipline for control engineers (only small-scoped real-time topics are addressed in control engineering places).

The typical development process of a control system can be decomposed, like any other engineering process, in a series of phases that go from the identification of the need to the decommission of the control system.

An example of phasing can be:

1. Problem identification
2. Plant Modeling
3. Control design
4. Control implementation
5. Commissioning
6. Operation
7. Decommissioning

Research in control systems has been mainly focused in the second and third phases, because the first is considered an *a priori* for control engineering (*i.e.* it is always given) and from fourth to seventh they can be left to implementers (*i.e.* to raw work force). The separation between the control laboratory and the real plant is too wide for real engineering.

The basic technology used today to implement control systems is software technology. But, beyond a classical view of digital implementation of controllers [Ast97], software technologies are the basis of modern complex control systems, from SCADA:s and DCS:s to intelligent controllers based on soft computing [Gup96].

Control engineering is about *systems performance*; this means that the knowledge of the controlled system must involve not only the target system but the controller itself, and when controllers are software-based, giving a guarantee on global performance means a clear analysis and deep understanding of software issues. When controller complexity increases there are no available formal methods to guarantee behavior. Only good development processes can provide predictable quality. Good development processes involve all parts of the controller life-cycle; from the problem identification phase to the operation phase and even decommission.

When complexity increases due to software flexibility the probability of failure increases. Systems that were manually operated are now operated by computers and this leads to a critical computer dependence of many artificial systems. The case of the USS Yorktown is paradigmatical. The ship had to go back to the harbor due to a software failure.

While software is becoming a real problem, it is also providing some solutions. For example, advanced research topics on systems fault-tolerance are strongly based in information processing capabilities that are used to detect the fault, isolate it, and devise alternative control strategies that can overcome the fault [Bla00].

3.5 Complex Software for Control

Software systems can range from a small shoe shop database to Star Trek's *USSS Enterprise* control software. In a quick effort we can make a quick and dirty classification of software systems based on factors that induce systems complexity:

- Conventional: the shoe shop database.
- Real-time: meeting deadlines.
- Embedded: run within limited resources.
- Fault Tolerant: good behavior under faults.

- Distributed: run on several interacting computers.
- Intelligent: solving ill-posed problems.
- Large: millions of lines of code.

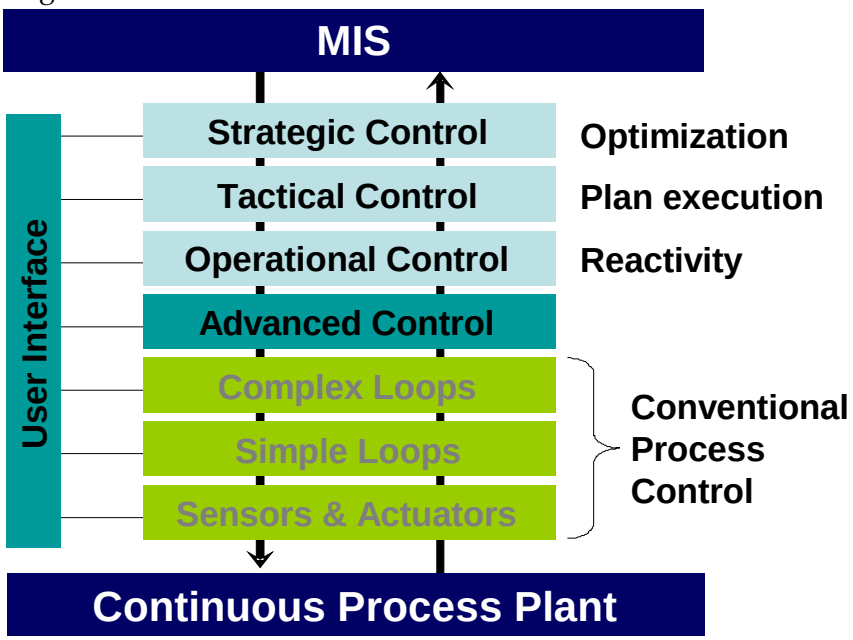

Figure 1: A classical layering of control entities in a complex continuous process control system. Layer quantity and labeling is somewhat field-dependent, but layer roles can be easily mapped from domain to domain.

- Integrated: interoperate with alien systems.
- Heterogeneous: run on heterogeneous platforms.

Complexity factors affect negatively the system development process. Development effort grows with complexity much more than linearly and there are even systems we cannot build; examples are 24x365 systems (total availability), one-shot systems (should work at the first try) or HP-LC (High Performance and Low Cost).

Software engineers have always been "raiders of the silver bullet" looking to solutions to software development problems. Complex software engineering is just an emerging discipline, introduced in frontier areas between those complexity topics mentioned before.

3.6 Complex control systems

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 collection of intelligent roles in complex controllers fitting as interfaces between pre-existent systems (plants, controllers and humans). Examples of these roles are data/action filters and monitors.

Classical hierarchical layering overcomes some of the difficulties of complex systems construction. A example of layering is shown in Figure 1 where some *intelligent* layers are added atop classical control layers in process control systems.

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 V an elusive, difficult to define property V is seen as the core factor affecting systems constructability. Conceptual integrity manifests in several system properties (some of them functional and some non-functional) that are considered extremely important in systems construction. These properties are the basic design principles of systems architecture [Sha96] (See table).

Table1: Architecture design principles

Conforming	Scalable
Suitable	Simple
Composable	Standard
Modular	Proven
Extensible	Performing
Fast	Efficient

We will see in the next section how object technology can provide us with some ideas and tools to approximate this ideal of system conceptual integrity.

3.7 Document Outline

Chapter 2 gives an overview of component technologies and CORBA. Chapter 3 discusses the current state of automation software and how it could benefit from component design. Introducing distributed component technology in a partly time-critical setting will create new demands on both the component technology as such and the design of the application, which in our scenario is the controller. The main complication with the controller design is how to deal with jitter and latency that may be introduced and in Chapter 4 we address these issues from a control theoretic angle. Finally, in Chapter 5 our requirement analysis of Hard Real-Time CORBA is summarized.

4 Distributed Objects and CORBA

4.1 Distributed Object Computing

Distributed Object Computing (DOC) or Object-Oriented Distributed Processing (OODP) is a software model based in the use of services provided by objects that are running in different hosts. Distribution means true concurrence even when most distributed applications serialize behavior of the application using some form of centralized controller.

DOC can be considered a generalization of the client/server model. In DOC, client and server roles are relative to a specific request and not to the whole life-cycle of the object (an object can be the client in a request and the server in the next one).

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. In many cases they are the same as for any other type of system, but in most situations they are of critical importance for control software due to the special requirements posed to control systems. Examples of these benefits are:

- 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.
- Design flexibility through transparency.

4.2 Integration

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 (see Table 2)):

Table 2: Coarse types of integration mechanisms

In-Thread	Client and server are parts of the same thread. Interaction is done by method call. This means serialization (no concurrence) and a simple integration vehicle (programming language routine invocations). This is easy to use, extremely fast and reliable.
In-Process	Client and server are parts of the same process but in different threads. We have inter-thread requests usually based on ITC (Inter-Thread Communication) mechanisms provided by the operating system. This is relatively complex but very fast and reliable.
In-Host	This situation is similar to the previous, but in this case client and server are in different processes. Inter-process requests are based on operating systems IPC (Inter-Process Communication). This is also a fast and reliable mechanism.
In-Net	Client and server are in different hosts. The basic integration mechanism is some form of remote procedure call (RPC). Lower level mechanisms can also be used but in most cases it is not worth the effort. Inter hosts requests rank lower in speed and reliability because it is easier to have different host states in the client, the server or even in the communication channel. Distribution means in many cases unpredictability and unreliability, but also balancing computational load and fault tolerance through redundant servers.

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. Object-oriented middleware is used to simplify the development and use of ubiquitous objects. Middleware tries to simplify the implementation of clients and servers for different relative locations; 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 (See Figure 2). 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,é] The response - if any - uses the same process to reach the client.

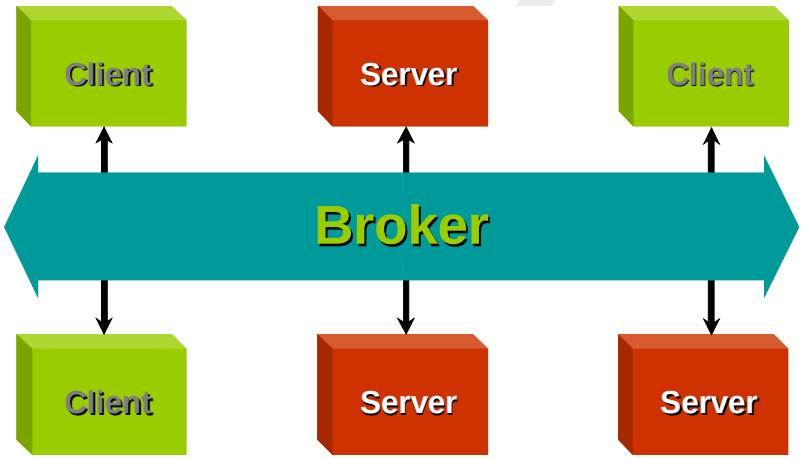

***Figure 2**: Brokering middleware is based on the use of an ``intelligent'' intermediary between clients and servers.*

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.

4.3 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. 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.

CORBA is designed with the following goals in mind:

- **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*. Objects reside in servers and are invoked by clients. Objects can be active or passive. An object can also simultaneously play the client and the server role.
- **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.
- **Programming language neutrality:** CORBA, in contrast to, e.g., Java RMI, is not dependent on any single programming language. Clients and servers can be implemented in a large number of different programming languages.
- **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.

4.3.1 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:

- **The Interface Definition Language (IDL)**. The Interface Definition Language is used to define the interface of a CORBA object. The interface is the syntax part of the contract that the server object offers to the clients that invoke it. Any client that wants to invoke an operation on the object must use this IDL interface to specify the operation it wants to perform, and to marshal the arguments that it sends. When the invocation reaches the target object, the same interface definition is used there to unmarshal the arguments so that the object can perform the requested operation with them. The interface definition is then used to marshal the results for the reply, and to unmarshal them when they reach their destination. The IDL interface definition is independent of programming language, but maps to all of the popular programming languages via OMG standards. OMG has standardized mappings from IDL to C, C++, Java, COBOL, Smalltalk,

Ada, Lisp, Python, and IDLscript. CORBA interfaces are strongly typed, support multiple inheritance, but does not allow overloading,

- **Object Request Broker (ORB)**. The ORB contains the necessary infrastructure that enables clients to invoke operations on CORBA objects. It typically contains client stub code and server skeleton code obtained when the IDLs are compiled, and linked with the application. It also contains mechanisms for locating and activating remote servers.
- **Object references**. Object references are the basic entity for encapsulating the type and location of a CORBA object. Object references are represented as runtime objects. They contain information about the interface type of the CORBA object and thereby also of all supported operations for that object. They also contain information about the location of the object. This typically includes the host address and port number of the relevant server and a server-specific object key. Object references can also be passed between clients and servers, e.g., as a part of a remote invocation. For this purpose the interoperable object reference (IOR) has been defined.
- **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. When an object adapter receives a request message, it identifies the target object implementation using the object key and invokes the corresponding operation on behalf of the client. The object implementation is known as a servant. The object adapter is also responsible for managing the life-cycle of the CORBA objects, e.g., object activation and deactivation. The Portable Object Adapter (POA) is the object adapter version used in the current CORBA standard.
- **Inter-ORB protocols**. CORBA Inter-ORB Protocols (IOP)s define interoperability between ORB end-systems. The IOP:s are normally mapped down onto reliable full-duplex connection-oriented transport protocols such as TCP that are implemented as octet (byte) streams. The General Inter-ORB Protocol (GIOP) is the basis for all IOP:s. GIOP defines three core elements: a Common Data Representation (CDR), message formats, and the transport assumptions described above. The CDR defines how data should be represented during transport (byte ordering, byte alignment, serialization ordering, etc). Eight different message formats are defined for handling communication between ORBs. These include request and reply messages. 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 supported three communication models: synchronous two-way communication where the client blocks until the

reply from the server is received (the most commonly used CORBA model), one-way communication without any reply implemented on top of TCP or UDP, and deferred synchronous communication in which the client is not blocked, but can itself chose to poll to see if the reply has been received or do a blocking wait for the reply. However, both the one-way communication and the deferred synchronous communication had certain drawbacks. For example, the deferred synchronous invocation mode could only be used if the request is invoked using the Dynamic Invocation Interface (DII), as opposed to the normal Static Invocation Interface (SII). To alleviate this, CORBA 3.0 introduced Messaging that supports asynchronous method invocation (AMI), time-independent invocation (TII), and messaging quality-of-service policies. Using AMI, operations can be invoked asynchronously using the static invocation interface. Two communication models are supported. In the *polling model* each asynchronous two-way communication returns a *Poller* object that the client can use to check whether the reply has arrived or not. In the *callback model* server responses are dispatched to special *ReplyHandler* objects. Time-independent invocations is a specialization of AMI that supports "store-and-forward" semantics, where requests may outlive clients and the response may be handled by a completely different client. The inclusion of AMI and TII into CORBA can be seen as OMG's response to the strong industrial interest for Message-Oriented Middleware (MOM). CORBA 3.0 also supports messaging quality-of-service (QoS) properties. All QoS properties are defined as interfaces. Applications can define QoS properties at multiple client- and server-side levels, e.g., the ORB level, thread level, and object reference level. The client-side policies give control over things like request and reply timeouts, priorities and ordering, routing semantics, etc. CORBA Messaging can partly be viewed as a replacement for the CORBA Event Service and Notification Service.

The CORBA architecture is summarized by Figure 3.

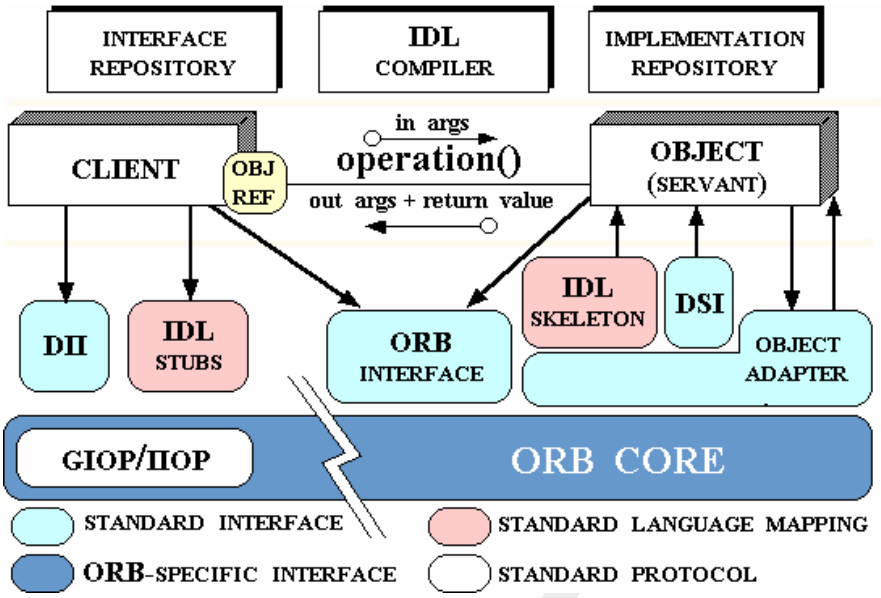

Figure 3: The CORBA architecture (from [Sch02]).

4.3.2 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.

- **CORBA Event Service.** The CORBA Event Service decouples the communication between objects. Two roles for objects are defined: the supplier role and the consumer role. Suppliers produce event data and consumers process event data. Events are sent asynchronously between suppliers and consumers. This is achieved using event channel objects that implements the Mediator pattern to implement asynchronous communication between multiple suppliers and multiple consumers. Two communication models are supported. In the push model it is the supplier of events that initiates the transfer of event data, whereas in the pull model it is the consumer that is taking the initiative.

- **CORBA Notification Service.** The Notification Service enhances the Event Service by providing QoS support, event filtering, structured events, and event subscription.

4.3.3 The CORBA Component Model

The traditional CORBA object model has several limitations when viewed as a component model. For example, it has no standard way to deploy object implementations, a fairly restricted interface model, no standard object life cycle management, the availability of CORBA Services cannot be guaranteed, and a very high degree of flexibility which requires the designer to do a large number of design choices and specify a large number details. To address the limitations the OMG in 1999 adopted the CORBA Component Model (CCM) to extend and subsume the CORBA object model. CCM is an distributed component model with close similarities to Enterprise Java Beans (EJB). The model contains an architecture for defining components and their interactions, a packaging technology for deploying binary multi-lingual executables, and a container framework for injecting lifecycle, (de)activation, security, transactions, persistence, and events. Components are created and managed by *homes*, execute in *containers* that manage system services, and are hosted by application component servers. They have several input and output interfaces, and support both synchronous and asynchronous operations. Components are defined using a number of extensions to CORBA IDL 2. For example IDL 3.0 is used to define component-oriented collaborations (component types, event types), OMG Persistent State Definition Language (PSDL) defines storage types and homes, and OMG Component Implementation Description Language (CIDL) defines compositions and segments. The CCM specification is quite large and complex. Therefore it has not yet been fully accepted among ORB providers. For these reasons, we will primarily focus on the CORBA object model rather than the component model in HRTC.

4.4 CORBA for Real-time and Embedded Systems

Apart from the importance of having a platform for integration and development of modularized controllers, there are some new issues in CORBA that are especially relevant for distributed control systems engineering. These issues are: predictable behavior, fault tolerance and execution in an embedded environment.

The Real-time platform task Force is addressing all these topics and focuses their activities on real-time systems, which often also are 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:

- **Real-time CORBA**: The Real-Time CORBA (RT-CORBA) specification (in addition to the Messaging specification) provides mechanisms for controlling resource usage to enhance application predictability.
- **Fault-tolerant CORBA**: The specification provides mechanisms for fault tolerance based on entity redundancy.
- **Minimum CORBA**: Minimum CORBA addresses the construction of CORBA applications on systems with scarce resources like embedded computers, where small memory footprint is important. This specification eliminates most dynamical interfaces that are not necessary in static applications (most embedded applications are ROM-ed applications).

4.4.1 Real-time CORBA

RT-CORBA 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 that 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 [Sch00]:

- **Priority type systems**. 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.
- **Priority Models**. Two priority models are defined. Using *server-declared priorities* it is the server that decides the priority at which an object invocation should execute on the server-side. The client is made aware of the priority at which the object invocation will execute through a tagged component in the object reference and can take use this information

internally. With *client-propagated priorities* it is the client that declares the invocation priorities which the server then must honor. The invocation priority is transferred to the server in the service context part of the GIOP request message.

- **Priority transforms**. A server is permitted to define priority transforms that sets 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. Each POA must be associated with one thread pool, but a thread pool can be associated with multiple POAs. Thread pools without lanes is created with a fixed number of statically allocated threads which the ORB uses for executing client invocations. To handle request bursts the number of threads is allowed to grow through the creation of dynamic threads. The thread pool can also be pre-configured for a maximum buffer size or number of requests. Using thread pools with lanes the threads in a thread pool are partitioned into subsets, each with different priorities or priority bands. Lanes with higher priorities are allowed to borrow threads from lower priority lane.
- **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. The service is provided in the form of a CORBA object that is responsible for allocating system requirements in order to meet the QoS requirements. 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. The scheduling service is, however, only an optional part of RT-CORBA 1.0.
- **Protocol properties**. An interface is defined that allows applications to specify ORB- and transport-specific protocol properties that control various communication protocol features. The protocol property structures reside in a protocol list that is part of the object references. The order in which the protocols appear indicates the order of preference in the case when parallel protocols are available. Servers can export protocol

preferences to clients through object references. Clients can use protocol policies to select which protocol to use when acquiring a binding to an object.

- **Explicit binding**. In standard CORBA connections (bindings) between a client and a server are established on-demand. The connections are normally persistent and it is allowed for an ORB to multiplex multiple invocations to the same server on the same connection. 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. Using *priority-banded connections* it is possible for clients to specify explicit priorities or priority bands for each connection or to select an appropriate connection at run-time based on the CORBA priority of the thread. Using *private connections* a connection may not be reused for other invocations until the reply for the previous request has been received.
- **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 communications 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. Work on a dynamic scheduling addition to RT-CORBA has been started.

4.4.2 Fault-tolerant CORBA

Fault-tolerant CORBA tries to enhance application fault tolerance reducing to a minimum the impact to the application (computing overheads and increase of complexity). Fault tolerance is increased by means of entity replication: cold passive replication, warm passive replication, active replication or active replication and majority voting.

4.4.3 Minimum CORBA

Embedded CORBA applications reduce memory footprint by means of elimination of some features (dynamic interfaces and repositories), the use of standardized operating system services or special transports. The elimination of a specific service from the specification does not mean that the application cannot use it, only that it will not be necessarily provided by a compliant CORBA implementation.

4.5 Bridging Domains

While the Minimum CORBA specification reduces the requirements posed to the ORB, the *Real-time CORBA* and *Fault Tolerant CORBA* specifications can increase the size and complexity of the application.

Thanks to interoperability, it is not necessary at all to have all the application running atop the same ORB. It is possible to have the critical part of an application running over a Real-time ORB and the rest over a more conventional one. It is possible to use a CORBA gateway to bridge between two different worlds in a control application.

4.6 State and Future of the Technology

CORBA technology is impressive but perhaps too impressive for normal control systems developers. It suffers what is called a second system effect, trying to address all possible functionality or requirement. We must identify our own needs and determine if the CORBA way fits our needs. If not, we are still in time to modify it.

Perhaps the main question is w*hy we need integration?* Beyond many obvious answers (to build complete plants, to achieve total safety, to be the first in the market, to spend less money, etc.), the authors would like to stress one door that this approach opens for us: 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 [Sha96]. 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 no-compromises frameworks, *i.e.* 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, designs; 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 [San99], and a transparent object-oriented real-time middleware.

4.7 What should Hard Real-time CORBA be?

CORBA and RT-CORBA contain a number of features that are useful also for hard real-time applications, 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/RT-CORBA is the transport protocol. The IIOP (GIOP over TCP) transport does not give any end-to-end timing guarantees. 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, e.g., the ATM transport protocol, or which are intended for communication within a node using, e.g., shared memory, Unix sockets, or VME-bus. An exception to this is the support for the unreliable, connection-free communication provided by UDP, which is provided by certain ORBs, in particular for multicast messages.

  In order for CORBA to be applicable to hard real-time applications it is necessary to support transport protocols with higher levels of determinism. The minimal requirement on a deterministic transport is an upper bound on the end-to-end latency. If the transport also can guarantee a lower bound on the latency the level of determinism increases. The less conservative (tighter) the bounds are the smaller will the jitter in the latency be.

- Periodic activities
  CORBA was originally based on a client-server communication model. In later CORBA versions support has also been added for message-passing. For real-time communication a sender-receiver model is more appropriate, where a sender periodically transmits messages to one or multiple receivers. Using the client-server model this would typically correspond to a client thread located in one node that periodically invokes an operation on a server object located in another node. IDL is focused on describing the interface of the CORBA objects residing on the server side. In order to support periodic real-time communication CORBA also 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. However, an activity is primarily used to describe a sequence of, possibly nested, operation invocations. Hence, it only concerns the client-side of the communication.

- Scheduling

A communication network is a shared resource. 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. Depending on the type of communication protocol that is used and the degree of determinism that is desired the scheduling requires different amount of prior information. The output of the scheduling is also dependent on the type of scheduling used. For example, the output of a static time-triggered scheduling approach would be the time slot allocation for the different nodes. In a worst-case scheduled switched Ethernet the output would instead be the maximum send rates for the different nodes.

The need for global information about distributed object invocations does not fit into CORBA very well. One approach is to simply say that the scheduling is something that is external to HRT-CORBA. In this approach one would then simply assume that all clients generate distributed invocations according to some pre-defined schedule. However, this would rule out all more dynamic scheduling approaches. In order to dynamically decide whether a new periodic communication request should be allowed or not (admission control) it is necessary to have some global source of the scheduling information. Where and how this information should be provided is an open issue. One possibility could be to have a special CORBA scheduling object that resides within some node and that is defined using IDL. Another possibility would be to introduce the scheduling support as a CORBA service.

RT-CORBA again mentions global scheduling. However, not in a way that fits the demands for scheduling of communication traffic. The RT-CORBA scheduling service is only assumed to apply to RT-CORBA activities. Also, the scheduling service is only an optional part of RT-CORBA 1.1. To our knowledge it has not been implemented in any commercial ORB.

- 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.

4.8 Competing Technologies

Two major advantages of CORBA are the language independency and the platform and vendor independency. The two main competitors to CORBA, Java technology and Microsoft technology do not share these advantages. However, the strong position that Java has within Web-computing and the strong position that Microsoft has within industrial automation in general, make the competition fierce.

The Microsoft based technologies can be divided into two groups: pre-.NET and .NET technologies. The pre-.NET technologies include COM, Microsoft's component framework, DCOM, the distributed version of COM, and MTS which adds persistency and transaction services. Together they constitute COM+. COM relies on binary interoperability conventions and on interfaces. A COM interface can be seen as a C++ virtual class. COM provides a simple protocol that COM objects can use to dynamically discover or create objects and interfaces. From an implementation point of view a COM object is a piece of binary code that can be packaged in executables or DLL dynamic libraries. DCOM extends COM with distribution based on the DCE Remote Procedure Call (RPC) mechanism.

The .NET component model relies on language interoperability and introspection rather than binary interoperability. In order to enable this .NET is based on an internal byte-code language called Microsoft Intermediate Language (MSIL), very similar to Java byte-code. The interpreter for this language is called the Common Language Runtime (CLR), which is very similar to the Java virtual machine. A number of languages can be compiled to MSIL. .NET represents the programming language approach to component programming. The program contains the information related to the relationships with other components, and the compiler is responsible for generating the information needed at execution.

Microsoft's drastic change of technology has partly upset the automation industry. Traditionally they have been using COM technology, which no longer is the main approach pursued by Microsoft. However, the interface and bridge support between .NET and COM is good.

Another Microsoft technology of importance to the automation industry is OPC (OLE for Process Control). OPC is designed to bridge Windows-based applications and process control hardware. OPC is based on OLE (Object Linking and Embedding), a part of Active-X that provides object plug-and-play functionality within Windows, and on COM/DCOM.

OPC software are either OPC-clients or OPC-servers. An OPC-client is typically a data-sink, e.g., a GUI or SCADA system that needs on-line process data. An OPC-server is a data source – a device-specific program that collects process data from a field device and then makes it available to an OPC-client. Due to the relatively large memory footprint of OPC-servers they are rarely embedded in the field devices per se. Instead the OPC-server is typically part of the control system, either in the control stations or at some supervisory level. Used in this way the OPC-server encapsulates the underlying process and control data and makes it available to clients. OPC provides a number of different services, e.g., online data access, alarm and event handling, and historical data access. The main difference between OPC and CORBA is that OPC is based on signals whereas CORBA is object-oriented. However, part of the functionality that hard RT-CORBA will provide will clearly overlap with OPC. One possibility for handling this problem would be to either encapsulate an OPC-server as a CORBA object or to provide a completely CORBA-based implementation of OPC.

The Java environment provides its own solutions both to distributed object computing and to component technologies. The Java distributed object model is Java RMI (Remote Method Invocation). Java RMI shares many of the features of CORBA and during recent years Sun has strived to unite them even further. For example, RMI over IIOP enables the programmer to develop CORBA compliant distributed Java applications using the RMI framework. To develop CORBA applications in other languages IDL models can be automatically generated from Java programming language interfaces. RMI over IIOP includes the full functionality of a CORBA ORB and is a part of both the Java Standard Edition and the Java Enterprise Edition. Java also directly supports CORBA through Java IDL. Using Java IDL the distributed object interfaces are programmed directly in IDL, rather than in RMI.

The Java component technology is known as Enterprise Java Beans (EJB). The EJB server-side component model simplifies development of middleware components that are transactional, scalable, and portable. EJB servers reduce the complexity of developing middleware by providing automatic support for middleware services such as transactions, security, database connectivity, and more. EJBs use the RMI/IDL CORBA subset for their distributed object model, and use the Java Transaction Service (JTS) for their distributed transaction model. When Enterprise JavaBeans are implemented using the RMI-IIOP protocol for EJB interoperability in heterogeneous server environments, the standard mapping of the EJB architecture to CORBA enables the following interoperability:

- A client using an ORB from one vendor can access enterprise beans residing on an EJB server provided by another vendor.

- Enterprise beans in one EJB server can access enterprise beans in another EJB server.
- A non-Java platform CORBA client can access any enterprise bean object.

5 Components in Control Systems

5.1 Introduction

The primary market drivers in the automation industry today are increased productivity and flexibility, increased quality and yield, reduced life-cycle costs, safeguarding investments, and environmental and safety management. A key goal is to provide vertical integration, i.e. to make real-time information available across the enterprise, allowing users and applications to make informed production decisions. The real-time information also needs to be shared between multiple automation platforms, including control systems, transmission networks, PLCs, safety systems, SCADA systems and maintenance systems. The goal which all process and manufacturing enterprises are striving for today is a seamless integration of plant and enterprise systems. In order to achieve this it is necessary to build the system on a strong architectural foundation. A common distributed object and component model is a key element of this.

5.2 Industrial Control Systems

Modern industrial control systems of the DCS (Distributed Control System) or PLC (Programmable Logic Controller) type are complex, distributed, heterogeneous systems. The majority of the complexity is due to software issues rather than hardware issues. The systems are hierarchically organized and are often depicted in the form of a triangle or pyramid, see Figure 5. Traditionally the information flow between the layers has been small, compared to the information flow within the layers. However, the trend towards vertical integration in process control systems is beginning to change this. The refinement level of the information is also different between the layers. At the layers close to the process the information primarily consists of signal data, e.g., measurement signals and control signals. Higher up in the hierarchy the information content is of the same nature as what is found in general enterprise systems. In a similar way the real-time characteristics are different between the layers. Close to the process the

information is updated periodically with high frequency and the real-time requirements are of hard or "semi"-hard nature, whereas at higher layers the traffic is of a more aperiodic nature with softer real-time requirements.

An example of the network architecture of modern control systems is found in Figure 4. The figure shows ABB's new Control IT system. The solutions provided by the other four largest control companies (Emerson, Honeywell, Invensys and Yokogawa) are presented in deliverable D4.1. Control IT is the controller part of ABB's new Industrial IT concept. The system contains several communication networks. At the bottom level there is one or several fieldbuses for connecting remote field devices (e.g., sensors and actuators) with the control system. Examples of fieldbuses that currently are wide-spread are Foundation Fieldbus, Profibus, LonWorks, and HART. The majority of the communication concerned with networked control loops takes place at the fieldbus level.

The control network is used to connect the individual controllers with each others, with the Connectivity Server and with the Control Builder Server. The Connectivity Server is an OPC server that publishes the control system information to higher level applications. The Control Builder Server is responsible for the downloading of the control application code to the individual controllers. Although it is possible to close control loops over the control network it is not so common. The control networks have traditionally been based on vendor-specific communication protocols, but this is gradually changing towards the use of more standard protocols. The Control IT system uses the ISO Manufacturing Message Specification (MMS) application protocol layered on top of the TCP stack. MMS has been defined by experts from process control and manufacturing. It allows definition of abstract virtual devices in terms of data and services. Originally MMS was developed to run on top of the MAP communication stack. MAP support interoperability between heterogeneous shop-floor devices, PLCs, robots, sensors, and actuators. However, despite large investments MAP never became a real success.

For security and efficiency reasons the control network is split up into two parts, the part that connects the controllers and the part that connects the different user workplaces, e.g., Operator Stations or Engineering Stations, and the application servers. The latter network is typically a standard TCP/IP client-server network. The control network is then connected through a bridge to the plant intranet, which in turn is connected to the ordinary Internet.

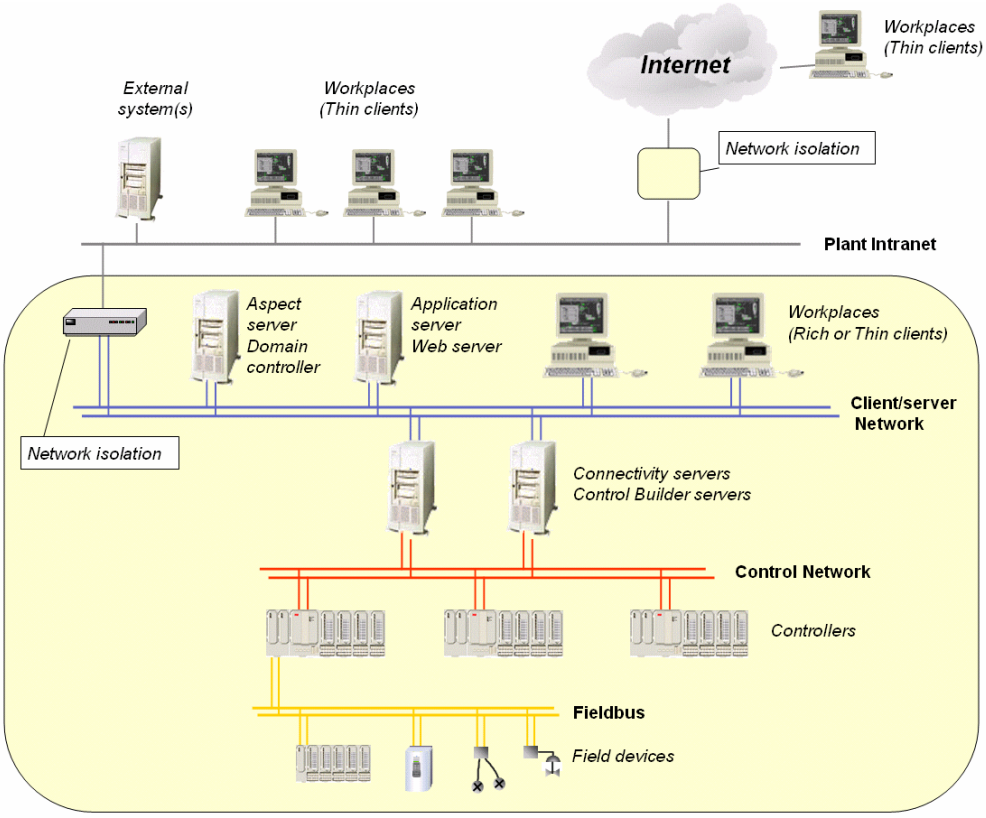

Figure 4: . The network architecture of the ABB Control IT system.

Object models are important architectural cornerstones in modern automation systems. An example of this can again be collected from ABB. The ABB Industrial IT concept is based on the Aspect Object model, a multi-view object model where each object is described from several different views or *aspects*. The objects themselves can be physical process objects such as valves or motors, or they can represent products, materials, orders, et cetera. Aspects can include human machine interface, configuration, simulation, quality report, production reports, maintenance records, electrical diagram, control diagram, and many more. The aim of the Aspect Object model is to model one aspect of an object at a time, rather than to create a complete single object. The Aspect object can be seen as a container of references to different "ordinary" objects that implement the individual aspects. The aspects are implemented by different software systems, so called *aspect system*, which store, manage, and present information in an aspect-specific way. The result is a system of loosely integrated independent software systems held together by a special aspect directory service. Aspect objects allow reuse of knowledge and components and facilitate one-time entry of information. The Aspect Object model implementation is based on COM. The Aspect Object model is quite advanced and contains several features which

cannot be found in conventional software object models, such as CORBA and COM.

5.3 General characteristics of control systems

There are two fundamental characteristics that a control system must have to be able to perform effectively:

- Dependability,
- Maintainability

Other required or typical characteristics are:

- Scalability
- Configurability

Finally, its distributed architecture is considered.

5.3.1 Dependability

A system is a dependable if it is operative when it is needed. The major factors that contribute to dependability are availability, security, and ease of use.

- ***Availability*** is the proportion of time the system performs to specifications. Superior system availability is provided by the following factors:
  - *Integrity*. In addition to the architecture, integrity is achieved by self-checking failure detection, fault containment, and the inherent safety designed into the system to ensure correct operation. Errors that do persist should be tolerated with little or no degradation of system performance, affecting only the device or module in which they occur. If one of these devices fails, some capacity, throughput, or functions may be lost, but the system continues to operate.
  - *Redundancy*. Multiple elements can produce correct output when one or more of the elements is not functioning correctly. The most critical system elements are usually fully redundant with automatic switchover.
  - *Maintenance*. It is the activity that keeps the equipment in satisfactory working order, including tests, diagnoses, measurements, replacements, adjustments, and repairs.
- *Security*. If unqualified users of a system are able to make improper changes to the process or control information stored in the system, it

will reduce the user's level of confidence that the available information is correct. Furthermore, improper changes might cause damage to the process or plant, and injury to people. To keep this from happening, the different levels of access have to be provided.

- *Ease of use*. Time spent in finding how to do something with the system or in retrying an operating procedure is unproductive.

5.3.2 Maintainability

Maintainability is achieved by:

- Standardization of hardware and software.
- Automatic diagnostics where the system records and analyzes both hardware and software errors and makes recommendations to replace devices it suspects of impending failure.
- Off-line tests when self-diagnostics cannot pinpoint a problem.
- Hot-replaceability For maintenance or repairs, any module can be removed from service and returned to service while the remainder of the system is on-line.

5.3.3 Scalability

Another key design objective for automation systems is scalability. The same automation system family should support the full range of applications from small PLC-type applications involving, e.g. only a single standalone PC to large enterprise-wide control systems with replica-determinism to achieve fault-tolerance.

5.3.4 Configurability

A characteristic feature that distinguishes industrial control systems from most real-time software systems is the fact that the systems are end-user programmable. From a software point of view the systems consists of two layers: the system level and the application level. The system level provides configuration support, run-time support, and a communication infrastructure. The control applications are typically programmed by the customer using a domain-specific programming language. Many times, the user does not write custom software, but enters the information that the standard software uses. This entry process is known as "system configuration".

The domain-specific languages are often graphical in nature and are based on quite simple computational models, e.g., signal or activity flows. The most common example are the five languages in the IEC 61131-3 standard. Although these languages are not object-oriented in the strict sense, they contain structuring elements that can be compared to objects. There is also a strong interest in extending these languages with more powerful object constructs. The fact that the system contains two separate layers further increases the potential for using component technology in their programming.

5.3.5 Distributed architecture

Data acquisition and control functions are distributed throughout the plant, using a variety of process-connected devices to meet a plant's needs, as it was shown in the ABB's Control IT example in the previous section.

In other industrial control systems, similar four network levels are encountered:

1. Information network. It is a network that is capable of sharing information with the operation network.
2. Operation network. It links operator stations, processing modules, and gateways/interface modules. New systems implement an Ethernet TCP/IP network.

   On this networks are the modules:
   a. Human machine interfaces (operator stations, etc).
   b. History database
   c. Advanced control (MBPC, statistical, etc.)
   d. Gateways and interfaces to:
      i. Control networks
      ii. Other process control subsystems like PLCs
      iii. Information networks
      iv. Other operation networks
   e. Safety instrumented systems controllers

3. Control network. It links process-connected devices with each other and with the operation network (through the gateway). The devices connected are process controllers.
4. Fieldbus. Fieldbus is a *digital*, *two-way*, *multi-drop* communication link among intelligent control devices. The main benefits are:

   a. Higher accuracy and data reliability
   b. Multi variable access
   c. Remote configuration and diagnostics

d. Wiring cost savings
e. Reduce commissioning time (due to the diagnostic and configuration information available)
f. Benefits of moving control functions to the field devices improves the control.

It links process controllers (through an I/O unit) with instruments. There are other I/O units in the process control modules: analog, digital, serial, etc.

The communication advances is what is making possible the use of smart sensors and. There has been an evolution from the traditional analog transport (where a 4-20mA was transmitted indicating the percentage of the process variable measured or the percentage to act on the valve) to the current digital fieldbuses where different "standard" protocols are competing to rule in the field level.

In between there is an hybrid solution, the HART (Highway Addressable Remote Transducer) protocol which superimpose a digital signal to the classical analog one. This enables the use of the existing devices and take the advantages of digital communication. Although this has been widely used in the process industry it seems that the digital fieldbuses will take over finally in the coming years.

Some control strategies at high levels require calculations that may require, for example extensive file handling or computation (optimization, simulation) that generally do not need to be synchronized with control algorithm execution. In addition, collection, storage, and manipulation of exceptionally large quantities of historical data may be required. Via the information network such computer programs can access and write information anywhere in the system. This makes it feasible to establish a system-wide information network to make timely data available at all levels of decision-making within the organization.

The gateway between the operation and control network passes event and alarm information from the control network to the other and responds to requests from the modules in the operation network for information about the process. It also passes configuration files to the control network modules. On the control network, the gateway validates existence of requested points, checks for device status, and checks for device and parameter error conditions.

As it was said above, control systems implement redundancy in networks and equipment. Regarding the network, either of the two redundant cables can be designated as the active one; the other is then the backup. If the active cable fails or has an excessive error rate, the roles of the cables are automatically switched (the cables can also be manually switched). Additional security can be achieved by running each cable over a different route.

The HRTC project is focused on distributed hard real-time CORBA applications. It is therefore of interest to investigate the types of network communication that takes place in a typical control system and their real-time characteristics. Some examples are:

- Sensor data and control signals
  Sensor data that is being sent from remote field devices to the controllers and control signals that are sent in the opposite direction are perhaps the most important network traffic. The real-time characteristics depend on the control application and the type of control being performed. For example, in many cases a discrete logic controller has harder real-time characteristics than a continuous control loops. In the latter case it often acceptable with an occasionally long delay or lost sample. Raw sensor data or filtered sensor data is also transmitted to the supervisory layers, e.g., to the HMI and to different supervisory control applications. In general, this type of traffic has less hard real-time constraints. The data transfer is in many cases based on OPC.
- Events and Commands
  Events or alarms are typically generated from the controllers when, e.g. some abnormal situation has been detected that require operator invention. Hence, the real-time requirements are typically quite high. Commands correspond to discrete operations that are performed from the supervisory control levels, including the HMI, and that affect the operation of the controllers. Also, here the severity of the real-time constraints is application-dependent.
- Binary Code
  Binary code is typically downloaded from the configuration and engineering workstations to the controllers when the control system is installed or updated. The possibility to perform the updates on-line (hot-swap) is important. This implies that the real-time requirements can be quite high also for this type of traffic.
- Internet traffic
  The use of web server techniques is becoming increasingly common also at the lower layers of the control system hierarchy. A typical application is equipment diagnosis. The traffic is typical of traditional HTTP nature.

5.4 Common functions in industrial control systems

5.4.1 Data acquisition

In a typical scheme, the control system acquires data from process connected devices distributed on networks. Each device scans its associated process instrument(s) at regular intervals, checks the input signals, and converts them to a form suitable for storage in its own process database.

All of the information about the process that the system collects or produces must be structured in some way for easy retrieval, e.g., a collection of closely related data values (such as all parameters associated with a control loop) and instructions for their processing. At the scheduled time, standard software within each kind of device or module automatically collects process variables, stores them, performs any calculations or other manipulations, and sends outputs to designated locations.

5.4.2 Alarms

Alarm states are detected by comparing values against limits, ranges, or other conditions. Changes in the state of alarms are events to other modules. There are alarms for process variables and for deviations of process variables from setpoints. Alarms can be in one of several detection modes ranging from the no-action level, which totally ignores the occurrence of the alarm (used, e.g., to filter out nuisance alarms during startup and shutdown), to the emergency level, which immediately brings the alarm to the attention of the operator in several ways: audible signal, red text on the screen, flashing text, flashing lights, printed messages, etc. Alarms have to be acknowledged by the operator.

5.4.3 Control

The basic control functions of control systems can be classified in two types:

- Continuous
- Discontinuous

For every unit, a set of continuous control functions (loops) are defined. Also, sequences are defined. Each sequence is divided into phases containing several steps (statements of the programming language). As these sequential statements control the sequence of operations being performed, they also initiate any continuous or logic operations required, such as setting setpoints or checking that a valve is closed before admitting fluid into a tank. These sequences can run in different modes, automatically or under the operator intervention. The engineer can specify whether the operator or the system can change the mode and the system has provision for automatically performs all the functions

necessary to prevent "bumps" in the process, including initialization, ramping, and antiwindup whenever a mode is changed.

5.4.4 Safety functions

In response to abnormal conditions, usually detected using logic functions, safety sequences can be triggered that stops the normal execution order and proceeds with the instructions the engineer gave for handling the particular situation. Usual responses are:

- Hold all variables at their last good value until otherwise instructed
- Execute the given normal shutdown procedures
- Execute the given emergency procedures to get the process to the safest possible state in the fastest possible manner (emergency shutdown).

The most critical safety functions are performed by dedicated equipment: The Safety Instrumented System (SIS), which are based on multiple redundant PLCs and instruments.

5.4.5 Communication with other systems

Control systems are frequently composed of a principal system and a wide and complex variety of independent subsystems, both in process (laboratory, package units, etc) or not (management information system, shipping, etc.), that need to work together to accomplish operating objectives. To meet this requirement for integration of products from other vendors, it is necessary to provide a number of interfaces at the different network levels with capabilities such as data conversion, buffering, and processing necessary for an efficient interchange of information and smooth startup and shutdown.

5.4.6 Reports

Control systems have the capability for producing reports with current and historical data about the process and the system, which can be printed (on demand or at specified intervals) or displayed. Reports can be classified into:

- *Journals*, that collect a chronological list of events of a specified class, such as process alarms or operator changes to the process, that occurred within a specific time interval.
- *Logs*, that collect historical values for a specified set of data point parameters.

- *Trends*, that graphically show the history of points or parameters over a specified time interval.

A report can include a mixture of the three types.

5.5 Elements of control systems

Any control have the following elements : sensor, actuator, controller and the communication media between them all. A brief summary of the trends in the elements and the communication media is presented in this section. Besides the basic elements, there are other significant components in any control system: Human Machine Interfaces and Databases.

5.5.1 Sensors

- The trend is the smart sensor. This smart digital devices offer new functionalities such as:Transmission of many data as: operating range, maintenance conditions
- Remote operation by the user (change the span, software update,…).
- Simpler communication. The digital communication removes the need of digital/analog and analog/digital converters.
- Easy integration in the DCS configuration

But these capabilities are not cost-free, an increase of complexity (with more possibilities of failure, although more means to detect it) is the major drawback when implementing the smart sensors.

5.5.2 Actuators

These devices are mostly valves in the process industry. The trend is the same as it was in sensors, having intelligent actuators. Besides the intelligent features there is an additional capability, what is being done is to transfer some control functions (basic control, and basic algorithms-PID) from the control room to the field. The new valves incorporate control blocks making the control more distributed.

5.5.3 Controllers

Process controllers handle data acquisition and control functions. They can be configured with a selectable set of I/O units (which perform input and output

processing on field I/O, independent of control processing functions), and control units with algorithms for continuous, batch, or hybrid applications that scan and write I/O units on the control module. Capabilities include sophisticated regulatory control, fully integrated interlock logic functions, and user programmable functions. With special units it can perform sequence of events functions providing typical 1 ms resolution time stamping of digital state changes.

A wide range of signal types can be handled (even communication with PLC), including redundant pairs to maximize availability for critical applications. Redundancy can be extended to controllers units and power supply.

5.5.4 Human Machine Interfaces

Operators view and monitor the process through standard operating displays that need to present a wide range of detail. In operator stations the operator monitors and controls the process, and handles process and system alarms. He can also display and print process history, trends, and averages, print reports and monitor and change the status of the control system. Process engineers set up the process database, displays and reports, establish the interfaces with other computers, and manages system software.

Levels of detail range from operating parameters and limits for individual points, to summaries of operating conditions (such as alarms) for individual process units (both continuous and discontinuous). Operators in different stations have to securely share data. The refresh frequency in displays ranges from seconds to fractions of seconds.

A minimum of two operator stations is recommended, because this provides ongoing operations capability if one station is needed for process engineering or system maintenance, or if one should fail. Three stations are preferred: The operator typically uses one station for an overview of the process area, another for a more detailed view of a unit or part of a unit in the area, and the third for an alarm summary display. If one of the stations is being used for process engineering or maintenance, the other two can take over its functions.

5.5.5 History database

Control systems include a database that records historical data as specified of the process values used by several departments of the plant, at several levels. The database should be configurable (defining the history to be collected) in different ways to accommodate those needs.

Apart form mass storage of process data by time and event (process alarms, changes made by an operator to the process or the system, etc.) the history database usually stores also system program images, the entire system database as configured by the process engineer (reducing the time to restore application software of the modules), diagnostic information, custom graphic displays and user programs.

5.6 The Control System Landscape

Increased globalization and a consolidation of the automation market through mergers by several major control system manufacturers, highlights the necessity of component based frameworks.

The current, more traditional approach to software architecture, with one monolithic structure, becomes expensive to maintain, port, upgrade and customize. Significant portions of today's automation systems are becoming either functionally inadequate or logistically insupportable.

According to [Lüd01], one major problem is that of adding new I/O-modules, communication interfaces and protocols. Adapting the software to new products from a growing number of hardware manufacturers requires modifications and extensions to this monolithic structure.

Examples of new features that are demanded by the market in ever shorter cycles, presented in [Mül02], are

- Local and remote human/machine interface
- Automation processes
- Remote control options via a fieldbus

Managing this within one single application is a error prone and expensive task, and the resulting piece of software is growing both in terms of size and complexity. At the same time, only a very small percentage of the code is related to control and most effort is put on just "make it run" and not on critical control issues.

In [Gre99] three crucial factors of industrial efficiency are pointed out:

- Easy and smooth co-operation of various, possibly embedded software systems along the production course.
- Re-engineering tools of legacy industrial systems, improving information flows and control systems, while keeping basic heavy machining and manufacturing equipment.
- Distance co-operation between development and manufacturing sites.

5.6.1 The Role of Components

The use of a proper component model allows systems to be configured dynamically from binary components. A system would then be configured by loading *only* the necessary modules at a point in time when both the target hardware and software environment are known. The setup would resemble the way third party hardware is managed in the PC world through the use of device drivers. Of course, in an automation setting we need more than that, we need guarantees. It is never acceptable that the introduction of new hardware or software may cause unpredictable behavior or even system crashes. In automation applications we need to be certain that the deployed components will work together as intended. An application, designed as a composite of components, must be safe and predictable with respect to both its functional and temporal behavior.

A component framework would improve code reuse and hence shorten development time. In the current situation where software is becoming a bigger and bigger part of development cost for automation systems, this would be a major win. The use of a component model would also have the effect of standardizing the way functions are implemented, and the introduction of well defined interfaces would enhance understanding of how functions are used.

Besides creating a superior foundation for maintenance and upgrades, maybe a more important advantage of a component based framework is that it provides an increased level of abstraction, resulting in better support for design and implementation of complex automation applications. A distributed component technology will allow the programmer to postpone decision regarding the final application and instead focus on the design of the individual parts. The final application is then designed as an assembly of component, configured for a particular target. Good support for design of distributed systems design, greatly relieves the programmer from concerns about networking and communication, and simplifies implementation of heterogeneous systems. For example, seamless integration between high-performance servers, which are used to calculate advanced robot trajectories and low-level PLCs, which handle the high-frequency feedback loops.

Furthermore, integrating different parts of a factory is also of great importance. A six layer model of the factory levels is found in Fig.5. The suggested layers are:

- Enterprise management level
  Enterprise-wide control activities such as resource planning, supply chain management, financing and accounting.
- Production or manufacturing level
  Management and administration of work batches

- Process control level
  Plant-wide remote supervision and control of the plant using operator stations and processing systems.
- Group control level
  A set of control loops controlling a set of process level systems, e.g. the feedback loops belonging to a particular manufacturing cell.
- Field level or single control level
  This is the level of sensors, actuators, drives, etc, i.e. the interface equipment of a control system to the physical processes.
- Process level
  This is the controlled physical plant.

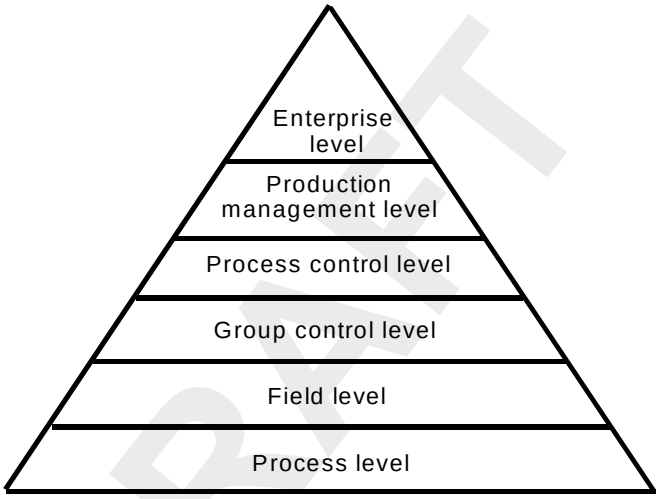

Figure 5: The six-layer model of an industrial control system [Pre02].

The introduction of a distributed component model such a CORBA would provide a software backplane allowing both horizontal and vertical integration of the layers on Fig.5. Horizontally, a standardized component would increase interoperability between modules from different manufacturers, while vertically the integration would allow business systems and administrative systems to directly access and query the control loops and, for example, monitor the progress of a particular assembly line.

Middleware like CORBA provides high level object oriented communication mechanisms, transparency and interoperability, which relieves the system designer of making decisions about hardware architectures and operating systems. When applied to control systems, CORBA technology provides a simple mechanism for sub-system independence, by means of transparent support for active objects and multi-threaded server construction. CORBA greatly simplifies

evolutionary changes to a system since adding new components may be added without any necessary changes to the rest of the system. Adding software fault tolerance by means of replication (active or passive) is much better supported in a component based framework in comparison to a monolithic program.

5.6.2 The Challenges

Supervision and control systems such as plant control systems, traffic management systems, energy distribution systems, are typical examples of application domains to be built on top of a CORBA platform. However, they often exhibit real-time requirements which the platform must comply to. The adoption of the current component models, such a CORBA or DCOM to a time critical setting, possibility in an embedded environment where system resources, such as CPU, memory and power, are scarce is a non-trivial task. While using a standard distributed component model in this type of applications certainly was not an option at the time when CORBA was initially designed, today it is thanks to increasingly powerful hardware for less and less cost. A CORBA component interface specifies only the functional behavior, and this does not suffice for real-time applications. For an embedded control systems application it is necessary to support analysis of the timely behavior of the components (and the composite). The interface must specify execution times, sampling times, memory consumptions, synchronization needs, etc. It also must support the ability to query a component about its resource requirements.

6 Some Application Examples

In the following a number of examples of how component technology in general and CORBA in particular has been applied or could be applied in industrial control systems are presented.

6.1 Networked Control Loop

In this example CORBA is used in the sensor nodes, controller node, and actuator node of a networked control loop. Figure 6 shows a networked control system, where the process samples are transmitted to a controller unit, which calculates a control signal and transmits it to the actuator node. This example will be further discussed in Chapter 4.

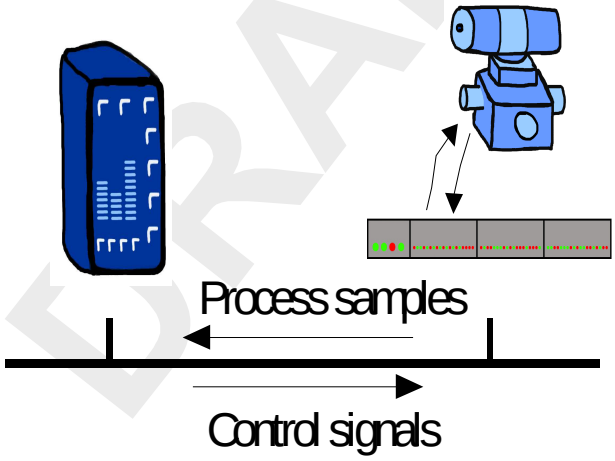

Figure 6: A networked control loop.

6.2 Distributed Supervisory Control Loops

In a distributed supervisory system the actual feedback control loop resides locally in the node, while reference trajectories or mode commands are transmitted over the network. This setup commonly, however not always, requires support for hard real-time. A stable system may behave dangerously if

fed an incorrect trajectory. However, given some logic in the local node, this setup may become much less dependent on the network and may function autonomously in a stand alone mode in case of communication failure. This setup is shown in Fig.7, where one supervision node is serving two control loops with reference values.

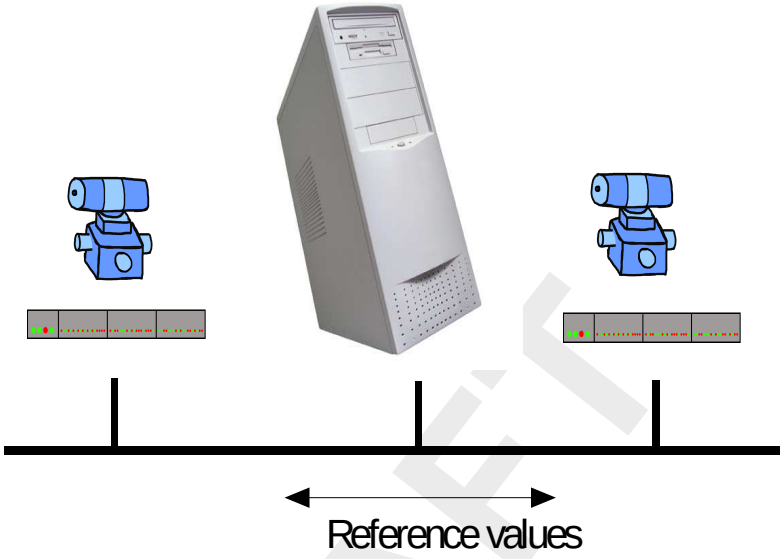

Figure 7: Spervisory computer transmits reference values out to one or more control loops. In this figure there are two low-level feedback control loops managed by the supervisory node

6.3 CORBA-based MMS

In [Gre99] the COCA project is described where CORBA is used to implement the ISO MMS protocol in a distributed object-oriented manner. The approach is based on a mapping of MMS's virtual manufacturing devices (VMD) onto CORBA objects. Similarly, MMS services are represented as object methods. The asynchronous services in MMS are replaced by synchronous CORBA method invocations. The advantages of the CORBA-based approach are openness and modularity. In order to reduce the risk of decreased performance due to the added software layers a very efficient ORB implementation is used. For example, remote invocations between objects across Ethernet use the medium directly and avoid the TCP/IP stack.

6.4 Componentization of I/O and Communication

One of the largest component-technology applications inside ABB Automation, apart from the Aspect Object model, concerns the use components for modularisation of the software related to IO and communication in Control IT

[Lüd02]. The ABB Control Builder is used to specify the hardware configuration of a control system, comprising one or more ABB Controller, and to write the IEC 61131-3 programs that will execute on the controllers. When the control application is downloaded to the control systems via the control network, the system software of the controllers is responsible for interpreting the configuration information and for scheduling and executing the control programs.

For market reasons the control system must support a wide range of I/O systems, communication interfaces and communication protocols. In the existing software architecture the addition of a new I/O module or a new communication interface or protocol would require updates in a large number of software modules, or components. To simplify the support for addition of new I/O and communication it was decided to restructure the systems by splitting all the components in two parts: one generic part containing code that is shared by all hardware and protocols and one non-generic part containing the code that is special to a particular hardware or protocol. The latter parts are called protocol handlers and they reside on the PC running the Control Builder and/or in the controllers. Using the new architecture the addition of a new IO or communication only requires the addition of the necessary protocol handlers. ABB's control system is based on COM and the COM Interface Description Language (IDL) is used for defining the component interfaces.

6.5 Factory Integration Frameworks

In [Yan99] a factory integration framework is described where CORBA technology is used for the development of manufacturing applications, including manufacturing execution systems and machine control. The framework focuses on three levels: factory level, cell level, and equipment level. At the equipment level the framework uses OPC. The framework uses three different communication models: synchronous two-way communication, asynchronous messaging using Java Messaging, and event-driven interaction through the Event Service.

Component-based approaches in flexible manufacturing systems are also described in [Mor02].

6.6 CORBA-enabled PLC

The development of an ORB for a PLC is described in [Kus98]. The ORB executes on a PC that is connected to a standard Melsec PLC. CORBA objects have been defined that represent the components and functions of the PLC. These PLC objects encapsulate the corresponding functionality of the PLC.

6.7 Component-oriented reference architectures

A number of reference architectures for open, inter-operable control systems have been proposed. Most of them take a component-based approach. Some examples are OSACA (Open Systems Architecture for Controls within Automation Systems) [OSA96], OMAC (Open Modular Architecture Controllers) [OMA98], and the Open Control Architecture for Windows NT [OCF98].

The OROCOS (Open Robot Control Software) project is an ongoing European open source software project aiming at creating highly configurable control components, based on CORBA principles, for robot control systems. However, CORBA is used mainly for the IDL and component interfaces, while distributed communication and RT properties of the run-time system is left out of the IDL. Instead, timing considerations are managed by supporting mechanisms in the control system architecture, but without HRT communication and execution.

6.8 Control Block Components

A trend in both industrial automation and robotics is the increased use of model-based analysis and simulation using tools such as Matlab/Simulink. In addition to using this technology off-line during design it is also interesting to be able to use it on-line, e.g. as a part of a model-based control scheme. In Simulink controller functionality is encapsulated in blocks with well-defined signal interfaces. If this should be used on-line there are currently two options. One option is to use existing code-generation tools for generating on-line code. However, it is sometimes quite difficult to fit this automatically code into the existing software structure. The second option is to include Matlab/Simulink in the on-line feedback loop. Software licensing costs often prohibit this solution. An interesting possibility would be to encapsulate the control functionality in a component that could be used both during off-line design and on-line operation.

A similar situation can be found within the IEC 61131-3 standard. Here, software blocks are available both in term of function blocks and 61131-3 programs. Although it is possible to write user-defined function blocks in C, it is not straightforward to integrate legacy software into 61131-3 application. A component-based approach where it would be possible to include components as, e.g. function blocks would be a very interesting approach.

6.9 Robot Tele-operation

As a laboratory experiment CORBA has been used to build a robot tele-operation application (see Figure 8) at UPM. The application contains three CORBA objects: a six DoF (degrees of freedom) full force feedback master, a seven DoF robot

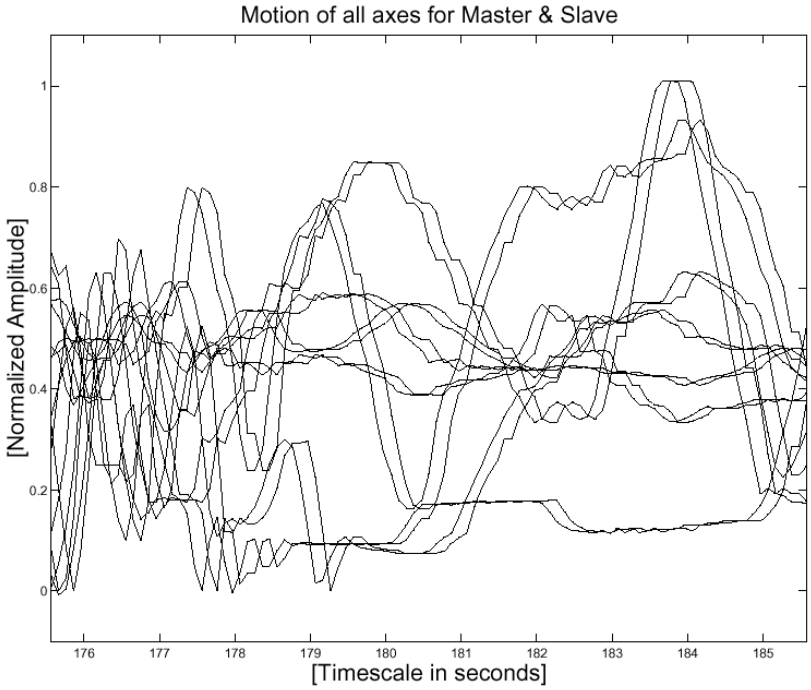

Figure 8: position evolution in master and robot during a test.

slave and a coordinate space mapper (transforms master axis space into robot axis space).

A large delay is obtained, approx. 250 ms, but with a small jitter. The test was done using the common laboratory 10baseT network in normal state (about 20% load).

6.10 Risk Management

Another application of interest is RiskMan. This is a system for emergency management in a chemical complex composed by nine plants. The system

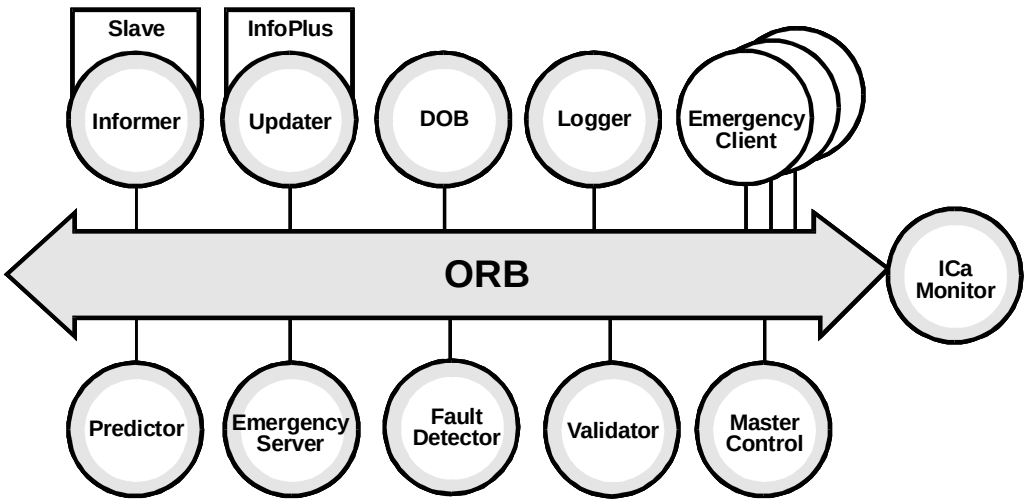

Figure 9: Some of the CORBA objects that compose the RiskMan application. Informer and Updater are wrappers of external systems.

supports the whole life-cycle of emergency management: prevention, detection, firing, diagnosis, handling, follow-up & 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.

All these components are built as coarse-grained CORBA objects (See Figure) that provide a concrete set of services to others. Global functionality is obtained form the autonomous cooperation of these agents.

6.11 Real-time Video for Tele-operation

HydraVision is a real-time video system for the support of remote operation of hydraulic power plants. It uses a country-wide fiber optics WAN network of a

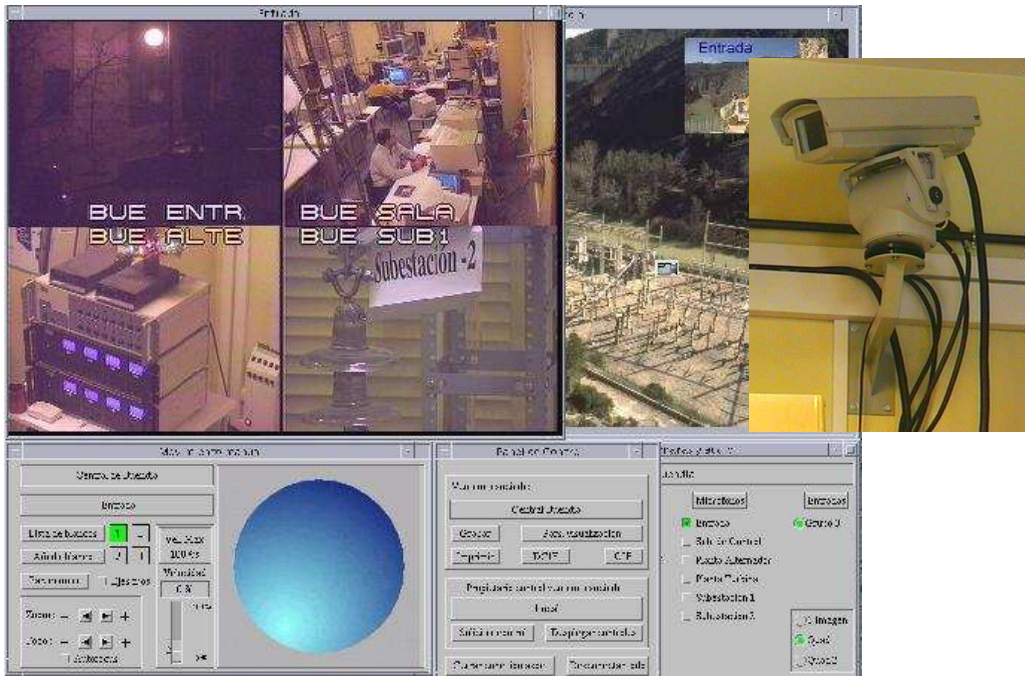

Figure 10: The HydraVision main user interface and one of the object-wrapped cameras.

electric company to integrate a collection of objects that wrap physical entities in the system (see Figure 10).

The physical systems that are wrapped as CORBA objects are: cameras, MPEG compressors, image/audio multiplexers, microphones, loudspeakers, video monitors, video stores, still image printers, etc.

The system is supports multicasting and bidirectional streaming. It used by human operators to: get a visual confirmation of the status of the remote plant, video-conference, faking human presence, remote diagnosis, etc.

6.12 Strategic operation of Cement Plants

PIKMAC is an operator support system designed to address plant-wide strategic decision making in a cement plant. The system is used by operators specially in night and weekend shifts when there is only one one person in the plant (see Figure 11).

The system is composed by a collection of interacting CORBA objects that provide four top level functionalities:

- Clinker quality estimation using neural network technology.
- Instantaneous cost estimator using deep models.
- Alarm management using expert systems.
- Inter-shift communication using multimedia technology.

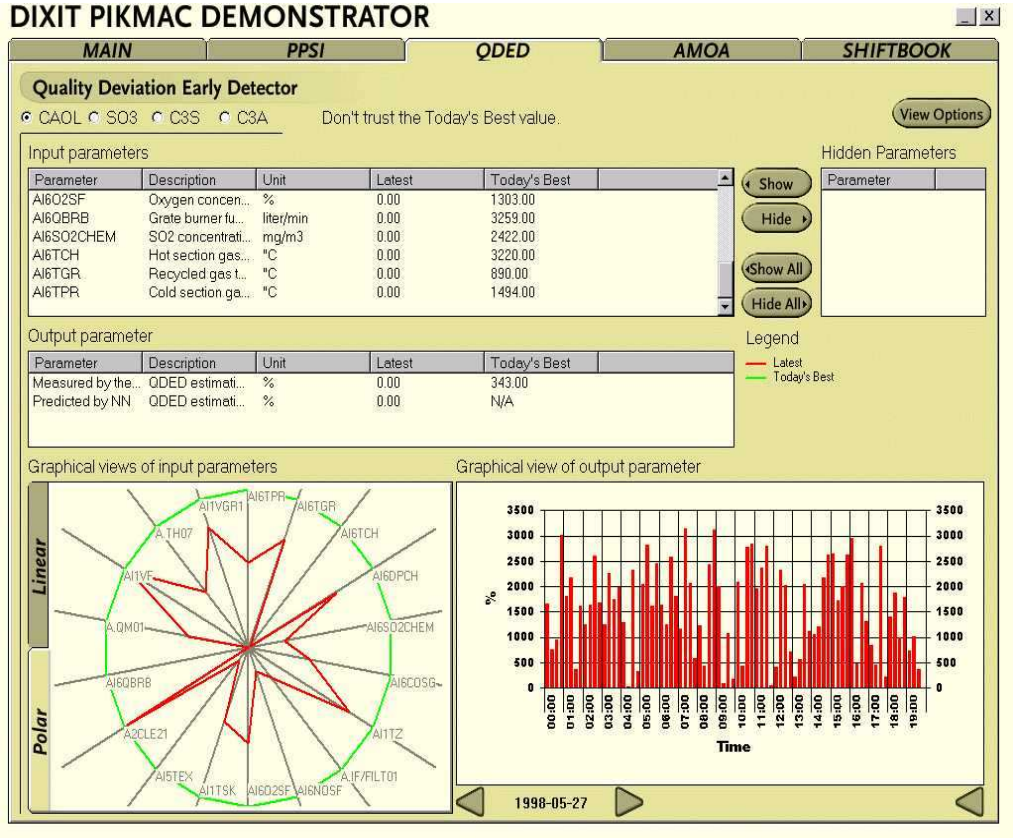

Figure 11: Part of the user interface that shows the results of the on-line quality estimator QDED. It uses neural networks to estimate present clinker quality because it is not possible to have a direct real-time measure of it.

6.13 Substation Automation

In the DOTS project, [San01], an example of using CORBA automation objects embedded in field devices based on the Electric Utilities IEC 61850 Draft standard is presented, see Figure 12.

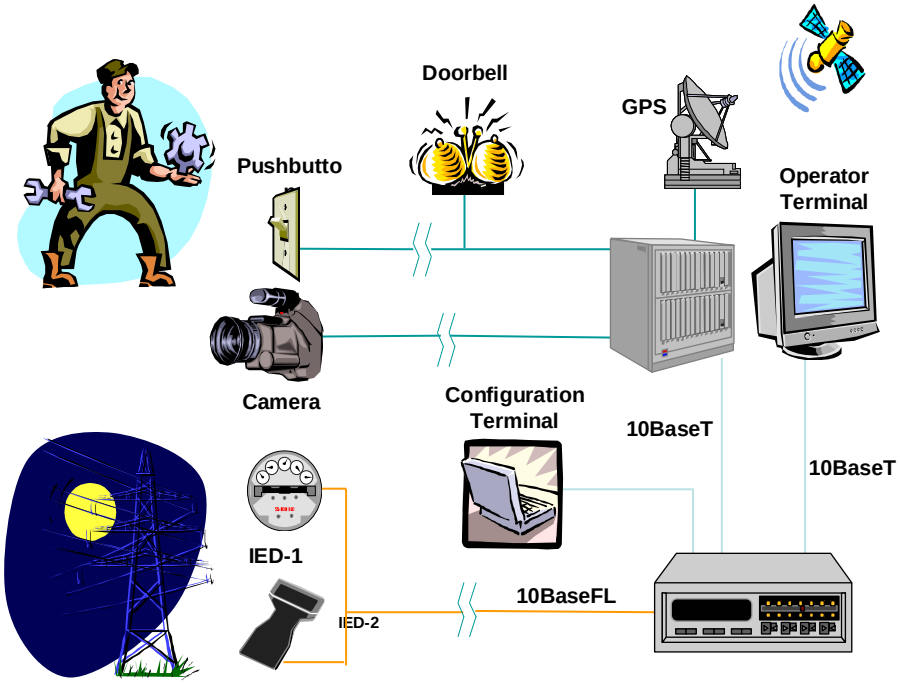

Figure 12: *The DOTS application in substation automation.*

7 CORBA in the Control Loop

7.1 CORBA Controllers

The basic distributed control loop can be modeled in CORBA in at least two different ways:

- **Distributed CORBA Loop:** In the distributed CORBA approach the sensor node, the controller node, and the actuator node are all equipped with ORBs, see Figure 13. Inside the corresponding nodes, the sensor, actuator, and controller are all modeled as CORBA objects. Different possibilities exist with respect to which of these objects that should be active objects. One possibility is to let the sensor object be an active time-triggered object that periodically takes a sensor measurement and invokes an *execute* operation on the passive controller object. The *execute* operation would typically be a one-way message. During the execution of the execute operation the controller object would invoke the *actuate* operation on the actuator object passing the control signal as the argument. Also this can be a one-way message. Another alternative is to let the controller be an active time-triggered object that every sample invokes the two-way operation *getMeasurement* on the sensor object, calculates the control signal, and invokes the *actuate* operation on the actuator object. A drawback with this approach is the risk for sampling jitter in the control loop.

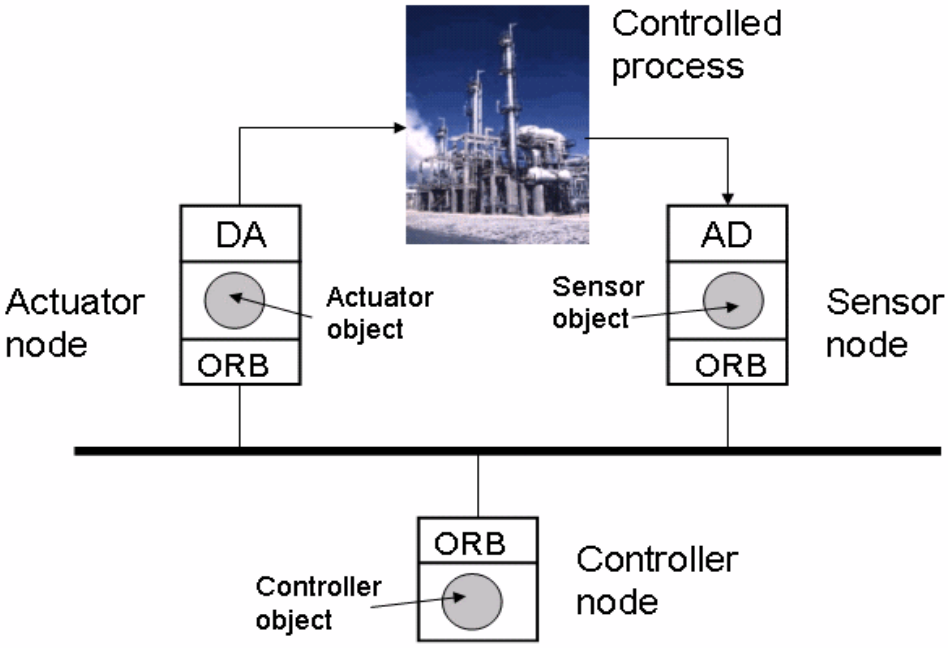

Figure 13: Distributed CORBA Control Loop

- **Encapsulated CORBA Loop:** In this approach we only require the controller to be a CORBA-compliant. The controller object then implements the communication with the actuator and sensor nodes using non-CORBA technology, e.g. using some fieldbus. The controller object could be active or it invoked from some other client representing the task execution control of the controller. The approach is outlined in Fig. 14.

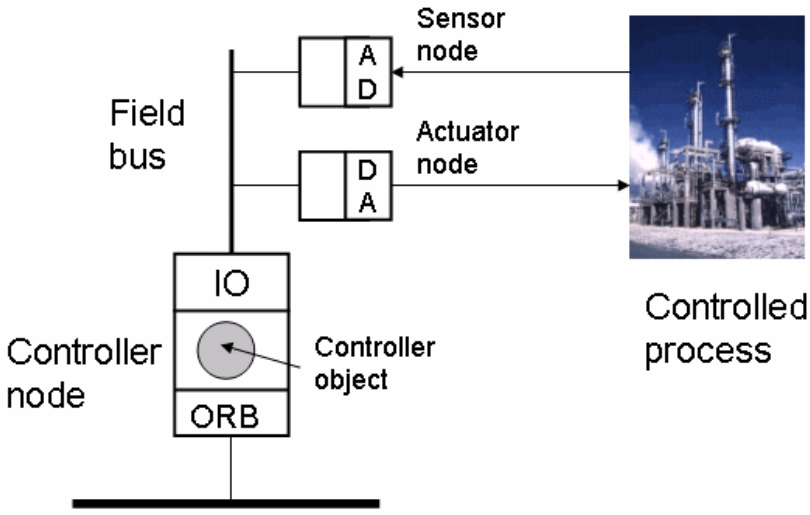

Figure 14: Encapsulated CORBA loop

Since the feedback loop is closed over the network the stability and control performance is directly affected by the quality of the transport. A network that possesses too long latencies or have too much timing variations may cause the control performance to deteriorate or completely fail. However, if the network characteristics are well known it is sometimes possible to take them into account when designing the control algorithm or compensate for them on-line. The need for a network with well-known timing properties is evident in the design of such a networked control systems. Both the timing properties of the network transport and of the protocol stacks must be taken into account when analyzing the timely behavior.

7.2 Timing Constraints

The basic control loop has two main timing constraints. The first is the sampling period, h, which should be constant, i.e., without jitter. The second constraint involves the *input-output latency*, ⚜⚜, from the sampling of the measurement signal to the control signal actuation. This is also known as the *control delay* or the *computational delay.* In a distributed control loop the input-output latency also includes the *communication delays* from the sensor node to the controller node and from the controller node to the actuator node. The constraints are illustrated in Figure 15, where we assume that the controller is implemented in a single task.

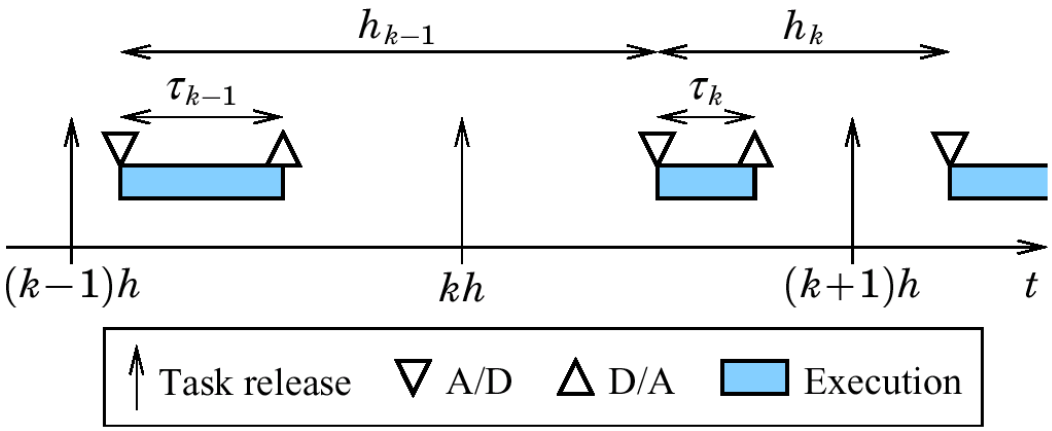

Figure 15: Control loop definitions.

From a control performance point of view the following specifications are important:

- The sampling period should be constant, i.e., the sampling jitter should be negligible. This holds for all time-triggered control loops, which is the most common case. An example of the opposite, i.e., an event-triggered control loop is found in combustion engine control systems which normally are sampled against crankshaft revolutions.
- The input-output latency should be negligible or constant, i.e. without jitter. A negligible latency can be ignored and a constant latency can be compensated for statically in the control design.
- Most control loops are more sensitive to latency than to sampling jitter.
- In most cases it is better to have a small latency with jitter than a larger latency without jitter, even if the larger latency is compensated for in the control design.

7.3 Loop Timing

Equidistant sampling intervals and a negligible or constant control delay from sampling to actuation. However, this can seldom be achieved in practice. Within a node, tasks interfere with each other through preemption, and blocking when accessing shared resources. The execution times of the tasks themselves may be data-dependent or may vary due to hardware features such as caches. On the distributed level, the communication gives rise to delays that can be more or less deterministic depending on the communication protocol. Some sources of communication delays are:

- **Processing delay:** The time required to process the message (e.g. a packet) within the nodes (hosts, routers, bridges) that the message passes through. At the sending at receiving hosts this consists of the time needed to pass through the different protocol layers (link-network-transport), whereas in the hosts it consists of the time it takes to examine the message header in order to decide where to direct the message. The processing delay can also include other factors, e.g., the time needed to check for bit-level errors.
- **Queuing delay:** The amount of time that the message spends in the output queue (buffer) of the host or router, waiting to be transmitted.
- **Transmission delay:** The amount of time required to transmit all of the bits in the message into the link. For routers and bridges this is also known as the store-and-forward delay.
- **Propagation delay:** The time required for the bits to propagate over the link.
- **Transport-level acknowledgement delay:** In reliable transport protocols such as TCP, acknowledgments and resending is used to guarantee a reliable connection in spite of bit errors and lost packets. The latter can be caused by buffer overflow due to congestion. This source of delay can be removed if unreliable transport protocols such as UDP can be used.
- **Link-layer resending delay:** This is the delays caused by collision detection and the subsequent back-off and resending in multi-access link layer protocols, e.g., Ethernet (CSMA/CD) or CAN (CSMA/CA). This source of delays can be removed if the network is scheduled in such a way that the collisions are eliminated, e.g., using time-division multiplexing or through the separate collision domains of switched Ethernet.

7.4 Delays in Control Design

The problems caused by delays can be approached in two ways. Either they are reduced or eliminated through the choice of implementation techniques and platforms, for example, a time-driven static scheduling like TTP, or the controller is designed to be robust against, or even dynamically compensate, for timing variations.

Handling of delays control systems design involves three different activities:

- Delay Modeling.
  In order to be able to analyze the system a suitable delay model must be chosen.
- Analysis.

The delayed system must be analyzed with respect to, e.g., stability and performance.

- Synthesis.
  A controller must be designed in such a way that the control loop meets its performance specifications in spite of the delays.

Delays can be modeled in several ways:

- Constant delays
- **Independent random delays.** The delays are modeled by a random distribution. The delays are independent from transfer to transfer.
- **Dependent random delays**. The delays are dependent from transfer to transfer. The delay can, e.g., be modeled by a Markov chain containing different states for different traffic conditions ("low load", "medium load", "high load")

Constant delays are straightforward to handle. In the continuous-time domain, the influence of the delay on the control loop performance can easily be decided using, e.g., frequency-domain techniques (Nyquist and Bode diagrams). The delay gives rise to a phase lag that decreases the phase margin. Constant delays can be compensated for using, e.g., Otto-Smith controllers or using a lead-filter compensation link.

In the discrete-time domain a constant delay is normally handled by including it in the sampled process model. For example the following continuous-time process model, where x denote the state vector, u the control signal, and y the process output and where we assume that the delay, τ, only occurs on the process input, i.e., between the controller node and the actuator node

$$\frac{dx}{dt} = Ax(t) + Bu(t-\tau)$$
$$y(t) = Cx(t) + Du(t)$$

has the following sampled representation. i.e., what in continuous-time is a infinite-dimensional system becomes time-invariant finite-dimensional discrete-time system.

$$x(kh+h) = \Phi x(kh) + \Gamma_0(\tau_k)u(kh) + \Gamma_1(\tau_k)u(kh-h)$$

where

$$\Phi = e^{Ah}$$

$$\Gamma_0 = \int_0^{h-\tau} e^{As}\, ds\, B$$

$$\Gamma_1 = \int_{h-\tau}^{h} e^{As}\, ds\, B$$

Constant delays are most easily achieved using a time-triggered approach, i.e., using statically scheduled computations and a statically-scheduled time division protocol such as TTP, see Figure 16.

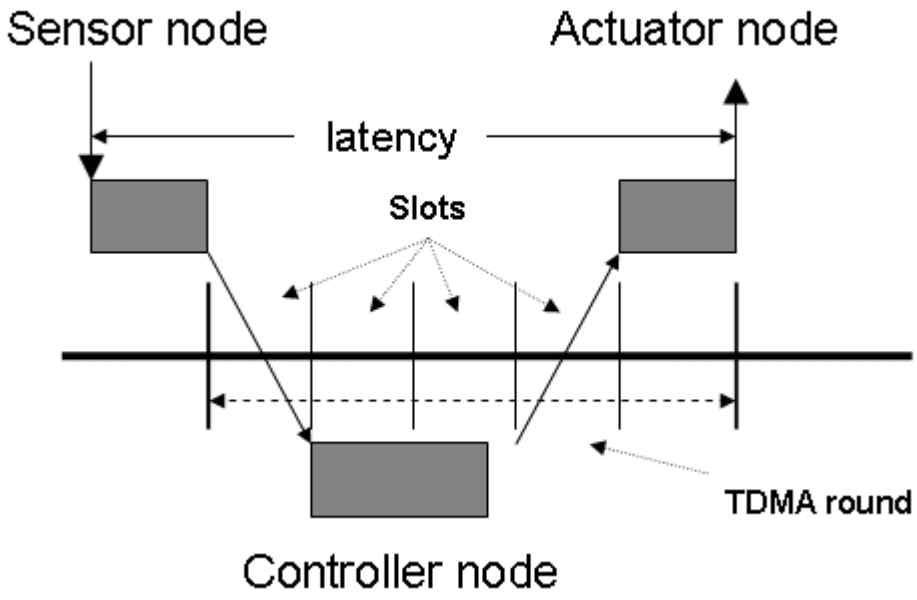

Figure 16: TDMA-based networked control loop.

The communication between the sensor and the controller and between the controller and the actuator is only performed within pre-defined slots within the TDMA round.

For varying and random delays the situation easily becomes more complex. For example, it is possible to find systems that are stable for all constant delays, but become unstable when the delay varies. A number of theoretical results are available. However, it is important to always keep in mind the assumptions that are made. For example, if the results are applicable only to constant delays but where the delays may lie within a certain range, i.e. a parametric uncertainty, or whether the results are applicable to varying delays within a certain range.

A system with varying delays can be sampled in the same way as a system with constant delays. The resulting system

$$x(kh+h) = \Phi x(kh) + \Gamma_0(\tau_k)u(kh) + \Gamma_1(\tau_k)u(kh-h)$$

with

$$\Phi = e^{Ah}$$
$$\Gamma_0(\tau_k) = \int_0^{h-\tau_k} e^{As}\,ds\,B$$
$$\Gamma_1(\tau_k) = \int_{h-\tau_k}^{h} e^{As}\,ds\,B$$

now is a time-varying system. The same holds for the closed loop system (the controlled processes in a feedback connection with the controller),

$$z(kh+h) = \Phi_{cl}(\tau_k)z(kh) + \Gamma_{cl}(\tau_k)e(kh)$$

where ***e(kh)*** is the process noise. The same approach can also be used to handle sampling jitter. In this case the closed loop system matrices will also depend on the sampling intervals h_k.

Stability analysis methods are available both for the case when the delays vary according to a certain periodic pattern and for the case when the delays change randomly. In the latter case the analysis is based on Lyapunov methods. The system is stable if one can find a common Lyapunov function for all the delays.

In [Lin02a] a new stability criterion was presented for systems with varying time delays based on the small gain theorem. The criterion has a nice graphical frequency-domain interpretation.

For systems with varying delays it is in many cases possible for the controller to compensate for the delay, or part of the delay, provided that it has access to the actual delays. The normal approach for handling this is to use associate time information with the transmitted signal values. The time information is typically expressed with time stamps.

7.4.1 Optimal LQG Control

In [Nil98] an LQG-based control scheme was presented. The approach is based on time-driven sampling and event-driven control and actuation. It is assumed

that the full state can be measured and that all states from the same node are transmitted in the same frame. Further it is assumed that old delays are known through time-stamping and that the delays are independent with known distribution. Using this scheme the results in Fig. 17 can be achieved. There the accumulated loss functions are shown for four different cases: a design neglecting the time delays, a design based on knowledge of the mean delay, a design based on introducing buffering thereby increasing the delays but eliminating the delay jitter (the Luck-Ray scheme), and the optimal LQG scheme.

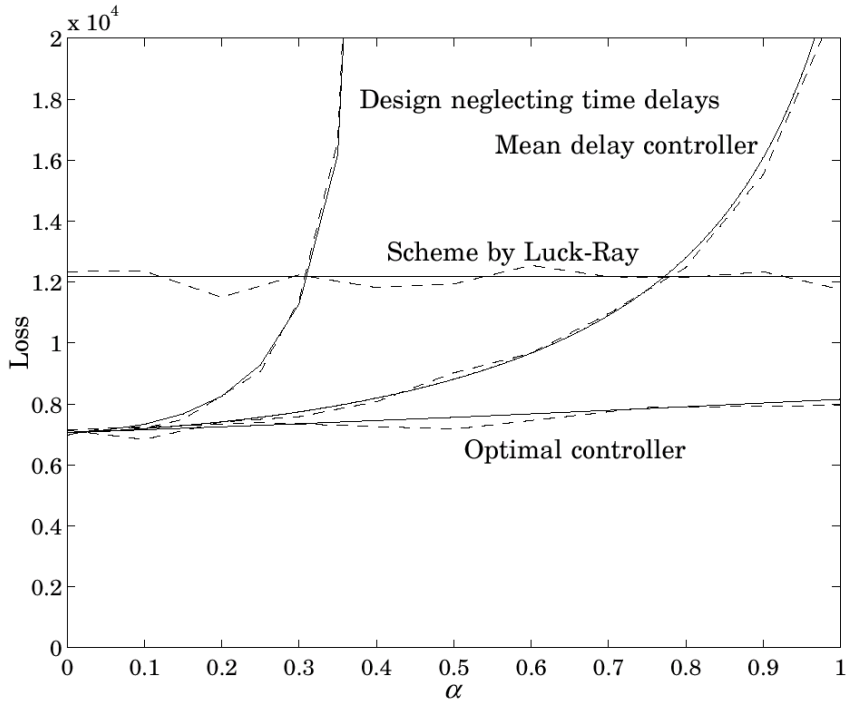

Figure 17: A comparison between different time delay compensation schemes.

The LQG scheme has been extended in several directions relaxing several of the assumptions. For example, to handle incomplete state knowledge (using estimators and output feedback), to handle dependent delays modeled by a Markov chain, networks with different delay distributions between sensor-controller and controller-actuator, systems with sampling jitter in the time-driven sensor node, and multi-input multi-output (MIMO) systems.

Within the same framework it is possible to investigate the effects of lost samples and whether it can be advantage to use timeouts for measurement signals. The idea behind the latter is to use a prediction-based controller that bases the feedback on a prediction of the measurement rather that the measurement itself when the timeout has expired. For control of systems with large measurement noise it is often an advantage to use timeouts.

7.4.2 Loop-shaping dynamic jitter compensation

The LQG approach described above has quite large demands on the available information about the controlled process (process model) and the delay characteristics. In [Lin02b] a substantially simpler approach was proposed. The approach is based on time-stamping, but it does not require full process model knowledge (only at high frequencies) and it does not assume any knowledge about the delay statistics. The approach is based on the addition of a linear delay-compensator to an existing controller. Frequency-domain conditions are used to evaluate stability and performance, Using loop-shaping techniques stability compensation and performance compensation can be achieved

7.5 Analysis using Jitterbug

If not handled correctly the delay and jitter introduced by the computer and communication system can lead to significant performance degradation. To achieve good performance in systems with limited computer resources, the constraints of the implementation platform must be taken into account at design time. To facilitate this, software tools are needed to analyze and simulate how the timing affects the control performance.

Jitterbug is a new Matlab-based toolbox that makes it possible to compute a quadratic performance criterion for a linear control system under various timing conditions [Lin02c]. The tool can also compute the spectral density of the signals in the system. Using the toolbox, one can easily and quickly assert how sensitive a control system is to delay, jitter, lost samples, etc., without resorting to simulation. The tool is quite general and can also be used to investigate jitter-compensating controllers, aperiodic controllers, and multi-rate controllers.

The use of Jitterbug assumes knowledge of sampling period and latency distributions. This information can be difficult to obtain without access to measurements from the true target system under implementation. Also, the analysis cannot capture all the details and nonlinearities (especially in the real-time scheduling) of the computer system. A natural approach is to use simulation instead. However, today's simulation tools make it difficult to simulate the true temporal behavior of control loops. What is normally done is to introduce time delays in the control loop representing average-case or worst-case delays.

7.5.1 Calculating Control Performance

The example we will look at is networked DC-servo control loop. The sensor, the actuator, and the controller are distributed among different nodes in a network. The sensor node is assumed to be time-driven, whereas the controller and actuator nodes are assumed to be event-driven. At a fixed period h, the sensor samples the process and sends the measurement sample over the network to the

controller node. There the controller computes a control signal and sends it over the network to the actuator node, where it is subsequently actuated. In this example, we will assume a CAN-type network where transmission of simultaneous messages is decided based on priorities of the packages.

We will begin by investigating how sensitive the control loop is to slow sampling and delays, and then we will look at delay and jitter compensation.

7.5.1.1.1 The DC servo process is given by the continuous-time system

$$G(s) = \frac{1000}{s(s+1)}$$

The process is driven by white continuous-time input noise. There is assumed to be no measurement noise. The discrete-time PD controller is implemented as,

$$H(z) = -K\left(1 + \frac{T_D}{h}\frac{z-1}{z}\right)$$

where the controller parameters are chosen as $K = 1.5$ and $T_D = 0.035$. (A real implementation would include a low-pass filter in the derivative part, but that is ignored here.)

The delays in the computer system are modeled by the two (possibly random) variables τ_1 and τ_2. The total delay from sampling to actuation is thus given by $\tau_{tot} = \tau_1 + \tau_2$ It is assumed that the total delay never exceeds the sampling period.

Finally, we need to specify the control performance criterion to be evaluated. As a cost function, we choose the sum of the squared process input and the squared process output:

$$J = \lim_{T\to\infty} \int \left(y^2(t) + u^2(t)\right)dt$$

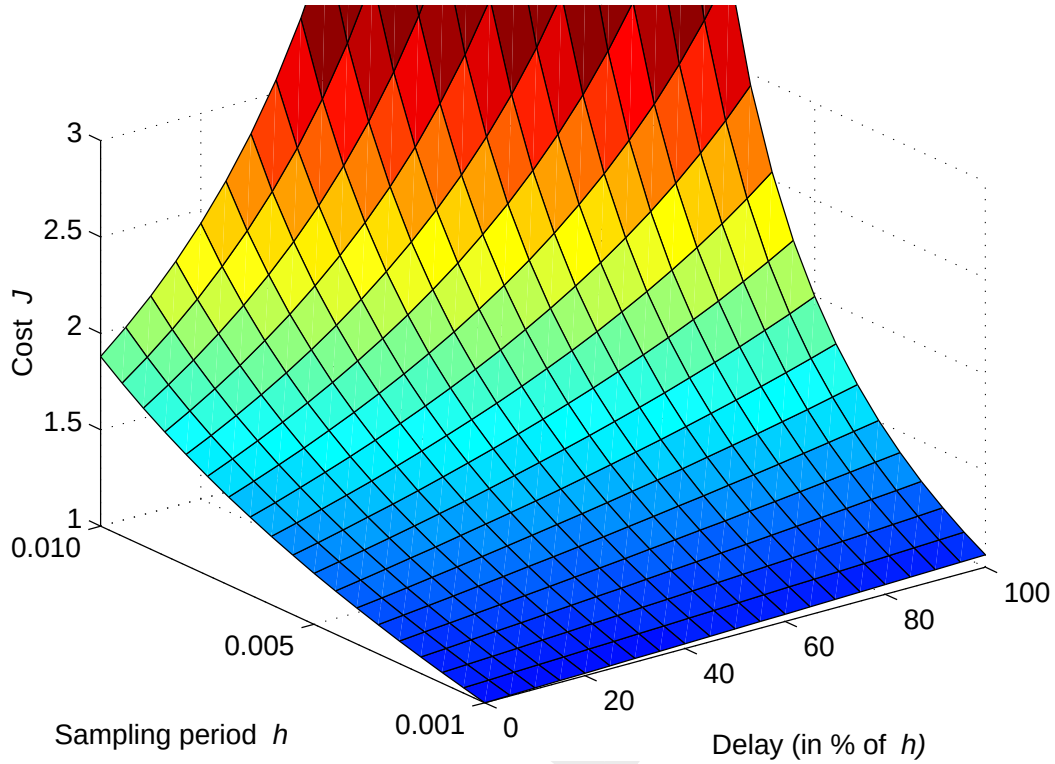

Figure 18: The cost function computed using Jitterbug. The plot shows the cost as a function of sampling period and delay in the network.

A control system can typically give satisfactory performance over a range of sampling periods. In textbooks on digital control, rules of thumb for sampling period selection are often given. One such rule suggests that the sampling interval h should be chosen such that

$$0.2 < \omega_b h < 0.6$$

where ω_b is the bandwidth of the closed-loop system. In our case, a continuous-time PD controller with the given parameters would give a bandwidth of about $\omega_b = 80 rad/s$. This would imply a sampling period of between 2.5 and 7.5 ms. The effect of computational delay is typically not considered in such rules of thumb, however. Using Jitterbug, the combined effect of sampling period and computational delay can be easily investigated. In Fig.18, the cost function J for the networked control system has been evaluated for different sampling periods in the interval 1 to 10 milliseconds, and for constant total delay ranging from 0 to 100% of the sampling interval. As can be seen, a one-sample delay gives

negligible performance degradation when $h = 1$ ms. When $h = 10$ ms, a one-sample delay makes the system unstable (i.e., the cost J goes to infinity).

7.6 TrueTime

The new Matlab/Simulink-based tool TrueTime facilitates simulation of the temporal behavior of a networked control loops consisting of nodes with multitasking real-time kernel executing controller tasks, and communication networks [Hen02]. The tasks are controlling processes that are modeled as ordinary Simulink blocks. Different task scheduling policies may be used (e.g., priority-based preemptive scheduling, static cyclic scheduling, and earliest-deadline-first (EDF) scheduling) and different communication protocols can be used (e.g., Ethernet, CAN, and TDMA). (For more on real-time scheduling, see [Liu00] and for more on communication protocols, see HRTC Deliverable 2.1.)

TrueTime makes it possible to study more general and detailed timing models of computer-controlled systems. The toolbox offers two Simulink blocks: a Real-Time Kernel block and a Real-Time Network block, see Figure 19. The delays in the control loop are captured by simulation of the execution of tasks in the kernel and the transmission of messages over the network.

The Simulink blocks are event-driven, so there is no need to specify a time-grain for the model. The execution of a task can be simulated on an arbitrarily fine timescale by dividing the code into segments. We need not simulate the task execution on instruction level. In fact, it is enough to model the timely aspects of the code that are of relevance to other tasks and to the controlled plant. This includes computational phases, input and output actions, and blocking of common resources (other than the CPU).

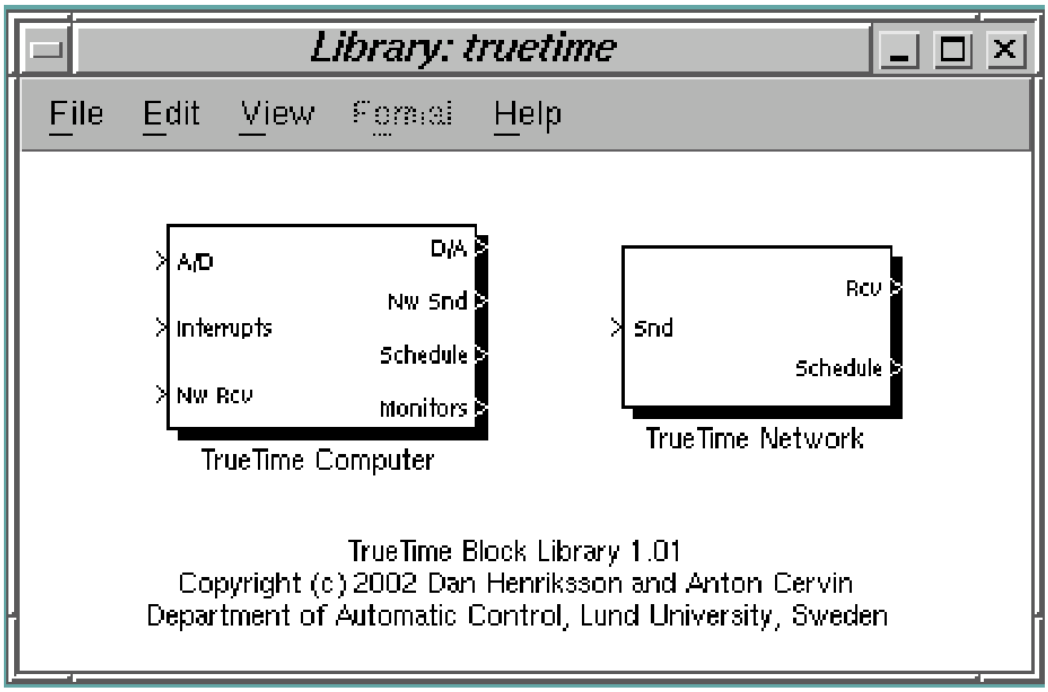

Figure 19. The TrueTime block library. The Kernel block is used to simulate a real-time kernel and the network models the behavior of a network

7.6.1 Simulating a Distributed Control System

As a recurring example in this section, we will study a control loop that is closed over a communications network. Closing control loops over networks is becoming increasingly popular in embedded applications because of its flexibility, but it also introduces many new problems. From a control perspective, the computer system will introduce (possibly random) delays in the control loop. There is also the potential problem of lost measurement signals or control signals. From a real-time perspective, the first problem is figuring out the temporal constraints (deadlines, etc.) of the different tasks in the system, and then scheduling the CPUs and the network such that all constraints are met during runtime.

We will study the setup in Fig. 20. In our control loop, the sensor, the actuator, and the controller are distributed among different nodes in a network. The sensor node is assumed to be time-driven, whereas the controller and actuator nodes are assumed to be event-driven. At a fixed period *h*, the sensor samples the process and sends the measurement sample over the network to the controller node. There the controller computes a control signal and sends it over the network to the actuator node, where it is subsequently actuated. This kind of setup was studied in [Nil98], where an optimal, delay-compensating LQG controller was derived. Here we are more interested in the interplay between control and real-time design and choose to study a simple process and controller. Using Jitterbug

we can theoretically calculate the impact of sampling period, delay, and jitter on the control loop performance, while TrueTime allows us to simulate the timely behavior complex distributed real-time control systems.

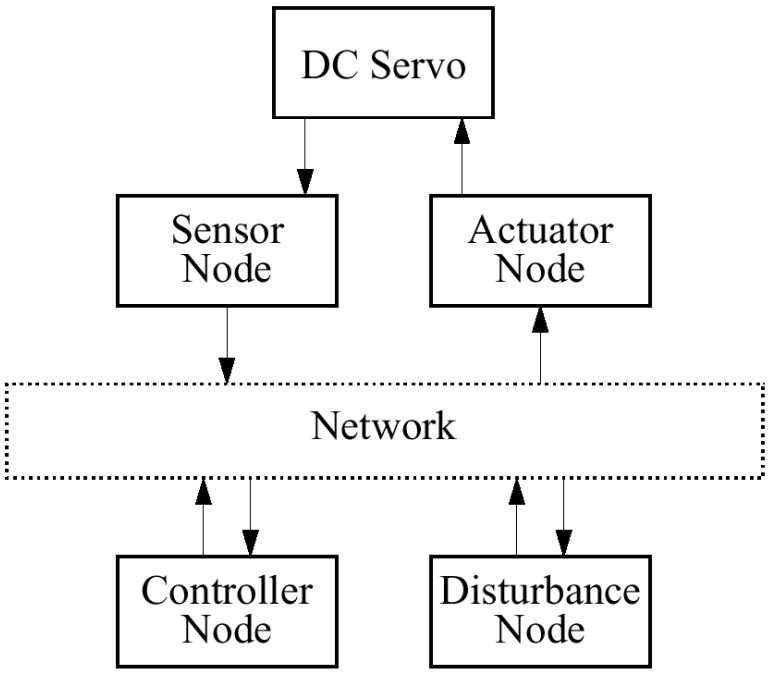

Figure 20: A networked control system.

In this example, we will assume a CAN-type network where transmission of simultaneous messages is decided based on priorities of the packages. The PD controller executing in the controller node is designed a 10-ms sampling interval. The same sampling interval is used in the sensor node.

In a first simulation, all execution times and transmission times are set equal to zero. The control performance resulting from this ideal situation is shown in Fig. 21.

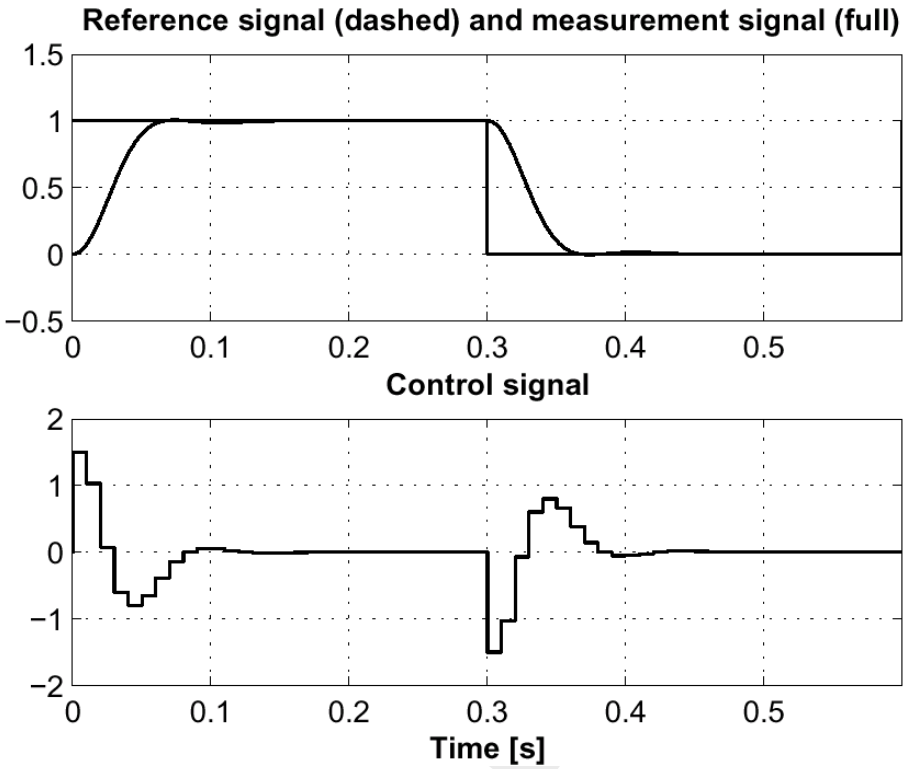

Figure 21: Control performance without time delay

Next we consider a more realistic simulation where execution times in the nodes and transmission times over the network are taken into account. The execution time of the controller is 0.5 ms and the ideal transmission time from one node to another is 1.5 ms. The ideal round-trip delay is thus 3.5 ms. The packages generated by the disturbance node have high priority and occupy 50% of the network bandwidth. We further assume that an interfering, high-priority task with a 7-ms period and a 3-ms execution time is executing in the controller node. Colliding transmissions and preemption in the controller node will thus cause the round-trip delay to be even longer on average and time varying. The resulting degraded control performance can be seen in the simulated step response in Fig. 22.

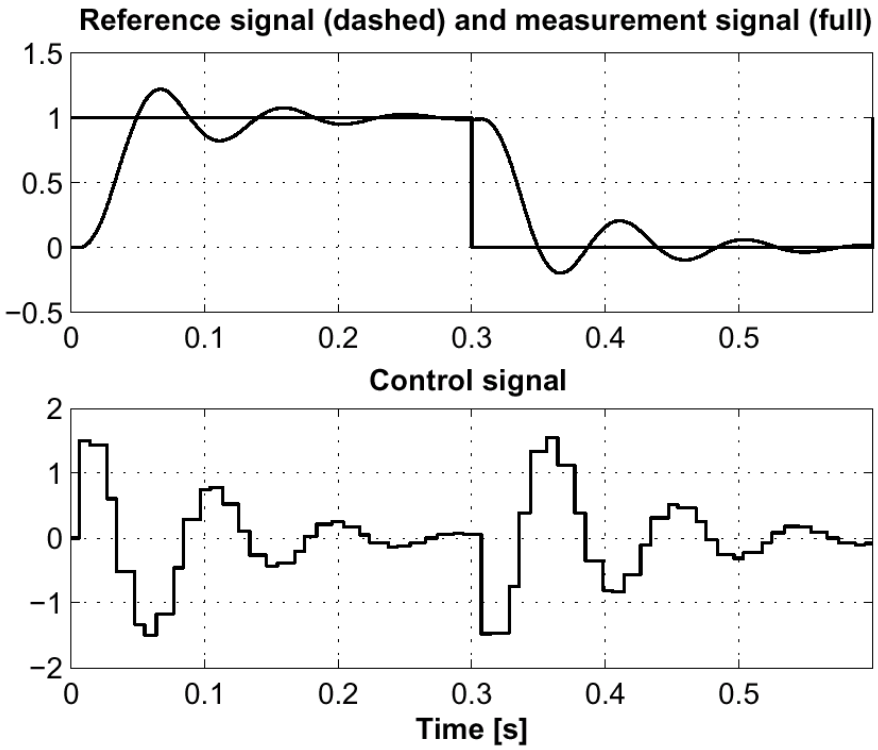

Figure 22: Control performance with interfering network messages and interfering tasks in the control node.

The execution of the tasks in the controller node and the transmission of messages over the network can be studied in detail (see Fig. 23).

Finally, a simple compensation is introduced to cope with the delays. The packages sent from the sensor node are now time-stamped, which makes it possible for the controller to determine the actual delay from sensor to controller. The total delay is estimated by adding the expected value of the delay from controller to actuator. The control signal is then calculated based on linear interpolation among a set of controller parameters pre-calculated for different delays. Using this compensation, better control performance is obtained, as seen in Fig. 24.

This small example demonstrated that while the control performance degrades severely due to latency and jitter it is sometimes possible to handle this by a more advanced control algorithm.

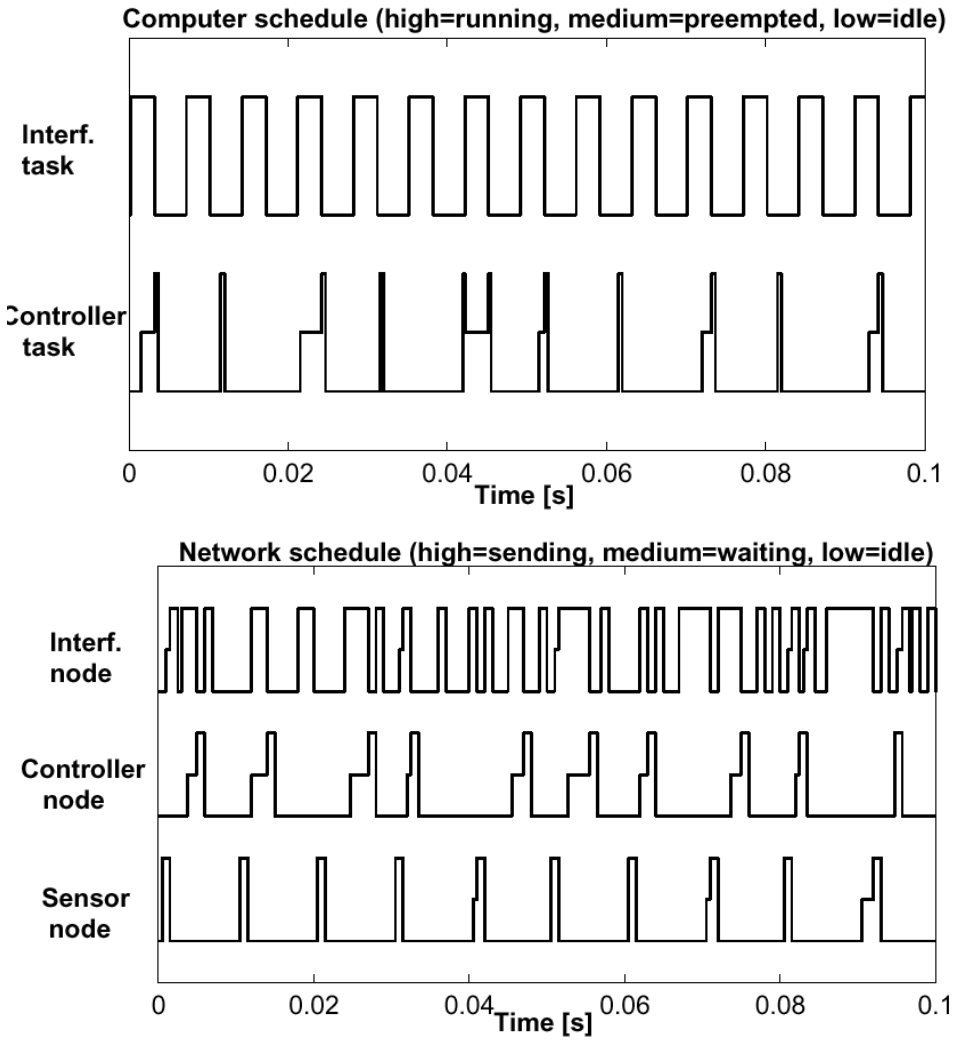

Figure 23: Close-up of schedules showing the allocation of common resources: network (top) and controller node (bottom). A high signal means sending or executing, a medium signal means waiting, and a low signal means idle.

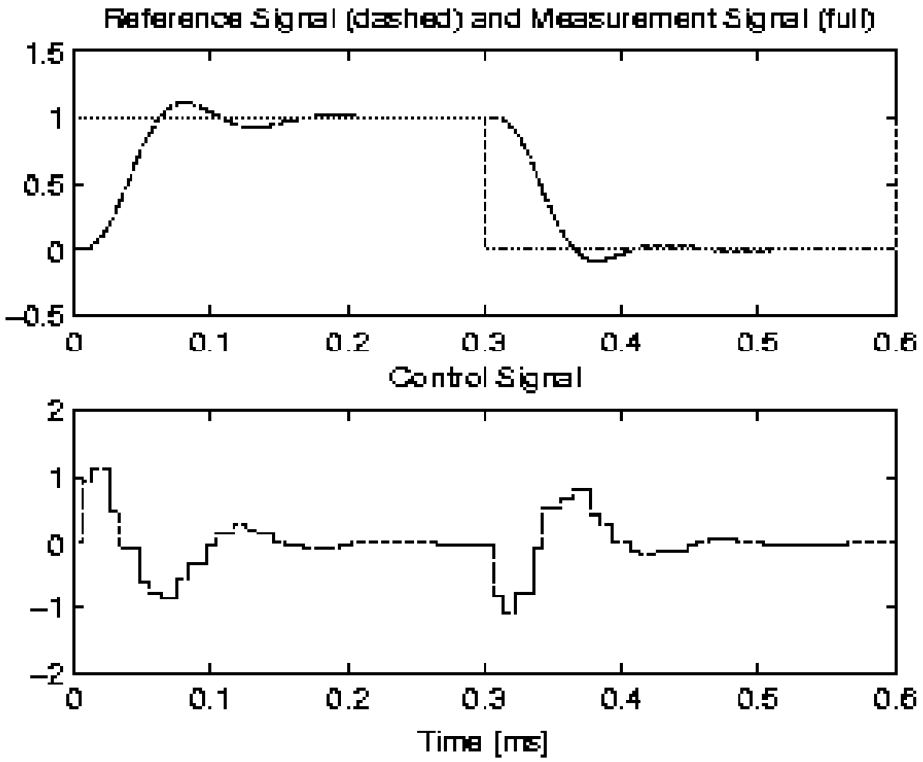

Figure 24: Control performance with delay compensation.

A screen capture of a typical TrueTime session is shown in Fig. 25.

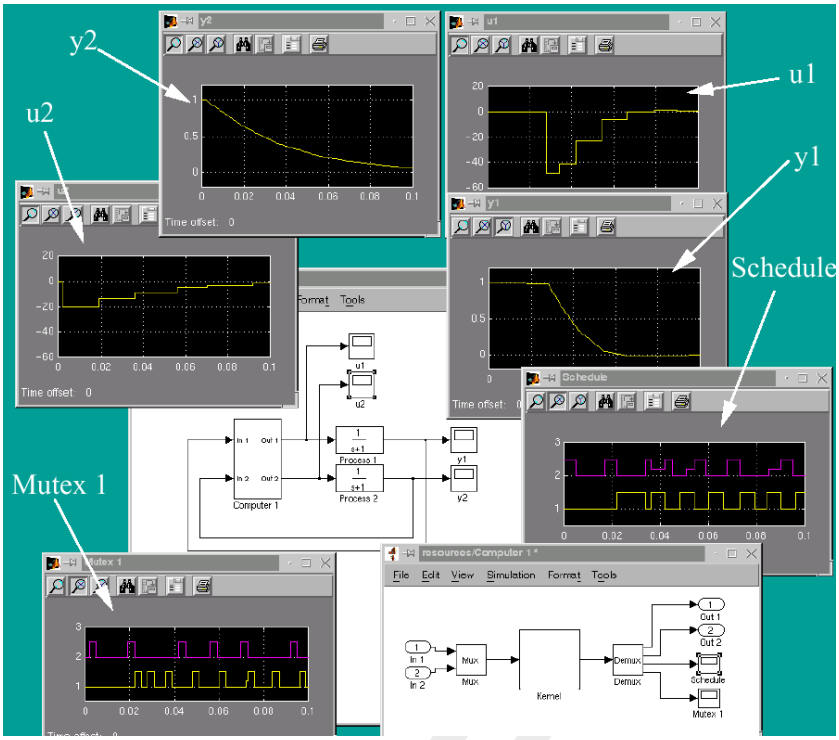

Figure 25: Screen capture from a typical TrueTime session.

7.7 TrueTime Simulation of CORBA Control Loops

Within HRTC TrueTime will be used to simulate CORBA-based control systems, within the context of the Robot Control Testbed. Using TrueTime it is possible to simulate both shared and switched Ethernet. TrueTime also supports TDMA networks. Hence it will be possible to simulate both the Real-Time Ethernet and the TTP/C approach to Hard R-T CORBA. Furthermore, support for TCP has recently been implemented. This makes it possible to also simulate networked control loops using TCP and both CORBA and R-T CORBA 1.1. This will be further described in future HRTC deliverables.

8 Summary

The use of CORBA in industrial control systems and for embedded real-time applications with hard time constraints pose a number of challenges that current CORBA and R-T CORBA do not meet. The following requirements summarize the most important issues:

- Backward Compatibility
  The HRT CORBA mat not deviate too much from CORBA and RT-CORBA. It is important to maintain interoperability between HRT CORBA and conventional CORBA. If this is not the case seamless vertical control system integration is not possible.
- Scalability in size
  HRT CORBA needs to execute on a wide range of systems, from small field devices and embedded devices to PC-type computer systems. On embedded platforms a small footprint is crucial. This necessitates a modular approach where the HRT-ORB is structured into a mandatory "micro-ORB" part that provides only the absolute necessary functionality of the HRT-ORB and a number of optional parts that each adds increased functionality.
- Scalability with respect to temporal determinism
  Different control applications require different levels of temporal determinism. Hence, HRT CORBA must be able to support a range of different transport protocols with different levels of temporal determinism. For example, for certain applications it is necessary with a time-triggered transport that minimizes communication latency jitter, whereas for other applications a transport that only provides a bounded latency suffices.
- Scheduling support
  A communication network is a shared resource. 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. Communication involves at least two partners: a sender and a receiver. In order to be able to schedule the communication HRT CORBA needs to support the description of periodic invocations from a sender (client) to a receiver (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. In order to be able to schedule also the computation within the nodes it must also be possible to associate information of a temporal nature with the CORBA objects, e.g., whether objects are active or passive, worst-case execution times, deadlines, whether object operations are blocking, etc.

9 References

[Ast97] K. J. Åström and B. Wittenmark *Computer Controlled Systems*. Third ed. Prentice-Hall. New York, NJ. 1997.

[Bla00] M. Blanke, Ch. Frei, F. Kraus, R. J. Patton and M. Staroswiecki *Fault-tolerant control systems*. In: Aström et al. (eds) *Control of Complex Systems*. Springer. 2000.

[Gre99] E. Gressier-Soudan, M. Epivent, A. Laurewnt, R. Boissier, D. Razafindramary, M. Raddadi *Component Oriented Control Architecture: the COCA Project* Microprocessors and Microsystems, 23, 1999.

[Gup96] M. M. Gupta and N.K. Singh *Intelligent Control Systems*. IEEE Press. Piscataway, NJ. 1996

[Hen02] D. Henriksson, A. Cervin and K.-E. Årzén, *TrueTime: Simulation of Control Loops Under Shared Computer Resources*, Proceedings of the IFAC World Congress, Barcelona, 2002

[Kop97] H. Kopetz, Real-Time Systems: Design Principles for Distributed Embedded Applications. Boston, MA: Kluwer, Academic Pub., 1997.

[Kus98] K. Kusunoki, I. Imai, H. Ohtani, T. Nakakawaji, M. Ohshima, and K. Ushijima A CORBA-based Remote Monitoring System for Factory Automation, First International Symposium on Object-Oriented Real-Time Distributed Computing, 1998. (ISORC 98)

[Lin02a] B. Lincoln, *A Simple Stability Criterion for Control Systems With Varying Delays*, Proceedings of the IFAC World Congress, Barcelona, 2002

[Lin02b] B. Lincoln, *Jitter Compensation in Digital Control Systems*, Proceedings of the 2002 American Control Conference.

[Lin02c] B. Lincoln and A. Cervin, Jitterbug: A Tool for Analysis of Real-Time Control Performance, Proceedings of the 41st IEEE Conference on Decision and Control, Las Vegas, 2002

[Liu00] J.W.S. Liu, *Real-Time Systems*. Upper Saddle River, NJ: Prentice Hall, 2000.

[Lüd02] F. Lüders, I. Crnkovic and A. Sjögren *Case Study: Componentization of Industrial Control Systems*, IEEE COMPSAC 2002, Oxford, UK, August, 2002.

[Mor02] Y. Morton, D. Troy and G. Pizza, An Approach To Develop Component-Based Control Software For Flexible Manufacturing

Systems, Proceedings of the American Control Conference, Anchorage, 2002

[Mül02] P. O. Müller, Ch. M. Stich, and Ch. Zeidler *Component-Based Embedded Systems* in I. Crnkovic and M. Larsson (eds) *Building reliable component-based software systems*, Arttech House, UK, 2002.

[Nil98] J. Nilsson, *Real-Time Control Systems with Delays*. PhD thesis ISRN LUTFD2/TFRT--1049--SE, Department of Automatic Control, Lund Institute of Technology, Sweden, January, 1998.

[OFC98] Open Control Foundation. Open Control Interface, Version 1.4, February 1998

[OMA98] OMAC API Work Group, *OMAC API SET, Version 0.18*, Working Document, February 1998.

[OMG95] Object Management Architecture Guide (OMA Guide), Third Edition, Wiley. June 1995.

[OMG00] Common Object Request Broker: Architecture and Specification (CORBA), Revision 3.0.

[OSA96] OSACA Open Architecture for Controls within Automation Systems OSACA I & II Final Report, Project No EP 6379 & EP 9115, EU Brussels, April 1996

[Pre02] O. Preis and M. Naedele Architectural Support for Reuse: A Case Study in Industrial Automation in I. Crnkovic and M. Larsson (eds) Building reliable component-based software systems, Arttech House, UK, 2002.

[Rod99] M. Rodríguez, and R. Sanz *HOMME: A modeling environment to andel complexity*. In: IASTED Modeling and Simulation Conference. 1999

[Rus99] J. Rushby Partitioning in avionics architectures: Requirements, mechanisms, and assurance. Technical Report NASA/CR-1999-209347. NASA. 1999.

[San99] Sanz, Ricardo, Idoia Alarcón, Miguel J. Segarra, Angel de Antonio and José A. Clavijo (1999a). *Progressive domain focalization in intelligent control systems.* Control Engineering Practice 7(5), 665–671, 1999

[San00] R. Sanz, "Agents for complex control systems," ch. 10 in *Automation, Control, and Complexity: New Developments and Directions*, T. Samad and J. Weyrauch, Eds., pp. 171–190. Chichester, U.K.: John Wiley and Sons, 2000.

[San01] R. Sanz, J.A. Clavijo, M. Segarra, A. de Antonio, and M. Alonso CORBA-based Substation Automation Systems, Proceedings of the 2001 IEEE International Conference on Control Applications, Mexico City.

[San01b] R. Sanz and M. Alonso CORBA for Control Systems, Anual Reviews in Control, vol. 25, pp. 169–181, 2001.

[Sch00] D. Schmidt and F. Kuhns *An Overview of the Real-Time CORBA Specification*, IEEE Computer, June 2000

[Sch02] D. Schmidt *Overview of CORBA* http://www.cs.wustl.edu/~schmidt/corba-overview.html

[Sha96] M. Shaw and D. Garlan *Software Architecture. An Emerging Discipline*. Prentice Hall. Upper Saddle River, NJ. 1996

[Sho00] Shokri, Eltefaat and Phillip Sheu, *Real-time distributed object computing: An emerging field.* IEEE Computer pp. 45–46, 2000

[Val92] Valenzo, Demartini, Ciminiera *MAP and TOP communcations*, Addison- Wesley, Reading, MA, 1992.

[Yan99] Z. Yang, G. Huang, R. Lim Yan Guan, R.Gay *CORBA as Object-oriented Infrastructure for Factory Communication and Control*, Communications, APCC/OECC '99. Fifth Asia-Pacific Conference on ... and Fourth Optoelectronics and Communications Conference

More information about the Object Management Group can be found at their website:
http://www.omg.org/

More information about the Control Systems Working Group can be found at their website:
http://www.omg.org/realtime/working_groups/ControlSystems.html