IST-2001-37652

Hard Real-time CORBA

Title

Engineering Handbook

For CORBA-based Control Systems

Authors

Ricardo Sanz (UPM)
Miguel Segarra (SCILabs)
Thomas Losert (TUW)
Julita Bermejo (UPM)
Karl-Erik Årzén (LTH)

Reference IST37652/072
Date 2003-10-23
Release 1.0
Status Final
Clearance Public

Partners

Universidad Politécnica de Madrid
Lunds Tekniska Högskola
Technische Universität Wien
SCILabs Ingenieros

Summary Sheet

IST Project 2001-37652
HRTC
Hard Real-time CORBA

Engineering Handbook

For CORBA-based Control Systems

Abstract:

The present document contains guidelines and reference information for the construction of software-intensive complex control systems based on CORBA technology.

HRTC Partners:

Universidad Politécnica de Madrid
Lunds Tekniska Högskola
Technische Universität Wien
SCILabs Ingenieros.

Release Sheet (1)

Release:	**0.1 Draft**
Date:	2003/08/09
Scope	Initial version
Sheets	All

Release:	**0.2 Draft**
Date:	2003/08/17
Scope	New structure
Sheets	All

Release:	**0.3 Draft**
Date:	2003/09/24
Scope	New content in methodology and CORBA products
Sheets	All

Release:	**0.4 Draft**
Date:	2003/09/29
Scope	Content added in OMG Specifications
Sheets	All

Release:	**0.5 Draft**
Date:	2003/10/14
Scope	Reorganisation of chapters. New content
Sheets	All

Release:	**0.6 Draft**
Date:	2003/10/19
Scope	New content on real-time issues
Sheets	All

Release:	**0.7 Draft**
Date:	2003/10/20
Scope	First Complete Draft
Sheets	All

Release:	**0.8 Draft**
Date:	2003/10/22
Scope	New real-time chapter
Sheets	All

Release Sheet (2)

Release:	**0.9 Draft**
Date:	2003/10/22
Scope	New fault-tolerant chapter. Various corrections and inclusions.
Sheets	All

Release:	**1.0 Final**
Date:	2003/10/23
Scope	Revision, corrections, indexes.
Sheets	All

Table of Contents

Part 1 Overview and introductory material 9

1 Introduction 11
1.1 Purpose of the document 11
1.2 Structure of the document 11
1.3 Sources of Information 12
1.4 How to use the document 13

2 CORBA for Control Systems Rationale 14
2.1 Introduction 14
2.2 CORBA for Control Systems 16
2.3 CORBA for EAI 18

3 Glossary 20
3.1 Introduction 20
3.2 Sources employed in this compilation 20
3.3 A special note on acronyms 20
3.4 The glossary itself 21

Part 2 OMG Technology 47

4 The Object Management Group 49
4.1 Overview and Objectives 49
4.2 Structure and Activity 50
4.3 Overview of Specifications 50

5 Overview of OMG Specifications 53
5.1 The Object Management Architecture (OMA) 53
5.2 Modelling 55
5.3 CORBA/IIOP Specifications 57
5.4 IDL / Language Mapping Specifications 60
5.5 Specialized CORBA Specifications 60
5.6 CORBA Embedded Intelligence Specifications 61
5.7 CORBAservices Specifications 62
5.8 CORBAfacilities Specifications 62
5.9 OMG Domain Specifications 63
5.10 CORBA Security Specifications 64

6 Core CORBA technology 65
6.1 The Common Object Request Broker (CORBA) 65
6.2 Overview of Architectural Components 66
6.3 CORBA IDL 69
6.4 OMA Middleware 70
6.5 CORBA for Real-time Control 71

6.6 Bridging Domains 72

7 RT-CORBA 73
7.1 Fixed-priority Real-Time CORBA 74
7.2 Dynamic-Scheduling Real-Time CORBA 82

8 Extensible Transports Framework 88

9 Fault-tolerant CORBA 90
9.1 Introduction 90
9.2 Grouping Abstractions 91
9.3 Architectural Overview 92

10 UML 95

11 CCM 98

12 Data-Distribution Service 100

13 Other Domain Specifications 102
13.1 Enhanced Views of Time 102
13.2 Smart Transducers 103

Part 3 Software and Hardware 105

14 ORBs 107
14.1 Implementations of CORBA ORBs 107
14.2 Qualitative Feature Comparison 116

15 Design Tools 118
15.1 UML 118
15.2 IBM Rational 122
15.3 I-Logix Rhapsody 123
15.4 Artisan Software Real-Time Studio 125
15.5 PrismTech OpenFusion CORBA Explorer 125
15.6 No Magic MagicDraw UML 127
15.7 Microsoft Visio 127
15.8 Design Tools Comparison Chart 128

16 Platforms 129
16.1 Embedded industrial PC Boards 129
16.2 Networked Device Servers 129
16.3 Control Units 130
16.4 Time-Triggered Hardware 131
16.5 Telecom Equipment 132

Part 4 Core Methodology 133

17 A Methodological Approach 135
17.1 Methodology rationale 135
17.2 A Methodology for Complex Process Controllers 137

17.3 Key Concepts for a Methodology 138
17.4 Divide and Conquer 139

18 Basic processes 141
18.1 Introduction 141
18.2 Early requirements 142
18.3 Late requirements 143
18.4 Analysis 144
18.5 Architectural design 145
18.6 Detailed design 146
18.7 Implementation 146

19 Engineering Objects for Real-time 148
19.1 Schedulability Analysis 148
19.2 Engineering of real-time control applications 151

Part 5 Case Studies 155

20 Strategic Plant Control 157
20.1 Introduction 157
20.2 Strategic Process Control 158
20.3 Operational objectives for the Contes plant 160
20.4 The PIKMAC decision support system 162
20.5 Global Application Structure 162
20.6 Lessons learnt 169

21 Strategic Emergency Management 171

22 The HRTC Process Control Testbed 174
22.1 Introduction 174
22.2 Process description 174
22.3 Computing Components 175
22.4 Functionality 178
22.5 Hardware Setup 179
22.6 Software Setup 180

23 The Integrated Control Architecture 183

Part 6 Additional Materials 185

24 Common pitfalls 187
24.1 Don't start from the beginning 187
24.2 Overselling of CORBA solutions 187
24.3 Being religious or dogmatic about CORBA 188
24.4 Don't know why we want CORBA 188
24.5 Being generic for one-of-a-a-kind problems 188
24.6 Belief in silver bullets 189
24.7 Forget that the focus is developing software controllers 189
24.8 Forget about true physical concurrency 189

24.9 NIH Architectural Syndrome 190
24.10 Insufficient Intelligence in CORBA Agents 190
24.11 Excessive Intelligence in CORBA Agents 190
24.12 Seeing Objects/Agents Everywhere 191
24.13 Monolithic Agencies 191
24.14 All time working in the infrastructure 191
24.15 Insufficient Freedom for Agents 191
24.16 Excessive Freedom for Agents 192

Part 7 Appendices 193

25 References 195

26 OMG Specification Catalog 202

27 Final Comments 209

27.1 Some final thoughts 209
27.2 A lot of work to be done 210

Part 1
Overview and introductory material

This page has been intentionally left blank.

1 Introduction

1.1 Purpose of the document

This is the HRTC CCS Engineering Handbook. It contains several types of resources for the construction of CORBA-based controllers, from available specifications from the OMG to methodologies and RT-CORBA ORB implementations and tools.

This document can serve as base material to establish a organisation-specific methodology and background knowledge in the field.

The intended audience of this document is people involved in the specification, selection, design, implementation and/or deployment of complex distributed control systems.

It is the purpose of the authors of this document to make it available to a wide audience of control and CORBA people, making it evolve from its present, embryonic status, to a fully operative handbook status.

1.2 Structure of the document

This document is roughly structured in six main parts and some appendices. These parts are composed by several chapters each addressing specific issues.

Part 1: Overview and introductory material

This part sets the stage for what comes after. It establishes the rationale for the use of CORBA in control systems.

Part 2: OMG Specifications

This part describes available OMG specifications that are of relevance for the construction of CORBA-based control systems.

Part 3: CORBA Products

This part describes available products and tools that are of relevance for the construction of CORBA-based control systems.

Part 4: Core Methodology

This part describes a basic methodology to be used in the construction of CORBA-based controllers.

Part 5: Case Studies

Some case studies on the use of CORBA technology for the implementation of distributed control systems of varying complexity.

Part 6: Additional Materials

This part gathers useful heterogeneous material that cannot be cleanly placed in the other parts.

Part 7: Appendices

References and some final thoughts.

1.3 Sources of Information

The information in this document is based on many sources but a core reference is the summary of all OMG specifications – either published or about to be published – that are available at the OMG web site[1].

Especially for a beginner regarding OMG technologies there are good introductions and tutorials available on the web[2] that allow a quick overview on this topic. Newsgroups[3] are also a valuable resource for beginners as well as advanced CORBA-programmers.

Further there are good books about CORBA (*e.g.*, [Siegel 96]) that explain the various parts of the CORBA specification (the CORBA 3 specification

[1] http://www.omg.org/technology/documents/spec_catalog.htm
[2] http://www.omg.org/gettingstarted/index.htm or http://www.corba.org/
[3] *e.g.*, comp.object.corba

has more than 1000 pages) and provide an overview about related specifications.

Last but not least, especially in the scientific community, there are plenty of published papers and PhD theses about particular aspects related to CORBA.

1.4 How to use the document

This handbook can be used in several ways depending on the purpose and the background of the user regarding CORBA technology.

Type of Reader	Roadmap
Decision maker	Parts 1, 5 and 6
Designer/architect	Parts 1, 2, 4, 5 and 6
Programmer	Parts 1, 2, 3 and 6
Control engineer	Everything
Control customer	Parts 1 and 5

2 CORBA for Control Systems Rationale

2.1 Introduction

Nowadays, most control systems are heterogeneous distributed systems.

They are also complex systems as a great number of different hardware and software elements are combined together to provide global functions. The reason for heterogeneity is that specialised equipment performs better some functions of a system; sensor-reading and actuator control, process management and enterprise management are placed at different levels of the distributed control system and perform different tasks.

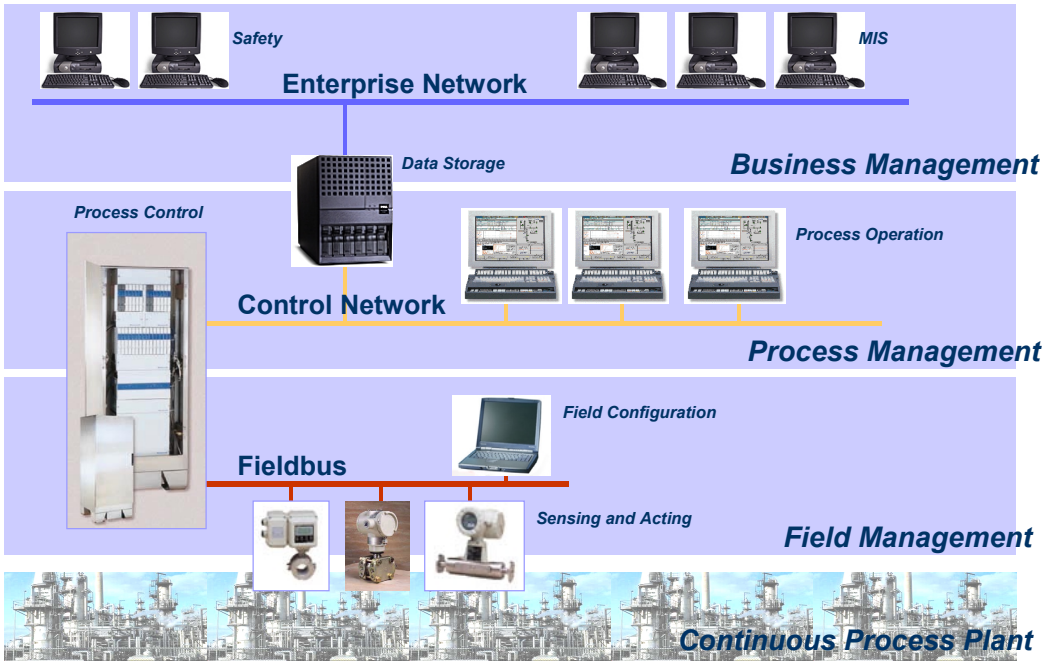


Figure 1: Heterogeneity and distribution in a process plant. Complexity is handled by means of subsystems organisation into layers: field, process, bussiness.

The main reason for distribution is that computing must be close to the process in order to ensure prompt reaction to process changes (e.g. lower-level control loops). **Figure 1** shows this situation in a complex process plant. But there are many other reasons for doing this:

- availability of suitable embeddable processors,
- timing requirements that forbid communication due to latencies,
- need of increased levels of performance that is achieved through parallelism,
- simplification of construction and maintenance tasks through modularity,
- reduction of cost and time-to-market by means of component-based reuse,
- integration of legacy systems
- availability of specific software platforms (for example for for AI)

The example of the process plant can be easily translated to other domains where distribution and heterogeneity are key factors. The reader may think of modern warfare systems where information flows from many different sources (radar, satellite, targeting, engine control data, etc.), and is drained by a similar amount of sinks (troops, avionics, combat systems, positioning systems, etc). Another example is in the automotive industry where modern ABS systems have a dedicated computer in each wheel exchanges braking information with the rest. Additionally the suspension, power management, and engine control systems work together with the ABS to keep the automobile under control. While these are common systems these days, ¿what are the common aspects among all of them?

They are difficult to build. While hardware costs keep decreasing over the years the software that runs on it becomes more and more complex and the effort and money to build it keeps rising. The problem has several facets two of which are distribution and heterogeneity, that we try to address in this work.

The best way to get rid of those problems is to make them disappear at the system development level. A common abstraction for a wide variety of systems is needed. Making the abstraction common means that it is platform-independent so perceived heterogeneity can be actually removed. To avoid the complication emerging from distribution we should develop the system as if it were a monolithic system. These are

precisely the ideas behind the Object Management Architecture (OMA), provide abstraction above low-level detailed platform-problems and provide distribution-transparency to systems in an object-oriented way. This is achieved by providing concrete object models based on the OMA; namely CORBA. Concrete implementations of the CORBA specification hand to system builders an homogeneous platform-independent infrastructure for system development.

But whereas the homogeneous CORBA abstraction has proven useful regarding some domains, the concrete object model of the CORBA architecture can be not good enough in some situations. CORBA was designed with the objectives of flexibility, interoperability and reduction of complexity without incurring in a significant loss of performance. Unfortunately, there are other control system demands and requirements as those related to real-time, embedded, and fault tolerant systems that cannot be overlooked and a careful analysis must be done.

2.2 CORBA for Control Systems

Distributed objects are a useful technology for control systems construction because control systems are software systems that continuously interact with the reality (i.e. with real, physical objects) so the software paradigm that best fits this domain is the object-oriented paradigm.

The CORBA community has been aware for a long time of the lack of suitability of the general CORBA specification for certain types of technical systems. They have also kept in mind the tremendous benefits that CORBA has brought to distributed system development. This change in the way systems are built has led engineers to think that the same concepts of abstraction and homogeneity could be applied to systems with more stringent operating conditions. These are basically systems that need to deal with the progression of time, dependability or scarce resources.

The efforts of the OMG community regarding the Distributed, Real-Time and Embedded (DRE) systems have been materialised in three different specifications which now form part of the CORBA specification:

- **Minimum CORBA specification**. This is a profile of CORBA for low-resource systems. Basically, it removes from the CORBA specification those parts which are of little use to systems where most things are known at design-time.

- **Real-Time CORBA specification**. This specification builds on top of CORBA to provide the control of resources necessary to achieve end-to-end predictability in real-time systems. The specification reuses concepts and some parts of other specifications as the Quality of Service framework from the Messaging specification or the Enhanced Time specification.
- **Fault-Tolerant CORBA specification**. There are many applications that have a need for fault-tolerance. This specification deals with the CORBA infrastructure services that an application might request to achieve fault tolerance. The specification supports a range of fault tolerant strategies such as request retry, redirection to an alternative server, and passive and active replication.

The increasing interest of the scientific/technical developer community in CORBA for technical systems have fostered a rising industry of ORB manufacturers for industrial applications. Most implementations are Real-Time CORBA ORBs built on top of the Minimum CORBA specification so as to take advantage of real-time features with a moderate use of resources.

There are some issues that have not still been fully addressed as some requirements mix-up in critical systems. These systems have requirements regarding real-time, fault-tolerance and embedded characteristics that pose a difficult problem still unsolved unless treated ad-hoc by traditional systems engineering. The OMG specifications regarding real-time, embedded, and fault tolerant systems have been done individually and whereas most real-time ORBs provide minimum CORBA implementations they do not provide reliability characteristics regarding fault-tolerance. This means that DRE applications can be fully built and deployed by the use of commercially available ORBs but the task of building the system becomes more problematic when requirements for predictability and dependability begin to be tangled together. This is also the situation in the case there is a decision to build the system in an ad-hoc way, so in benefit of CORBA the developer can always take the advantage of having a common abstraction for the development of DRE systems no matter the platform used. Other questions as the lack of awareness of the progression of time in the CORBA abstraction (IDL language) are beginning to be dealt with so in a future, all types of systems with hard real-time constraints may come into operation using CORBA technology.

Although, as reviewed, some problems still remain unsolved, the technology has established the quality of its value in a great extent of

industrial applications ranging from onboard ship command and control or research facilities to software defined radio, tactical radio systems, telecom applications or avionics. CORBA has clear advantages regarding transparency (object location), independency (from platforms and languages), scalability, flexibility, cost, etc. when compared to traditional system development procedures that can be fully applied to systems with tightly-binding requirements.

2.3 CORBA for EAI

What should be also stressed is the fact that CORBA systems effectively can bridge the gap between plant and business systems giving a convergence path toward real Enterprise Application Integration.

ITtoolbox has defined Enterprise Application Integration, or EAI, as "*the combination of processes, software, standards, and hardware resulting in the seamless integration of two or more enterprise systems allowing them to operate as one*."

EAI is not only applied to integration of systems within a business entity, EAI may also refer to the integration of enterprise systems of disparate corporate entities (B2Bi) when the goal is to permit a single business transaction to occur across multiple systems.

But in the case of control systems, what is important is the integration that CORBA enables between different business units that traditionally have been islands of IT (see Figure 2).

3 Glossary

3.1 Introduction

When elaborating complex control systems analysis and design documents we have the necessity of using highly specialized terms that sometimes do have several interpretations. This glossary includes *definitions* for words and expressions and *acronyms* with the accompanying HRTC interpretation. Entries tagged [TBD] are "to be defined" in further versions of the document.

3.2 Sources employed in this compilation

This glossary has been compiled using several well-established sources or, in some cases, defining concepts specific or strongly related to HRTC that the authors thought were necessary to include. The main contributions came from the Object Management Group (OMG) CORBA and UML Specifications, International Standards Organization (ISO) RM-ODP, IEEE Portable Applications Standards Committee (PASC) POSIX Specifications and draft specifications and FIPA Methodology Draft Glossary.

3.3 A special note on acronyms

Acronyms are not separated from the main entries because in the HRTC context (computer, control and communications) they tend to become full rank words. Nobody hesitates in using "*POSIX*" as a proper term instead of "*Portable Operating System Interface X*".

In this glossary acronyms are treated in two ways:

- In well known cases acronyms are expanded without further comment (e.g. LAN).
- In other cases, referent/referred entries are indicated with the special indication "See →" (e.g. ADT).

3.4 The glossary itself

Action	A fundamental unit of behaviour specification that represents some activity which an agent may perform. A special class of actions is the communicative acts.
4GL	An acronym meaning Fourth Generation Language. A 4GL is typically non-procedural and designed so that end users can specify what they want without having to know how computer processing is to be accomplished. Typically used to describe/configure control systems at a high, organizational level.
Abstract Class	A specialized class used solely for subtyping. It defines a common set of behaviours to be inherited by its subtypes. It has no instances. (Synonymous with Virtual Class in C++)
Abstract Data Type	A data type defined to model the data characteristics of real-world objects. An ADT provides a public interface via its permitted operations, but the internal representation and implementation of this interface are private.
Abstraction	The act of concentrating the essential or general qualities of an object or objects. The resulting concept embodies the "essence" of the objects under consideration.
Accessibility	The ability or permission to invoke a service provided by a particular object. Object-oriented programming languages implement both public and private methods of accessibility. (Synonymous with Visibility)
Accessor	A method or member function that provides a public interface to allow the "setting" or "getting" of an object's private instance variables or data members.
Acknowledged Data Transfer	The transmission of data from a source endpoint to one endpoint, or , in the case of multicast, more than one endpoint; and the subsequent response indicating the status of the data transmission.
Activation	Copying the persistent form of methods and stored data into an executable address space to allow execution of the methods on the stored data.
Active Agent	An object with a self-owned thread of control.
Actor	An external agent that interacts with an application or system. This is also a model for concurrent programming. In some context this is a synonym of → Agent.
Address	Field, or Fields, in a message identifying both the source and / or destination of the message.
ADL	→ Architecture Description Language. → Agent Description Language.
ADT	→ Abstract Data Type

Agent Description Language	A language to provide specification mechanisms for agent implementation. → Interface Definition Language
Agent Interaction Protocol	A common pattern of conversations used to perform some generally useful task. The protocol is often used to facilitate a simplification of the computational machinery needed to support a given dialogue task between two agents. Throughout this document, we reserve protocol to refer to dialogue patterns between agents, and networking protocol to refer to underlying transport mechanisms such as TCP/IP.
Agent Oriented Programming	Information systems development based on the division of the systems in a collection of interacting, semi-autonomous entities.
Agent	An entity that performs operations by his own, on behalf of itself, other systems or agents.
AIP	→ Agent Interaction Protocol
Analysis	The process of developing a specification of what a system does and how it interacts with its environment.
AOP	→ Agent Oriented Programming.
API	→ Application Program Interface
Application Facilities	Common facilities that are useful within a specific application domain.
Application Layer	The top layer, Layer 7, in the ISO Reference Model.
Application Objects	Applications and their components that are managed within an object-oriented system.
Application Program Interface	The programming interface used to access and control a library or program.
Application	A program or a set of programs that provides functionality to the end user. Also refers to specific Algorithm(s) implemented in an IED.
Architectural Domain	A → Domain for architectures, i.e. an area of knowledge defined by a family of related architectures. → Architectural Style.
Architectural Style	A common core design shared by a collection of software architectures.
Architecture Description Language (ADL)	Graphic and textual languages, standards, and conventions used to represent a software architecture. ADLs are usually related to → DSSAs. ADLs are intended to be used for building models during the development of new systems based on existing architectures or architectures in an → Architectural Domain.
Architecture	A high-level description of the organization of functional responsibilities within a system. Many different levels of architectures are involved in developing software systems, from physical hardware architecture through the logical architecture of

an application framework.

Archive	A Data Base used to retain and maintain Data records.
Assertion	An expression that evaluates to either true or false. Generally used to protect the integrity of a system or component.
Assignment	The activity of copying the values of one object into another object. The details of such an assignment vary according to the implementation language used.
Association	**1.** An Association is established when a client-server "link-up" is made and is manifested in the Communication path established between a client and a server for the exchange of messages. The Association is closed when the client-server link is Concluded or Aborted. Object association. **2.** Meaningful links between objects. A person associated with a company creates the concept of employment.
Asynchronous Event	Event that occurs independently of the execution of an application.
Asynchronous Interaction	An interaction between schedulable units (processes and/or threads) in which, after a schedulable unit invokes an operation to take part in the interaction, control is allowed to return to the schedulable unit before the completion of the interaction.
Asynchronous Message Communication	Asynchronous message communication provides the capability for objects to send messages, even without the existence of the receiving object at the instant the message is sent. The receiving object can retrieve messages at its convenience. There is no blocking or synchronization required between objects. Asynchronous message communication is a foundation for constructing true concurrent computing environments.
Asynchronous Request	A request in which the client object does not pause or wait for delivery of the request to the recipient; nor does it wait for the results.
Atomic Broadcast	The transfer of a broadcast message that is either guaranteed to be received by all possible receivers or it is not received by any.
Atomicity	The property that ensures an operation either changes the state associated with all participating objects consistent with the request, or changes none at all. If a set of operations is atomic, then multiple requests for those operations are serializable.
Attribute	An identifiable association between an object and a value. An attribute A is made visible to clients as a pair of operations: *get A* and *set A*. Read only attributes only generate a get operation. A characteristic or property of an object. Usually implemented as a simple data member or as an association with another object or group of objects.
Audience	The kind of consumer (caller) of an interface. An interface might be intended for use by the ultimate user of the service (functional interface), by a system management function within the system (system management interface) or by other participating services in

	order to construct the service from disparate objects (construction interface).
Autonomous agent	An agent which has the autonomy property (see Autonomy).
Autonomy	The autonomy of an agent can be expressed as following: 1) An agent has its own life, independently of the existence of other agents, 2) An agent is able to survive in dynamic environments without an external control, 3) An agent takes internal decisions about its behaviour only considering the perceptions, knowledge and representations it possesses.
Bearer	The kind of object that presents an interface. An object might be fundamentally characterized by the fact that it has a given interface (a specific object bears an interface), or an object can have an interface that is ancillary to its primary purpose in order to provide certain other capabilities (a generic object bears the interface).
Behaviour Consistency	Ensures that the behaviour of an object maintains its state consistency.
Behaviour	A behaviour is the observable effects of an operation or an event, including its results. It specifies the computation that generates the effects of the behavioural feature. → also Task
Belief	A belief depicts a mental state that an agent can have about the environment, other agents and about itself
Bi-directional transaction	A transaction in which a request and possibly data are conveyed from a Client to a Server and in which a response and possibly data are returned to the Client from the Server.
Binding	The selection of the method to perform a requested service and of the data to be accessed by that method. (→ also Dynamic Binding and Static Binding)
Block	A class primarily consisting of a compound statement made up of a series of operations and control structures. Block objects are used in control structures, usually as arguments for repeated or conditional execution. In-stances of this class essentially allow language constructs and operations to be bundled into an object.
Broadcast	A message placed onto a Communication Network intended to be read and acted on, as appropriate, by any device connected to the network. A Broadcast message will typically contain the sender's address and a Global recipient address. Example Time Synchronising.
Class Browser	A software facility used to view and modify classes, attributes and methods.
Built-In Type	An abstract data type that is provided as a part of the language. Also provided are the operators used to manipulate instances of built-in types.
CASE	→ Computer Aided Software Engineering
CCS	→ CORBA Control Systems → Complex Control Systems

Class Attribute	A characteristic or property that is the same for all instances of a class. This information is usually stored in the class type definition.
Class Hierarchy	Embodies the inheritance relationships between classes.
Class Inheritance	The construction of a class by incremental modification of other classes.
Class Member	A method or an attribute of a class.
Class Method	A class method defines the behaviour of the class. Such a method performs tasks that cannot or should not be done at the instance level, such as providing access to class attributes or tracking class usage metrics.
Class Object	An object that serves as a class. A class object serves as a factory. (→ Factory)
Class	A pattern that can be instantiated to create multiple objects with the same behaviour. An object is an instance of a class. Types classify objects according to a common interface; classes classify objects according to a common implementation.
Classification	The act of determining which class or type applies to a specific object.
Client	An object that requests a service from a server object in a client/server relationship. The code or process that invokes an operation on an object.
Client/server	A relationship between a client that requests services and servers that provide services. This relationship is paralleled in an O-O environment by message senders and receivers.
CM	→ Configuration Management
Cognition	The act or process of knowing; perception. O-O technology is intricately tied to how people think, act and interact while accomplishing work.
Collaboration	**1.** Two or more objects that participate in a client/server relationship in order to provide a service. **2.** Collaboration is concerned with the interactions between agents in a multiagent system when the whole system is also considered as an agent with certain structure of system's global state. Particularly, it is concerned with the relationships between individual agents' mental structures and internal states and the system's collective mental structure and state. For example, a collaborative model of multiagent systems may contain a model of system's global intention and individual agent's intention, and we can talk about congruence (that is the consistency between an agent's behaviour and the whole system's global goal or intention) and coherence (that is the consistency between an agent's internal state, such as intention, and the system's goal or intention).
COM	→ Common Object Model

Common Facilities	Facilities useful in many application domains and which are made available through Object Management Architectures (OMA)-compliant class interfaces. (→ Application Facilities)
Common Object Model	COM. Microsoft standard for object management
Common Object Request Broker Architecture	CORBA. A specification for objects to locate and activate one another through an object request broker. CORBA 2 extends the specification to facilitate object request brokers from different vendors to interoperate.
Communication Controller	A Communication Controller is a virtual device within an IED that provides communication services related to client-server associations.
Communications Stack	Hierarchy of communications software. For example the 7 Layer stack of the ISO Reference Model where each layer performs a specific functional role in Open Systems Interconnection communication.
Component	A conceptual implementation notion. A component is an object that is considered to be part of some containing object. Classes, systems or subsystems that can be designed as reusable pieces. These pieces can then be assembled to create various new applications. Sometimes used to refer to software modules with plug and play behaviour.
ComponentWare Consortium	A cooperative of companies concentrating their efforts in object based distributed systems.
Composition	The creation of an object that is an aggregation of one or more objects.
Compound Object	A conceptual notion. A compound object is an object that is viewed as standing for a set of related objects.
Computed Characteristic	An attribute derived from the values of other attributes.
Computer Aided Software Engineering	A collection of software tools that support and automate the process of analyzing, designing and coding software systems.
Concrete Class	A class or type that can have instances. (Contrast with Abstract Class).
Configuration Management	The discipline of identifying a system and its component parts at discrete points in time. Monitoring throughout versions and revisions enables CM to systematically control changes to maintain integrity and traceability of the system throughout a product's lifecycle. This includes hardware, environment, code, documents and objects.
Conformance Test	This test verifies that an object communication interface(s) complies with the specified requirements.
Conformance	A relation defined over types such that type x conforms to type y if any value that satisfies type x also satisfies type y.

Connection	An association established between Functional units for conveying information.
Connectionless	The transport of a single datagram or packet of information from one network node to a destination node, or multiple destination nodes, without the establishment of a network connection.
Connector	A n-ary relation defined over components. Usually employed in relation with software architecture modelling and → ADLs.
Constraint	A relational or behavioural restriction or limit. Usually regarded as a property that must always hold true.
Constructor	A method that is called when a new instance is created. Constructor methods are used to initialize the new instance.
Container Class	A class designed to hold and manipulate a collection of objects.
Context-Independent Operation	An operation in which all requests that identify the operation have the same behaviour. (In contrast, the effect of a context-dependent operation might depend upon the identity or location of the client object issuing the request.)
Contract	Defines the services provided by a server, along with the pre-conditions and post-conditions that apply to the use of those services.
CORBA	→ Common Object Request Broker Architecture
Coupling	A dependency between two or more classes, usually resulting from collaboration between the classes to provide a service. Loose coupling is based on generic behaviour and allows many different classes to be coupled in the same way. Tight coupling is based on more specific implementation details of the participating classes and is not as flexible as loose coupling.
CRC	Cyclic Redundancy Check. A CRC is performed for each frame and the value is included in that frame when it is transmitted. The CRC check calculation may be simple or complex depending on the protocol being used. The CRC value is used by the recipient communication interface to check and if possible correct errors incurred during transmission of that frame.
CSMA/CD	Carrier Sense Multiple Access/Collision Detection
CWC	→ ComponentWare Consortium
Data Item	A single piece of information to be communicated.
Data Link layer	Layer 2 of the ISO Reference Model, responsible for the transmission of data over a Physical medium. After establishing the Link, layer 2 performs data rate control, error detection, contention / collision detection and recovery.
Data Model	A collection of entities, operators and consistency rules.

Data Type	A categorization of values, operations and arguments, typically covering both behaviour and representation (e.g., the traditional non-OO programming language notion of type).
Datagram	A datagram contains within the message all the information for transmission without the requirement for establishing a network connection.
DCE	→ Distributed Computing Environment
DCOM	Distributed Component Object Model → Component Object Model
Declassification	The act of removing an object from a specific set of objects of a given type.
Deferred Synchronous Request	A request where the client does not wait for completion of the request, but does intend to accept results later. Contrast with synchronous request and one-way request.
Delegation	The ability of a method to issue a request in such a way that self-reference in the method performing the request returns the same object(s) as self-reference in the method issuing the request. (→ Self-Reference) The ability of an object to issue a request to another object in response to a request. The first object therefore delegates the responsibility to the second object.
Deliberative agent	A deliberative agent is a specific kind of agent that takes into account its beliefs, desires, intentions, the environment and beliefs it has on other agents to weigh its actions. A synonym could be a BDI agent.
Derivation	The act of subclassing an existing class to define a new subclass. (→ Inheritance)
Derived Class	The class created through inheritance. A derived class inherits the methods and attributes of its superclass(es) and usually adds its own to distinguish its capabilities or services.
Design Pattern	A pattern that specifies the way to construct something to satisfy some requirements (equilibrating forces) in some context.
Design	A process that uses the products of analysis to produce a specification for implementing a system. Also the result of this process. → Design Pattern → Pattern
Destructor	A method involved whenever an object is ready to be destroyed. It is usually implemented to revise the actions that were performed during initialization, such as recovery of allocated resources.
Device	A mechanism or piece of equipment designed to serve a purpose or perform a function.
Distributed Computing Environment (DCE)	OSF software specification and implementation to support development of distributed applications. (→ Open Software Foundation, → Remote Procedure Call)

Distributed Object Computing (DOC)	A computing paradigm that distributes cooperating objects across a ---possibly heterogeneous--- network and allows the objects to interoperate as a unified whole.
Distributed Objects Everywhere (DOE)	Codename of the OMA implementation project by SunSoft Inc. Commercialized as Sunsoft NEO.
DLL	→ Dynamic Link Library.
DOC	→ Distributed Object Computing.
DOE	→ Distributed Objects Everywhere.
Domain	A formal boundary that defines a particular subject or area of interest. The HRCT domain is the domain of software intensive, distributed, complex process control.
Domain Expert	A person who has special skill or knowledge of a particular domain.
Domain Model	A model (terminology and semantics) that characterize the elements, processes, and relationships within a family of related systems.
Domain-specific software architecture (DSSA)	An architecture that captures architectural commonality of multiple, related systems, i.e., systems within the same domain.
DSOM	Distributed System Object Model. → System Object Model.
DSSA	→ Domain-specific software architecture.
DTI	Data Template Identifier.
Dynamic Binding	Binding that is performed after a request is issued. (→ Binding)
Dynamic Classification	Classification of an object at runtime. This implies that an object's classification can change over time.
Dynamic Invocation	Constructing and issuing a request whose signature is not known until runtime.
Dynamic Link Library	A dynamically loaded run-time library.
Dynamic Object-Based Application	The end-user functionality provided by one or more programs consisting of interoperating objects.
EC	Event Control
Electra	An implementation of CORBA on top of the Isis and Horus reliable communications toolkits.
Embedding	Creating an object out of a non-object entity by wrapping it in an appropriate shell. (→ Wrapper)
Encapsulation	The technique used to hide the implementation details of an object. The services provided by an object are defined and accessible as stated in the object contract. (Often used interchangeably with

	Information Hiding)
Enterprise Modeling	A technique for modelling an entire business enterprise from the business manager's point of view. An enterprise model is composed of the objects, events and business rules that describe the enterprise. Separate but related business systems can be built from this model to enhance the efficiency and consistency of the operation of the enterprise.
Environment	The totality of circumstances surrounding an agent or group of agents, especially we can consider the physical and social environment. Physical Environment: The combination of external physical conditions that affect and influence the growth, development, actions and survival of agents. Social Environment: The complex of social and cultural conditions affecting the nature of an agent or a community.
Event	A significant change in the environment or the state of an object that is of interest to another object or system.
Exchange Format	The form of a description used to import and export objects.
Expandability	The expandability of a CCS is the criteria for the fast and efficient extension (both hardware and software).
Expectation Management	The process of guiding the user's expectations regarding the functionality and characteristics of any proposed system or technology.
Expert System	A rule-based program that implements the domain knowledge of a human domain expert. It is usually able to "reason" through new problems by applying its rules.
Export	To transmit the description of an object to an external entity.
Extension of a Type	The sets of values that satisfy the type.
Externalized Object Reference	An object reference expressed as an ORB-specific string. Suitable for storage in files or other external media.
Factoring	The process of extracting the common properties or behaviour from a group of objects so that the common elements can be propagated to a common subclass. Factoring eliminates duplication.
Factory Acceptance Test (FAT)	Includes customer agreed functional tests of the specifically manufactured CCS, or its parts, using the parameter set for the planned application; performed in the Manufacturer's factory using Process simulation test equipment.
Factory	A concept that provides a service for creating new objects.
Fault-Tolerance	The characteristic of a system that allows it to handle the loss of a particular component without interrupting normal operations.
Field Bus	A communications network shared by multiple, communicating, physical nodes.

Flexibility	The Flexibility is the criteria for the fast and efficient implementation of functional changes, including hardware adaptation, in an CCS.
FMS	Fieldbus Messaging Specification.
Formal Parameter	A named local object used as an argument to an operation. The value of the object (actual parameter) is assigned by the client who runs the method.
Frame Format	A template for the actual message to be transmitted. It typically defines the Header, Start of frame, Destination address, Source address, length/type of contained data, the actual data, padding bytes if required and some form of CRC data, then end of frame marker.
Framework	A set of collaborating abstract and concrete classes that may be used as a template to solve a specific domain problem.
Function	Functions are tasks that are performed in the application by the components of the CCS.
Functional Decomposition	The process of refining a problem solution by repeatedly decomposing a problem into smaller and smaller steps. The resulting steps are then programmed as separate modules.
Functional Interface	Interfaces that define the operations invoked by users of an object service. The audience for these interfaces is the service consumer, the user of the service. These interfaces present the functionality (the useful operations) of the service. An Object Service Definition.
Fusion	A second generation object-oriented development method that provides a systematic approach to O-O software development. It integrates and extends other methods. OMT/ Rumbaugh, Booch, CRC and Formal Methods.
Garbage Collection	The recovery of memory occupied by unreferenced objects, usually implemented by the language or environment.
Gateway	A network interconnection device which supports the full Stack of the relevant Protocol and can convert to a non 7 Layer Protocol for asynchronous transmission over Wide Area Networks.
Generalization	The inverse of the specialization relation.
Generic Object	An object (relative to some given Object Service) whose primary purpose for existence is unrelated to the Object Service whose interface it carries. The notion is that the Object Service is provided by having (in principle) any type of object inherit that object service interface and provide an implementation of that interface. An Object Service Domain.
Generic Operation	The concept that an operation is generic if it can be bound to more than one method.
Graphical User Interface	Any interface that communicates with the user, primarily through graphical icons.

GUI	→ Graphical User Interface.
Handle	A value that identifies an object.
Heuristic	A rule of thumb or guideline used in situations where no hard and fast rules apply. An empirical rule, or educated guess based upon past experiences.
HMI	Human Machine Interface.
Hub	A Hub is a Communications Network component providing multiple ports, each interfacing to a separate media link in a star topology.
Idiom	Low level → pattern. usually related to a specific programming language.
IDL	→ Interface Definition Language.
IED	→ Intelligent Electronic Device
Implementation Definition Language	A notation for describing implementations. The implementation definition language is currently beyond the scope of the ORB standard. It may contain vendor-specific and adapter-specific notations.
Implementation Inheritance	The construction of an implementation by incremental modification of other implementations. The ORB does not provide implementation inheritance. Implementation inheritance may be provided by higher level tools.
Implementation Object	An object that serves as an implementation definition. Implementation objects reside in an implementation repository.
Implementation Repository	A storage place for object implementation information.
Implementation	**1.** The development phase in which the hardware and software of a system become operational. **2.** A definition that provides the information needed to create an object and allow the object to participate in providing an appropriate set of services. An implementation typically includes a description of the data structure used to represent the core state associated with an object, as well as definitions of the methods that access that data structure. It will also typically include information about the intended interface of the object.
Implicit Invocation	A mechanism of invocation in which the invoker does not know anything about the invokee. Usually implemented through a callback registration mechanism in the invoker or a broadcast of an event to all possible invokees.
Import	Creating an object based on a description of an object transmitted from an external entity.
Incomplete Partition	A partition composed of some, but not all, of its partitioned

	subtypes. Information Hiding (→ Encapsulation)
Inheritance	The construction of a definition by incremental modification of other definitions. (→ Implementation Inheritance)
Initialization	Setting the initial attribute values of a new object.
In-Line Method	A mechanism that allows the compiler to replace calls to the method with an expansion of the method code.
Instance Name	An identifier associated with and designating an instance.
Instance Variable	A variable that contains a value specific to an object instance.
Instance	An object created by instantiating a class. An object is an instance of a class. A functional unit comprising an individual named entity having the attributes of a defined class and providing defined services.
Instantiation	Object creation. The creation of an instance of a specified class.
Integrability	The capability of a system of being used as part of another, bigger CCS system.
Integrated Project Support Environment	An environment that specifies the processes for systematically managing development projects to minimize costs, increase productivity, and build quality software products.
Interface Definition Language	When used in conjunction with an ORB, IDL statements describe the properties and operations of an object. IDL is used to specify the public interface of a CORBA object.
Intelligent Electronic Device	An IED is any device incorporating one or more processors, with the capability to receive, process or send, data / control from, or to, an external source.
Interface Inheritance	The construction of an interface by incremental modification of other interfaces. The IDL provides interface inheritance.
Interface Type	A type that is satisfied by any object (literally, by any value that identifies an object) that satisfies a particular interface. (→ Object Type)
Interface	**1.** A shared boundary between two functional units, defined by functional characteristics e.g.- common physical interconnection characteristics, signal characteristics or other characteristics as appropriate, and the provision of a declared collection of services. **2.** A description of a set of possible uses of an object. Specifically, an interface describes a set of potential requests in which an object can meaningfully participate. (→ also Object Interface, Principal Interface and Type Interface)
Interoperability	**1.** The ability of two systems to exchange services. **2.** The ability for two or more ORBs to cooperate to deliver requests to the proper object. Interoperating ORBs appear to a client to be a single ORB.

Invariant Relation	A relation that cannot be changed so long as it has instances.
IP	Internet Protocol -The TCP/IP standard protocol. IP defines the datagram that provides the basis of connectionless packet delivery. It includes control and error message protocol providing the equivalent functions to Network services, Layer 3, of the OSI Reference Model.
IPSE	→ Integrated Project Support Environment
ISO 9000 Standards	The International Organization for Standardization (ISO) issues the ISO-9000 guidelines for the selection and use of the series of standards on quality systems.
ISO	International Organization for Standardization
LAN	Local Area Network
Language Binding or Mapping	The means and conventions by which a programmer writing in a specific programming language accesses ORB capabilities.
Legacy System	A previously existing system or application.
Leveling	The process of grouping information or concepts at various levels of increasing detail. The top-most level is general in nature and each successive level adds more detail until all aspects of the given subject matter have been explained in detail.
Life-Cycle Service	The Object Life-Cycle Service provides operations for managing object creation, deletion, copying and equivalence. An Object Service Definition.
Link layer	Layer 2 of the standard ISO Communications machine.
Link	**1.** Connection between two processing entities. **2.** Relation between two objects (a concept).
Literal	A value that identifies an entity that is not an object. (→ Object Name)
LLC	Logical Link Control
Local Area Network	A communications network which typically covers the area within a building or small industrial complex.
Log	A record, a journal, of chronologically ordered data e.g. Events + Time Tags + Annotations.
LSDU	Link layer Service Data Unit
MAC	Media Access Control
Managed Object	Clients of System Management services, including the installation and activation service and the operational control service (dynamic behavior). These clients may be application objects, common facilities objects, or other object services. The term is used for

	compatibility with system management standards (the X/Open GDMO specification and ISO/IEC 10164 System Management Function, Parts 1 to 4). An Object Service Definition.
Mapping	**1.** A set of values having defined correspondence with the quantities, or values, of another set. **2.** A rule or process, the O-O equivalent of a mathematical function. Given an object of one set, a mapping applies its associative rules to return another set of objects. Member Function (→ Method)
MDI	Multiple display Interface
Message	The mechanism by which objects communicate. A message is sent by a client object to request the service provided by the server object.
Meta-Model	A model that defines other models.
Meta-Object	An object that represents a type, operation, class, method or object model entity that describes objects.
Meta-Type	A type whose instances are also types.
Method	1. In systems development, a cohesive set of rules, methods and principles used to guide the modelling and development of software systems. 2. Code that can be executed to perform a requested service. Methods associated with an object are structured into one or more programs.
Method Resolution	The selection of the method to perform a requested operation.
MIDL	Microsoft Interface Definition Language. IDL used by Microsoft Windows applications (→ Interface Definition Language)
MMS	Manufacturing Message Specification - [ISO 9506]
Multi-Agent System	A Multi-Agent System is a system composed of a great number of autonomous entities, named agents, having a collective behaviour that allows to obtain the desired function/service.
Multi-cast	A message placed onto the communication network intended for a limited set of recipients. A Multi-cast message will typically contain the sender's address and an address field defining a limited set of recipient's addresses.
Multiple Classification	Ability of an object to belong to more than one type.
Multiple Inheritance	The construction of a definition by incremental modification of more than one other definition.
NCA	→ Network Computing Architecture
NEO	It is thought that it stands for Network Objects, but its developers say it is not an acronym. OMA implementation by SunSoft Inc. for the Solaris operating systems. (→ Object Management Architecture)

Network Computing Architecture	One of the first specifications of a software architecture to build distributed applications. Initially developed by Hewlett-Packard, formed the core of DCE (→ Distributed Computing Environment)
NIDL	Network Interface Definition Language. IDL used by NCA(→ Interface Definition Language, → Network Computing Architecture)
OBAI	→ Object-Based Architecture for Integration
Object	A combination of a state and a set of methods that explicitly embodies an abstraction characterized by the behaviour or relevant requests. An object is an instance of a class. An object models a real world entity and is implemented as a computational entity that encapsulates state and operations (internally implemented as data and methods) and responds to requests for services. An object is a self-contained software package consisting of its own private information (data), its own private procedures (private methods), which manipulate the object's private data, and a public interface (public methods) for communicating with other objects.
Object Adapter	The ORB component that provides object reference, activation and state-related services to an object implementation. There may be different adapters provided for different implementations.
Object Attribute	A Field, or, a category or value of data that, together with other attributes, specify the services or data values related to the function and performance of an Object.
Object Broker	→ Object Request Broker
Object Creation	An event that causes an object to exist that is distinct from any other object.
Object Data Base Management System	These systems provide for long-term, reliable storage, retrieval and management of objects. Object Destruction An event that causes an object to cease to exist and its associated resources to become available for reuse. Object Identity (→ Handle)
Object Interface	A description of a set of possible uses of an object. Specifically, an interface describes a set of potential requests in which an object can meaningfully participate as a parameter. It is the union of the object's type interfaces.
Object Library/Repository	A central repository established expressly to support the identification and reuse of software components, especially classes and other software components.
Object Management Architecture	The generic architectural design proposed by the OMG. The CORBA specification is the first standard of technology for OMA
Object Management Group	A non-profit industry group dedicated to promoting object-oriented technology and the standardization of that technology.
Object Modeling Technique	An object-oriented systems development life cycle developed by General Electric. Now being integrated by its developer Rumbaugh

	into a new method named Unified Modeling Language.
Object Reference	A value that precisely identifies an object. Object references are never reused to identify another object.
Object Request Broker	Provides the means by which objects make and receive requests and responses. The middleware of distributed object computing that provides a means for objects to locate and activate other objects on a network, regardless of the processor or programming language used to develop and implement those objects.
Object Services	The basic functions provided for object lifecycle management and storage such as creation, deletion, activation, passivation, identification and location.
Object State	The current information about an object that determines its behaviour.
Object Type	A type the extension of which is a set of objects (literally, a set of values that identify objects). In other words, an object type is satisfied only by (values that identify) objects. (→ Interface Type)
Object Wrapper	The result of encapsulating a set of services provided by a non O-O application or program interface in order to treat the encapsulated application or interface as an object.
Object-Based Architecture for Integration	An architecture developed to facilitate legacy application migration to open systems, client/server and object-based computing. The primary function of OBAI is to allow new systems to be developed without having to abandon existing information systems and to allow the new systems to take advantage of the knowledge, information and data contained in the old systems.
Object-Based	A programming language or tool that supports the object concept of encapsulation, but not inheritance or polymorphism.
ObjectBroker	OMA's CORBA implementation by Digital Equipment Corporation for several operating systems, mainly Digital UNIX, OpenVMS and Windows NT. (→ Object Management Architecture). Sold to BEA Systems to form part of their BEA M3 product.
Object-Oriented Analysis	The process of specifying what a system does by identifying domain objects and defining the behaviour and relationships of those objects.
Object-Oriented Business Engineering	A framework and discipline used to effectively model business processes. It involves identifying business objects, processes, structures, rules, policies, organizational structure and authority, location and logistics, technology and applications. Its goal is to produce precise descriptive models of business objects that can be converted into reusable and easily modifiable software components.
Object-Oriented Design	The process of developing an implementation specification that incorporates the use of classes and objects. It encourages modelling the real world environment in terms of its entities and their interactions.

Object-Oriented	Any language, tool or method that focuses on modeling real world systems using the three pillars of objects encapsulation, inheritance and polymorphism.
ODBMS	→ Object Data Base Management System
ODP	→ Open Distributed Processing
OMA	→ Object Management Architecture
OMT	→ Object Modelling Technique
Oneway Request	A request in which the client does not wait for completion of the request, nor does it intend to accept results. Contrast with deferred synchronous request and synchronous request.
Ontology	An ontology is an explicit specification of the structure of a certain domain (e.g. e-commerce, sport, ...). In practical terms this includes a vocabulary (i.e. a list of logical constants and predicate symbols) for referring to the subject area, and a set of logical statements expressing the constraints existing in the domain and restricting the interpretation of the vocabulary. Ontologies therefore provide a vocabulary for representing and communicating knowledge.
OOBE	→ Object-Oriented Business Engineering
OOPL	→ Object-Oriented Programming Language
Open Distributed Processing	An standard from ISO in the area of open distributed systems. Used as acronym: ODP.
Open Software Foundation	Non-profit standardization organization dedicated to promote open software standards, i.e. OSF/Motif, OSF/DCE and OSF/1.
Operation	A service that can be requested. An operation has an associated signature, which may restrict which actual parameters are possible in a meaningful request.
Operation Name	A name used in a request to identify an operation.
ORB	→ Object Request Broker
ORB Core	The ORB component that moves a request from a client to the appropriate adapter for the target object.
Orbix	Iona Technologies implementation of CORBA.
OS	Operating System
OSF	→ Open Software Foundation
OSF/1	UNIX-like operating system based on a microkernel architecture. (→ Open Software Foundation)
Overloaded Operation	Multiple methods of the same name, each having a unique signature. This allows the methods of the same name to be invoked with various argument types.

Paradigm	A broad framework for thinking about and perceiving reality. A theoretical, philosophical model composed of identifiable theories, laws and generalizations used in defining and solving problems.
Parallel Processing	The simultaneous execution or computation of two or more programs or operations.
Parameter Passing Mode	Describes the direction of information flow for an operation parameter. The parameter passing modes are IN, OUT and INOUT.
Parameters	Parameters are data that define the behaviour of functions.
Parameterized Class	A class that allows users to declare member functions and data members of "Some Type," which can be used as a template for declaring specialized subclasses that supply the "Missing" types.
Participate	An object participates in a request when one or more actual parameters of the request identifies the object.
Partition	Decomposing a type into its disjoint subtypes.
Pattern	A pattern describes a problem, a solution to a problem, and when to apply the solution. Patterns may be categorized in several ways; by example as design patterns, business process patterns and analysis patterns. → Design Pattern
Persistent Object	An object that can survive the process or thread that created it. A persistent object exists until it is explicitly deleted.
Physical layer	Layer 1 of the ISO Reference Model.
Plug and Play	A type of component that needs little modification to be integrated into a system.
Pluggable Transport	A CORBA transport layer that can be added or eliminated in run-time.
Point to Point	A dedicated communication link between two nodes only.
Pointer	A variable that can hold a memory address of an object.
Polymorphic Operation	The same operation implemented differently by two or more types.
Polymorphism	The concept that two or more types of objects can respond to the same request in different ways.
Portable Object Adapter (POA)	An object adapter is the primary means for an object implementation to access ORB services such as object reference generation. An object adapter exports a public interface to the object implementation, and a private interface to the skeleton. It is built on a private ORB-dependent interface. The Portable Object Adapter offers functionality enough to build portable servers.
Post-Condition	A constraint that must hold true after the completion of an operation.

Pre-Condition	A constraint that must hold true before an operation is requested.
Presentation Layer	Layer 6 of the ISO Reference Model.
Principal Interface	The interface that describes all requests in which an object is meaningful.
Private	A scoping mechanism used to restrict access to class members so that other objects cannot them.
Property	An attribute, the value of which can be changed.
Protected	A scoping mechanism used to restrict access to class members.
Protection	The ability to restrict the clients for which a requested service will be performed.
Protocol	A set of rules governing the information transfer within a communications network. Protocols perform Data Link Control by defining frame format(s), timing, error correction and Handshaking.
Public	A scoping mechanism used to make member access available to other objects.
Query	An activity that involves selecting objects from implicitly or explicitly identified collections based on a specific predicate.
Rapid Prototyping	The iterative process of quickly developing a prototype of an application, usually with the aid of specific GUI-building tools. This process is used to help uncover unknown details of the system under consideration, and to build the system in small increments.
Redundancy	Refers to spare or duplicate functionality that allows a System to continue to operate without degradation of performance in the event of single failure. e.g. a blown fuse.
Reference Architecture	An architectural description for a family of applications that describes functional components, connections, protocols, and control. A reference architecture generally consists of a partially-specified system composed of generic or abstract → Components, that are replaced by real components when the architecture is instantiated for an actual system.
Referential Integrity	The property that ensures that a handle which exists in the state associated with another object reliably identifies a single object.
Relation	An object type that associates two or more object types. A relation is how associations are formed between two or more objects.
Repository	Usually a central location used to store and organize software components and related definitions, rules, etc. (→ Object Library/Repository)
Request	An event consisting of an operation and zero or more actual parameters. A client issues a request to cause a service to be performed. Also associated with a request are the results that can be returned to the client. A message can be used to implement (carry) a

	request and/ or a result.
Requirements	A document describing what a software system does from a user's point of view. This document is input into the object-oriented analysis process, where it will be transformed into a much more precise description.
Responsibility	A service or group of services provided by an object; a responsibility embodies one or more of the purposes of an object.
Result	The information returned to the client, which can include values as well as status information, indicating that exceptional conditions were raised in attempting to perform the requested service.
Reuse	Reuse is the process of locating, understanding and incorporating existing knowledge, design and components into a new system. Reuse should occur at all levels of system development analysis, design, implementation, testing, documentation and user training.
Role	A sequence of activities performed by an agent. A portion of the social behaviour of an agent that is characterized by some specificity such as a goal, a set of attributes (for example responsibilities, permissions, activities, and protocols) or providing a functionality/service.
RTOS	Real-time Operating System
RTU	Remote Terminal Unit- typically a station in a SCADA system, an RTU acts as an interface between the communication network and the plant equipment.
Scalability	The ability of a system to grow without sacrificing performance. The Scalability is the criteria for a universal and cost effective CCS, taking into account the varying functionality, plant sizes and magnitude ranges.
Schema	A formal presentation with a defined set of symbols and rules that govern the formation of a representation using the symbols. There are many different kinds of schema, including object, event and activity schemas.
Security Domain	A subset of computational resources used to define a security policy.
SEI	→ Software Engineering Institute
Self-Reference	The ability for a method to identify the target object for which it was invoked. This notion is referred to by the key words "self " in Smalltalk and "this" in C++.
Semantics	The meaning -- the essence -- of the definition.
Server	The entity that provides a service that can be requested.
Server Class	A Server Class comprises of the external visible behaviour of an Application process.

Server Object	An object providing response to a request for a service. A given object might be a client for some requests and a server for other requests. (→ Client Object)
Service	A functional capability of a resource which can be modified by a sequence of service primitives.
Service Cycle	The complete process -and the time elapsed on it- from the issue of a request till the response to it.
Service Primitive	Abstract, implementation independent, representation of an interaction between the service user and the service provider.
Service	A computation that can be performed in response to a request.
Session	Layer 5 of the ISO Reference Model; provides the protocol that manages the construction of the logical message into the actual messages for transmission.
Signature	Defines the parameters of a given operation including their number order, data types and passing mode; the results, if any; and the possible outcomes (normal vs. exceptional) that might occur.
Single Inheritance	The construction of a definition by incremental modification of one definition. (→ Multiple Inheritance)
Skeleton	The object-interface-specific ORB component that assists an object adapter in passing requests to particular methods.
Software Engineering Institute	The SEI is located at Carnegie Mellon University. Originally a U.S. Air Force project, the SEI objective was to provide guidance to the military services when selecting capable software contractors. The resulting method for evaluating the strengths and weaknesses of contractors proved valuable for assessing other software organizations. Since the late 1980s, SEI has been addressing the maturity of software within commercially developed applications.
SOM	→ System Object Model
Specialization	A class x is a specialization of a class y if x is defined to directly or indirectly inherit from y.
State Consistency	Ensures that the state associated with an object conforms to the data model.
State	The information about the history of previous requests needed to determine the behaviour of future requests.
State Integrity	Requires that the state associated with an object is not corrupted by external events.
State Machine	A formal description of the functionality, responses, actions and re-actions, as a series of discrete, linked states, together with the criteria governing the transition from one state to another specific state.
State-Modifying Request	A request that by performing the service alters the results of future

	requests.
Static Binding	Binding that is performed prior to the actual issuing of a request. (→ Binding)
Static Invocation	Constructing a request at compile time.
Static Member Function	In C++, a function declared part of a class declaration. These functions can be invoked independent of any instances of the class.
Strong Typing	A language characteristic that requires an explicit type declaration for every value or expression. Strong typing makes static binding feasible.
Stub	A local procedure corresponding to a single operation that invokes that operation when called. Subclass (→ Subtype)
Sub–functions	Sub–functions are sub parts of a main function. A sub–function may be shared by more than one main function.
Subscribed data	Data that a Client has requested to be supplied on a regular basis, or when trigger condition(s) are satisfied.
Subtype	A specialized or specific object type.
Superclass	A class that provides its methods and attributes to another class derived from it via inheritance.
Switch	An active Communications Network device that facilitates the exchange of data between two devices, on different LAN segments, by dynamically connecting the two LAN segments together only as and when data transfer is required. Effectively multiplies the available bandwidth, allowing LAN segments to run in parallel.
Synchronous Request	A request in which the client object pauses to wait for completion of the request.
System Object Model/ Distributed System Object Model	SOM is a class library, and DSOM is an ORB. Both provided by IBM.
System	Within this document, "System" refers to CORBA Control Systems, other types of system will be identified by their prefix name.
Target Object	An object that receives a request. (Synonymous with Server Object)
Task	Often used as synonymous of Behaviour (see Behaviour) but with the significance of atomic part of the overall agent behaviour.
TCP/IP	Transmission Control Protocol/Internet Protocol A suite of protocols which together provide the functionality up to layer 4, of the ISO OSI Reference Model, without exact layer for layer correspondence. NB- Another protocol is required to sit above TCP/IP to provide the required functionality for layers 5, 6 & 7.
TAO	The ACE ORB. CORBA implementation of Washington University.

Test Equipment	Includes all tools and instruments used to simulate and verify the inputs/outputs of the operating environment of the CCS.
Transient Object	An object whose existence is limited by the lifetime of the process or thread that created it.
Transport Layer	Layer 4 of the ISO OSI Reference Model, acts as an intermediary between the Network and the User application.
Trigger Rule	A cause-and-effect relationship. When a certain event type occurs, a specific operation will be performed.
Type	A predicate (Boolean function) defined over values that can be used in a signature to restrict a possible parameter or characterize a possible result. Types classify objects according to a common interface; classes classify objects according to a common implementation.
Type Interface	Defines the requests in which instances of this type can meaningfully participate as a parameter. Example: If given document type and product type in which the interface to document type comprises "edit" and "print," and the interface to product type comprises "set price" and "check inventory," then the object interface of a particular document that is also a product comprises all four requests.
UCA 2.0	Utility Communications Architecture version 2.0 describes the concepts of standardised models for Power System Objects.
UML	→ Unified Modelling Language.
Unified Modelling Language (UML)	Standardised constructs and semantics for diagrams, including state machines, which are used to describe / specify the functionality of an IED, Object Model or a Process.
Unsolicited Data or Unsolicited Message	A Message or Data which is supplied to a Client, or Clients, from a Server without the Client(s) subscribing to that data or message, e.g. “Reset”, “Abort”, “Time”. Does not require a Connection to be established.
Use Case/Scenario	A description of systems functionality. A description of the sequence of actions that occurs when a user participates in a dialogue with a system. It describes the behaviour that is invoked by a system function.
Value-Dependent Operation	An operation in which the behaviour of the corresponding request depends upon which names are used to identify object parameters (if an object can have multiple names). Virtual Class (→ Abstract Class)
Virtual Member Function	A member function that can be overridden by derived classes in order to implement a general behaviour in a specific manner. Dynamic binding is used at run time to determine which of these functions to actually invoke.
Visibility	→ Accessibility

VM	Virtual Machine.
WAN	Wide Area Network - a communication network which typically covers a geographical area i.e. The network linking a Central Control Room to a number of Substations.
Weak Typing	A language characteristic that does not require an explicit type declaration for each value or expression. Weak typing makes dynamic binding feasible.
Workflow	The structured flow of information through the well-defined steps of a business process where tasks are performed on elements of the information. Typical workflows have both sequential and concurrent tasks
Wrapper	→ Object wrapper

This page has been intentionally left blank.

Part 2
OMG Technology

This page has been intentionally left blank.

4 The Object Management Group

4.1 Overview and Objectives

The Object Management Group (OMG) is a not-for-profit organization established in 1989 with the goal of developing vendor independent specifications to foster object technology by means of the creation of a software marketplace for object technology. It aims to reduce the complexity, lower the costs, and hasten the introduction of new software applications. By using OMG's object technology any organization can leverage previous efforts in building control systems.

The OMG is a *standardization organization* with an open, vendor-neutral, international, widely recognized and rapid standardization process based on demonstrated technology. Nowadays software vendors, developers and users working in various different fields belong to the about 800 members as well as universities and governmental institutions. Further it maintains a strong liaison with other organizations like ISO, ITU-T, W3C, TINA-C, and Meta Data Coalition.

OMG's object technology is the object technology of reference: CORBA, IDL, UML, MOF, XMI, MDA[4], etc. Their standards allow interoperability and portability of distributed object oriented applications of different vendors. They do not produce software or implementation guidelines; only specifications which are put together using ideas of OMG members who respond to Requests For Information (RFI) and Requests For Proposals (RFP). The strength of this approach comes from the fact that most of the major software companies interested in distributed object oriented development are among OMG members.

[4] Common Object Request Broker Architecture, Interface Definition Language, Unified Modeling Language, MetaObject Facility, XML-Based Metadata Interchange, Model Driven Architecture.

4.2 Structure and Activity

The OMG technical activity is organized around three major bodies:

- The Platform Technology Committee (PTC) is responsible for CORBA core technology.
- The Domain Technology Committee (DTC) is responsible for specifications in vertical domains.
- The Architecture Board (AB) is responsible for the OMA and the verification that new specifications are compliant with it as well as making sure that something similar is not subject of standardization in another subgroup.

The work is performed by a collection of working groups in the different areas; from core technology like the interoperability protocols to domain specific activities like data acquisition or financial security.

The OMG specification process is based on the submission of specifications from private organizations in accordance with the Request For Proposal (RFP) issued by the PTC or the DTC of the OMG. In some cases it makes sense to issue an Request For Information (RFI) and get some feedback about the special issues of a particular topic in order to be able to draft an RFP.

This means that the specification elaboration process is not done by a standardization committee (ISO C++ took more than eight years) but by an – usually – small group of OMG members *based on their own criteria and previous developments*. If a company possesses a technology that fits an RFP, the company can send the specification of that technology as a proposal to the OMG and it has a good chance of getting it approved as an OMG specification. This has been the case, e.g., for UML proposed by Rational or the Fault-Tolerance specification proposed by Sun.

If there are several proposals, the different submitters try to find a consensus and deliver a single, consolidated version, supported by all of them. This is usually called a Joint Revised Submission.

4.3 Overview of Specifications

The following list include the main areas of specifications issued by the OMG (the remainder of this document will emphasize especially on specifications relevant for control systems):

- The Common Object Request Broker Architecture (CORBA) specification allows application interoperability independent of platform, operating system, programming language as well as network and protocol. It includes a number of specifications: OMG Interface Definition Language (OMG IDL), the network protocols GIOP and IIOP, an infrastructure for server-side scalability termed the POA (for Portable Object Adapter), and the CORBA Component Model (CCM).
- The Object Management Architecture (OMA) defines a set of interfaces in a standardized way (by using OMG IDL) for standard objects that support CORBA applications. It includes the base-level CORBAservices, the CORBAfacilities, and a large and growing set of DomainFacilities.
- The Unified Modelling Language (UML) standardizes representation of object oriented analysis and design. It is a graphical language including Use Case and Activity diagrams for requirements gathering, Class and Object diagrams for design, and Package and Subsystem diagrams for deployment and lets architects and analysts visualize, specify, construct, and document applications in a standard way.
- The MetaObject Facility (MOF) standardizes a metamodel for object oriented analysis and design, and a repository. Because they are based on the MOF metamodel, UML models can be freely passed from tool to tool using XMI – without the commonality of definition provided by the MOF, this would not be practical.
- The Common Warehouse Metamodel (CWM) standardizes a basis for data modelling commonality within an enterprise, across databases and data stores.
- XML Metadata Interchange (XMI) allows MOF-compliant metamodels (and therefore models, since a model is just a special case of a metamodel) to be exchanged as XML datasets. Both, application models (in UML) and data models (in CWM), may be exchanged using XMI. In addition to allowing model exchange, XMI serves as a mapping from UML and CWM to XML.
- The Model Driven Architecture (MDA) unifies the Modelling and Middleware spaces and thus supports applications over their entire lifecycle from Analysis and Design, through implementation and deployment, to maintenance and evolution.

For modelling complex control systems especially the UML, the MOF, the CWM, and the XML specifications are relevant while for developing control system applications the main focus must be on the CORBA specification. The MDA specification allows an integration of model and application.

5 Overview of OMG Specifications

OMG specifications go beyond purely CORBA specifications but most of them are relevant and/or tightly coupled with the CORBA specification or the OMG Object Model as expressed in the Object Management Architecture (OMA).

Some of these specifications are contained in the main CORBA document: *Common Object Request Broker Architecture and Specification* [OMG 98b].

The OMG provides extensions and profiles over the base specifications as separate documents that specify the points of departure from the main specification. Of special importance for control systems engineering are the MinimumCORBA specification; the Real-Time CORBA specification and the Fault-Tolerant CORBA specification.

These specifications are grouped into the following collections:

- Modelling
- CORBA/IIOP
- Interface Definition Language and Mapping
- Specialized CORBA Specifications
- CORBA Embedded Intelligence Specifications
- CORBA Services
- CORBA Facilities
- Domain Specifications
- CORBA Security

5.1 The Object Management Architecture (OMA)

The Object Management Architecture (OMA) belongs to the main contributions of the OMG to the OO world. This is a specification for the construction of open distributed object systems based on brokering and a collection of predefined services [OMG 97].

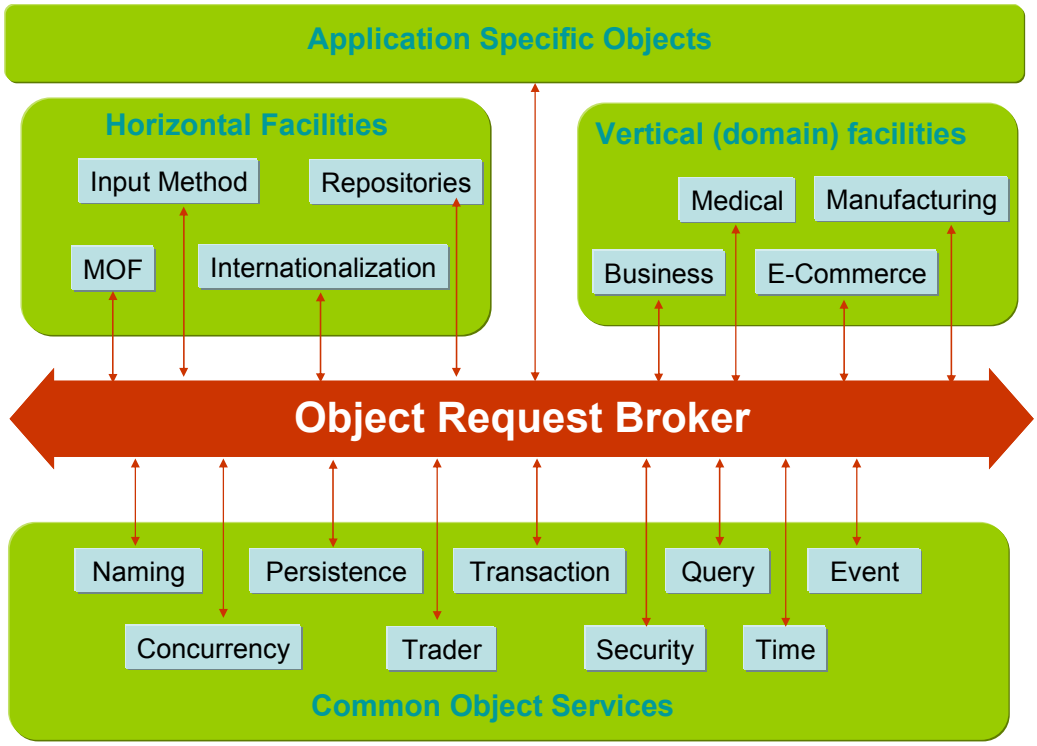


Figure 3: OMA Overview (main parts and an incomplete list of services)

It is a high-level vision of a complete distributed environment and consists of four components that can be roughly divided into two parts: system oriented components (Object Request Brokers and Object Services), and application oriented components (Application Objects and Common Facilities).

Figure 3 shows a basic overview of the main building-blocks (and an incomplete list of some examples) of the OMA:

- Object Request Broker (ORB): the central building block of the OMA that allows is the run-time integration vehicle for forwarding requests and responses between CORBA objects
- Common Object Services (now called CORBAservices): provide fundamental services that are nearly at systemlevel (e.g., Naming Service and Notification Service)
- horizontal CORBAfacilities: services that do not fit into a particular vertical market but are still at too high a level to be called a CORBAservice (the Printing Facility, the Secure Time Facility, the Internationalization Facility, and the Mobile Agent Facility)

- vertical CORBAfacilities (or Domain CORBAfacilities): provide standardized services for a particular field of application (e.g., Healthcare and Transportation)
- Application Objects: provide a particular service for a specific customer.

The Object Request Broker is the one of these parts which constitutes the foundation of the OMA and manages all communication between its components. It allows objects to interact in a heterogeneous, distributed environment, independent of the platforms on which these objects reside and techniques used to implement them. In performing its task it relies on Object Services which are responsible for general object management such as creating objects, access control, keeping track of relocated objects, etc. Common Facilities and Application Objects are the components closest to the end user, and in their functions they invoke services of the system components.

5.2 Modelling

5.2.1 Common Warehouse Metamodel (CWM™)

The main purpose of CWM is to enable easy interchange of warehouse and business intelligence metadata between warehouse tools, warehouse platforms and warehouse metadata repositories in distributed heterogeneous environments. CWM is based on three key industry standards:

- UML - Unified Modelling Language, an OMG modelling standard
- MOF - Meta Object Facility, an OMG metamodelling and metadata repository standard
- XMI - XML Metadata Interchange, an OMG metadata interchange standard.

These three standards form the core of the OMG metadata repository architecture. Al this is relevant to control engineering in the particular case of using databases. This is typical of large scale and continuous process control.

5.2.2 Meta-Object Facility (MOF™)

MOF is an extensible model driven integration framework for defining, manipulating and integrating metadata and data in a platform independent manner. MOF-based standards are in use for integrating tools, applications and data.

5.2.3 Software Process Engineering Metamodel (SPEM)

This specification presents the *Software Process Engineering Metamodel* (SPEM). This metamodel is used to describe a concrete software development process or a family of related software development processes. Process enactment is outside the scope of SPEM, although some examples of enactment are included for explanatory purposes.

5.2.4 Unified Modelling Language (UML)

A specification that defines a graphical language for visualizing, specifying, constructing, and documenting the artefacts of distributed object systems.

5.2.5 UML 1.4 with Action Semantics

Adds to UML the syntax and semantics of executable actions and procedures, including their run-time semantics. These semantics are contained within one Package, labelled Actions, which defines the various kinds of actions that may compose a procedure.

Using action semantics labelling it is possible to write UML models that can be used to generate 100% of final code in some execution contexts. This possibility is very interesting for model-based embedded control systems engineering.

5.2.6 UML Profile for CORBA

Provides a standard means for expressing the semantics of CORBA IDL using UML notation and support for expressing these semantics with UML tools.

5.2.7 UML Profile for Enterprise Application Integration

Provides a metadata interchange standard for information about accessing application interfaces. The goal is to simplify application integration by standardizing application metadata for invoking and translating application information.

5.2.8 UML Profile for Enterprise Distributed Object Computing

The vision of the EDOC Profile is to simplify the development of component based EDOC systems by means of a modeling framework, based on UML 1.4 and conforming to the OMG Model Driven Architecture.

5.2.9 UML Profile for Schedulability, Performance and Time

Specifies a UML profile that defines standard paradigms of use for modeling of *time-, schedulability-, and performance-related aspects* of real-time systems" that (1.) enable the construction of models that can be used to make quantitative predictions regarding these characteristics; (2.) facilitate communication of design intent between developers in a standard way; and (3.) enable interoperability between various analysis and design tools.

5.2.10 XML Metadata Interchange (XMI®)

XMI is a model driven XML Integration framework for defining, interchanging, manipulating and integrating XML data and objects. XMI-based standards are in use for integrating tools, repositories, applications and data warehouses.

5.2.11 XMI® Production of XML Schema

An XML schema provides a means by which an XML processor can validate the syntax and some of the semantics of an XML document. This specification provides rules by which a schema can be generated for any valid XMI-transmissible MOF-based metamodel.

5.3 CORBA/IIOP Specifications

5.3.1 Common Object Request Broker Architecture (CORBA/IIOP)

A specification of an architecture for middleware technology called Object Request Broker that provides interoperability among clients and servers distributed over a heterogeneous environment.

This is the common core of the CORBA specification. Optional parts of CORBA, such as mappings to particular programming languages, Real-time CORBA extensions, and the minimum CORBA profile for embedded systems are documented in other specifications that together comprise the complete CORBA specification.

5.3.2 Common Secure Interoperability (CSIv2)

Addresses the requirements of CORBA security for interoperable authentication, delegation, and privileges.

5.3.3 CORBA Component Model (CCM)

Specification of: a Component Implementation Definition Language (CIDL); the semantics of the CORBA Components Model (CCM); a Component Implementation Framework (CIF), which defines the

programming model for constructing component implementations; a container programming model describing how an Enterprise JavaBeans (EJB) component can be used by CORBA clients, including CORBA components; an architecture of the component container as seen by the container provider; how Component implementations may be packaged and deployed; and definitions of the XML DTDs used by the CORBA Components.

5.3.4 Fault Tolerance

Provides robust support for applications that require a high level of reliability, including applications that require more reliability than can be provided by a single backup server. The standard requires that there shall be no single point of failure.

5.3.5 Online Upgrades

Online Upgrades facilitates the safe and orderly upgrading of objects in a manner that is portable across systems and that is interoperable between systems. It is a first step towards a more general online upgrade capability. The specification aims to provide the ability to:

- Upgrade individual objects, where such upgrades change the implementation of the object but do not change the external interfaces of the object
- Pause an object, so that it can be upgraded, while allowing the object the opportunity to reach a safe and quiescent state
- Transfer state from an instance of the old implementation of the object to an instance of the new implementation of the object, with provision for such state transfers where the representations of the old state and the new state are different
- Resume service using an instance of the new implementation of the object without risk that messages will be lost, misordered or processed twice
- Allow client objects to continue to use a server object while remaining unaware that the server has been upgraded, and allow server objects to continue to serve a middle-tier client object that also acts as a server while remaining unaware that the client has been upgraded
- Address objects in such a way that a client can continue to use its existing object reference to access a server after it has been upgraded

- Rollback an upgrade, prior to the instance of the new implementation becoming operational, if some part of the upgrade fails
- Revert from an instance of the new implementation to an instance of the old implementation, if operation with the instance of the new implementation proves to be unsatisfactory

- Perform upgrades on small collections of objects by means of allowing the application to commit and rollback the upgrades explicitly.

5.4 IDL / Language Mapping Specifications

The language mappings provide the ability to access and implement CORBA objects in programs written in any of the mapped programming languages. Each one is aligned with a specific release of CORBA.

Supported (official) languages are:

- Ada
- C
- C++
- COBOL
- CORBA Scripting Language
- IDL to Java
- Java to IDL
- Lisp
- PL/1
- Python
- Smalltalk
- XML

The Java to IDL specification supports the inverse mapping of Java programming language constructs to OMG IDL constructs. It is aligned with CORBA 2.4. This is the only inverse specification due to the extremely concurrent models of CORBA and Distributed Java.

5.5 Specialized CORBA Specifications

5.5.1 Data Parallel Processing

Useful for high performance computing, this specification defines the architecture for data parallel programming in CORBA. The specification address data parallelism as opposed to other types of parallel processing that are already possible with distributed systems, namely pipeline parallelism and functional parallelism.

5.5.2 Dynamic Scheduling

Dynamic scheduling is widely employed in real-time and distributed real-time computing systems. This specification extends Real-time CORBA 1.0 to encompass these dynamic systems as well as static systems.

5.5.3 Lightweight Logging Service

This specification is primarily intended as an efficient, central facility inside an embedded or real-time environment to accept and manage logging records. These records are emitted from applications residing in the same environment and stored in a memory-only storage area owned and managed by the Lightweight Logging Service. The service was designed to be a mostly compatible subset of the Telecom Log Service, however, it differs in the way logging records are written to the log; or looked up and retrieved from the log. This service has a much wider application than just the software-defined radio domain. It will find its way into all areas of embedded systems, like machine control, onboard vehicle systems, etc., but also into ubiquitous computing devices like pocket computer and electronic organizers.

5.5.4 Minimum CORBA

A subset of CORBA designed for systems with limited resources.

5.5.5 Real-Time CORBA

Standard interfaces that meet Real-Time requirements by facilitating the end-to-end predictability of activities in the system and by providing support for the management of resources.

5.5.6 Unreliable Multicast

The purpose of MIOP (Unreliable Multicast Inter-ORB Protocol) is to provide a common mechanism to deliver GIOP request and fragment messages via multicast. The default transport specified for MIOP is IP Multicast 1 through UDP/IP 2 which will provide the ability to perform connectionless multicast. This requires that IDL operations will have one-way semantics.

This is useful for unreliable data dissemination (for example in most cases of monitoring).

5.6 CORBA Embedded Intelligence Specifications

This is a new subset of specifications that contains only one instance of major relevance for us.

5.6.1 Smart Transducers

Specifies a set of smart transducer interfaces that supports the following properties: (1.) the provision of a standardized set of functions, or services to a user in order to operate, configure and diagnose a generic transducer device; (2.) an encapsulation of the internal complexity of the generic smart-transducer hardware and software and the internal transducer failure modes to reduce the complexity at the system level; and (3.) a description of a canonical form of a communication service, or protocol.

5.7 CORBAservices Specifications

Services provide pre-built functionality for the construction of applications from CORBA building blocks. Most of them are useful in the context of controllers (in particular complex ones):

- 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
- Telecoms Log Service
- Time Service
- Trading Object Service
- Transaction Service

5.8 CORBAfacilities Specifications

Similar to services (but coarser):

- Internationalization and Time
- Mobile Agent Facility

5.9 OMG Domain Specifications

There are also many specifications in different domains that are more-or-less relevant to control systems engineering in that particular domain:

- Air Traffic Control
- Audio / Visual Streams
- Bibliographic Query Service
- Biomolecular Sequence Analysis (BSA)
- Clinical Image Access Service (CIAS)
- Clinical Observations Access Service (COAS)
- Computer Aided Design (CAD) Services
- CORBA-FTAM/FTP Interworking
- CORBA / TC Interworking and SCCP-Inter ORB Protocol
- Currency
- Data Acquisition from Industrial Systems (DAIS)
- Distributed Simulation Systems
- General Ledger
- Gene Expression
- Genomic Maps
- Interworking between CORBA and TMN Systems
- Laboratory Equipment Control Interface Specification (LECIS)
- Lexicon Query Service
- Macromolecular Structure
- Management of Event Domains
- Negotiation Facility
- Organizational Structure (OSF)
- Party Management Facility
- Person Identification Service (PIDS)
- Product Data Management (PDM) Enablers
- Public Key Infrastructure (PKI)
- Resource Access Decision (RAD)
- Surveillance Manager
- Task and Session
- Telecoms Log Service
- Telecom Service & Access Subscription (TSAS)
- Utility Management Systems (UMS) Data Access Facility
- Workflow Management Facility

Of particular importance are the specs in the manufacturing domain (DAIS, HDAIS).

5.10 CORBA Security Specifications

Another subset of specifications of relative importance (although growing due to interest in open distributed web-based control) is the security subset:

- Authorization Token Layer Acquisition Service (ATLAS)
- Common Secure Interoperability (CSIv2)
- Security Service
- Resource Access Decision Facility
- Security Domain Membership Management (SDMM)

6 Core CORBA technology

This chapter aims to give a short, practical introduction to the Object Management Group's Common Object Request Broker Architecture (CORBA). Although it is impossible to summarize a 1000+ pages specification to a few pages it should provide a good understanding of the basic mechanics of the architecture, give a rough overview of its components and provide the reader with some vocabulary used in the OMG document repository which is the most reliable resource of information about CORBA. At the end it also contains references to systems similar to CORBA, and some of the research connected with it.

For more information on CORBA take a look at the OMG specifications site[5]. The newsgroup `comp.object.corba` provides a good discussion forum.

6.1 The Common Object Request Broker (CORBA)

CORBA specifies a system which provides interoperability between objects in a heterogeneous, distributed environment and in a way transparent to the programmer. Its design is based on OMG Object Model.

6.1.1 The OMG Object Model

The OMG Object Model defines common object semantics for specifying the externally visible characteristics of objects in a standard and implementation-independent way. In this model *clients* request services from *objects* (which will also be called servers) through a well-defined interface. This interface is specified in OMG IDL *(Interface Definition Language)*. A client accesses an object by issuing a *request* to the object. The request is an event, and it carries information including an operation, the *object reference* of the service provider, and actual parameters (if any). The object reference is an object name that defines an object reliably.

[5] http://www.omg.org/library/specindx.htm

6.1.2 The Basic Mechanics of issuing a request

Figure 4 shows the main components of the ORB architecture and their interconnections:

The central component of CORBA is the *Object Request Broker* (ORB). It encompasses all of the communication infrastructure necessary to identify and locate objects, handle connection management and deliver data. In general, the ORB is not required to be a single component; it is simply defined by its interfaces. The ORB Core is the most crucial part of the Object Request Broker; it is responsible for communication of requests.

The basic functionality provided by the ORB consists of passing the requests from clients to the object implementations on which they are invoked. In order to make a request the client can communicate with the ORB Core through the IDL *stub* or through the *Dynamic Invocation Interface* (DII). The stub represents the mapping between the language of implementation of the client and the ORB core. Thus the client can be written in any language as long as the implementation of the ORB supports this mapping. The ORB Core then transfers the request to the object implementation which receives the request as an up-call through either an IDL skeleton, or a dynamic skeleton.

6.2 Overview of Architectural Components

The communication between the object implementation and the ORB core is effected by the *Object Adapter* (OA). It handles services such as generation and interpretation of object references, method invocation, security of interactions, object and implementation activation and deactivation, mapping references corresponding to object implementations and registration of implementations. It is expected that there will be many different special-purpose object adapters to fulfill the needs of specific systems (for example databases).

OMG specifies four policies in which the OA may handle object implementation activation: *Shared Server Policy*, in which multiple objects may be implemented in the same program, *Unshared Server Policy*, *Server-per-Method Policy*, in which a new server is started each time a request is received, and *Persistent Server Policy*. Only in the Persistent Server Policy is the object's implementation supposed to be constantly active (if it is not, a system exception results). If a request is invoked under any other policy the object will be activated by the OA in the policy specific way. In order to be able to do that, the OA needs to have access to information about the object's location and operating environment. The database containing this

information is called *Implementation Repository* and is a standard component of the CORBA architecture. The information is obtained from there by the OA at object activation. The Implementation Repository may also contain other information pertaining to the implementation of servers, such as debugging, version and administrative information.

The interfaces to objects can be specified in two ways: either in OMG IDL, or they can be added to *Interface Repository*, another component of the architecture. The *Dynamic Invocation Interface* allows the client to specify requests to objects whose definition and interface are unknown at the client's compile time. In order to use DII, the client has to compose a request (in a way common to all ORBs) including the object reference, the operation and a list of parameters. These specifications – of objects and services they provide – are retrieved from the Interface Repository, a database which provides persistent storage of object interface definitions. The Interface Repository also contains information about types of parameters, certain debugging information, etc.

A server side analogue to DII is the *Dynamic Skeleton Interface (DSI)*; with the use of this interface the operation is no longer accessed through an operation-specific skeleton, generated from an IDL interface specification, instead it is reached through an interface that provides access to the operation name and parameters (as in DII above the information can be retrieved from the Interface Repository). Thus DSI is a way to deliver requests from the ORB to an object implementation that does not have compile-time knowledge of the object it is implementing. Although it seems at the first glance that this situation doesn't happen very often, in reality DSI is an answer to interactive software development tools based on interpreters and debuggers. It can also be used to provide inter-ORB interoperability which will be discussed in the next section.

6.2.1 Interoperability

There are many different ORB products currently available; this diversity is very wholesome since it allows the vendors to gear their products towards the specific needs of their operational environment. It also creates the need for different ORBs to interoperate. Furthermore, there are distributed and/or client/server systems which are not CORBA-compliant and there is a growing need for providing interoperability between those systems and CORBA. In order to answer those needs OMG has formulated the ORB interoperability architecture.

Implementational differences are not the only barrier that separates objects; other reasons might include strict enforcement of security, or providing a protected testing environment for a product under development. In order to provide a fully interoperable environment all those differences have to be taken into account. This is why CORBA 2.0 introduces the higher-level concept of a domain, which roughly denotes a set of objects which for some reason, be it implementational or administrative, are separated from some all other objects. Thus, objects from different domains need some *bridging mechanism* (mapping between domains) in order to interact. Furthermore, this bridging mechanism should be flexible enough to accommodate both the scenarios where very little or no translation is needed (as in crossing different administrative domains within the same ORB), but efficiency is an issue, and scenarios which can be less efficient, but need to provide general access to ORB. This is critical in some control applications where isolation while keeping interoperability is necessary to achieve the necessary levels of predictability, safety and security.

The interoperability approaches can be most generally divided into *immediate* and *mediated* bridging. With mediated bridging interacting elements of one domain are transformed at the boundary of each domain between the internal form specific to this domain and some other form mutually agreed on by the domains. This common form could be either standard (specified by OMG, for example IIOP), or a private agreement between the two parties. With immediate bridging elements of interaction are transformed directly between the internal form of one domain and the other. The second solution has potential to be much faster, but is the less general one; it should be therefore possible to use both. Furthermore, if the mediation is internal to one execution environment (for example TCP/IP) it is known as a "*full bridge*", otherwise if the execution environment of one ORB is different from the common protocol we say that each ORB is a "*half bridge*".

Bridges can be implemented both internally to an ORB (say just crossing administrative boundaries), or in the layers above it. If they are implemented within an ORB they are called *in-line* bridges, otherwise they are called *request-level* bridges. The in-line bridges can be implemented through either requiring that the ORB provide certain additional services or through introducing additional stub and skeleton code. Interacting through the request-level bridges goes roughly like that: the client ORB "pretends" that the bridge and the server ORB are parts of the object implementation and issues a request to this object through the DSI

(remember, DSI needn't know the specification of its object at compile time). The DSI, in cooperation with the bridge, translates the request to a form which will be understood by the server ORB and invokes it through DII of the server ORB; the results (if any) are passed back via a similar route. Naturally, in order to perform its function the bridge has to know something about the object; thus if either needs to have access to the Interface Repository, or be only an interface specific bridge, with the applicable interface specifications "hardwired" into it.

In order to make bridges possible it is necessary to specify some kind of standard transfer syntax. This function is fulfilled by General Inter-ORB Protocol (GIOP) defined by the OMG; it has been specifically defined to meet the needs of ORB-to-ORB interaction and is designed to work over any transport protocol that meets a minimal set of assumptions. Of course, versions of GIOP implemented using different transports will not necessarily be directly compatible; however their interaction will be made much more efficient.

Apart from defining the general transfer syntax, OMG also specified how it is going to be implemented using the TCP/IP transport and thus defined the Internet Inter-ORB Protocol (IIOP). In order to illustrate the relationship between GIOP and IIOP, OMG points out that it is the same as between IDL and its concrete mapping, for example C++ mapping. IIOP is designed to provide "out of the box" interoperability with other compatible ORBs (TCP/IP being the most popular vendor-independent transport layer). Further, IIOP can also be used as an intermediate layer between half-bridges and in addition to its interoperability functions, vendors can use it for internal ORB messaging (although this is not required, and is only a side-effect of its definition). The specification also makes provision for a set of environment-Specific Inter-ORB Protocols (ESIOPs). These protocols should be used for "out of the box" interoperability wherever implementations using their transport are popular.

6.3 CORBA IDL

OMG IDL (Interface Definition Language) is an implementation independent language used to specify CORBA object interfaces. It is now an ISO standard and has several interesting characteristics: it supports multiple-inheritance (not so common in OO technology); it is – obviously – strongly typed; it is independent of any particular language and/or compiler and can be mapped to many programming languages (some mappings are specified by the OMG and others are contributed

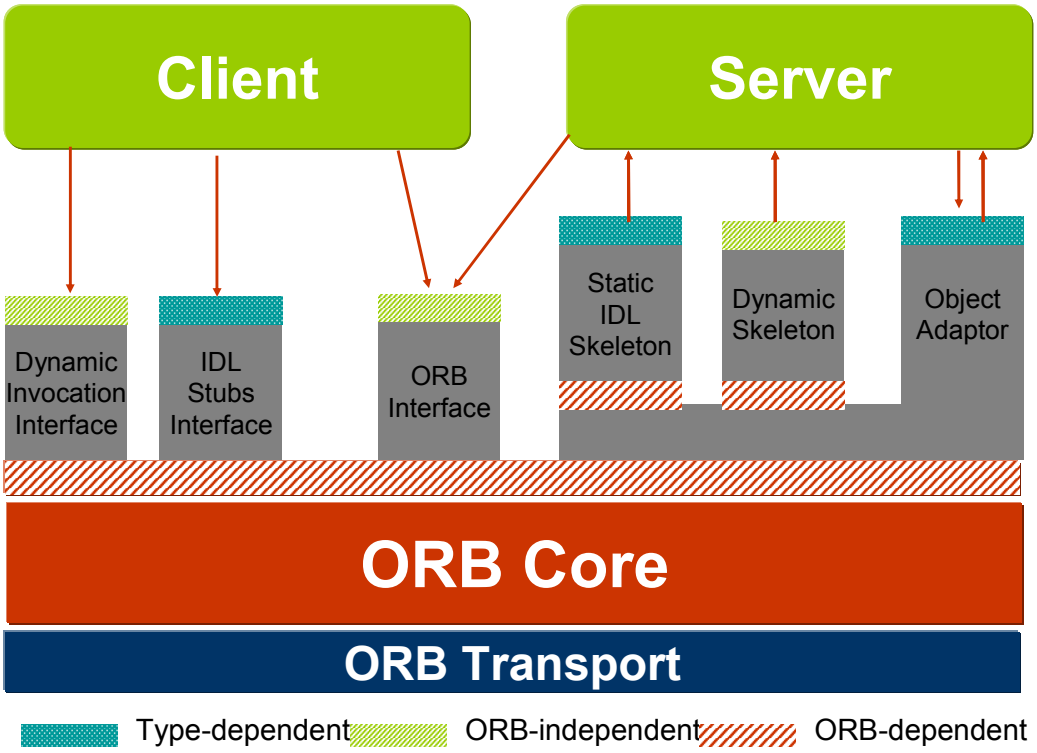


Figure 4: CORBA specifies a collection of interfaces used by clients and servers to use/provide different types of functionality.

specifications); it enables interoperability because it isolates interface from implementation.

6.4 OMA Middleware

The basic components in a running CORBA system are shown in Figure 4.

Object interfaces are specified in IDL which is language neutral and independent from operating system and hardware platform. The IDL compiler generates target code used to implement client stubs and server skeletons. CORBA defines how object invocations are mapped to a wire protocol, including argument encoding using a Common Data Representation (CDR) more effective that classical XDR. CORBA interoperability defines a generic protocol for interoperability (GIOP) between ORBs and a TCP/IP implementation of it, IIOP, that is the cornerstone of CORBA technology for interoperability.

As an example, a Java object running in a Netscape applet can invoke a method from an object written in good, old ANSI C and running on a remote PLC, almost transparently![6]

6.5 CORBA for Real-time Control

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 behaviour, fault tolerance and embeddability.

The Real-time platform task Force is addressing all these topics because they have focused their activities on real-time systems, and most real-time systems are also embedded and have some 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, organize workshops and other activities and involve 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 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**: Addresses the construction of CORBA applications on systems with little resources like embedded computers. This specification eliminates most dynamical interfaces that are not necessary in frozen applications (most embedded applications are ROMmed applications).

RT-CORBA standardizes the mechanisms for resource control (memory, processes, priorities, threads, protocols, bandwidth, etc.) and handling of

[6] But both the webserver and the PLC object wrapper must run on the same host due to Java applets security policies!

priorities in a distributed sense (for example forwarding client priorities to the server).

Using these mechanisms, the Real-time CORBA developer can control: Request time-outs, resource allocation and sharing, priority control and propagation, priority inversion, method invocation blocking, routing, transport selection, etc.

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.

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.

6.6 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.

7 RT-CORBA

RT CORBA specifies additional mechanisms that CCS builders can employ to increase the control of resources and end-to-end predictability of distributed object applications. **¡Error! No se encuentra el origen de la referencia.** provides an overview of the extensions compared to traditional CORBA.

The following sections provide a description of the key entities and features of a real-time CORBA broker as addressed in the Real-Time CORBA specification (see [OMG 98b] and [OMG 01b]).

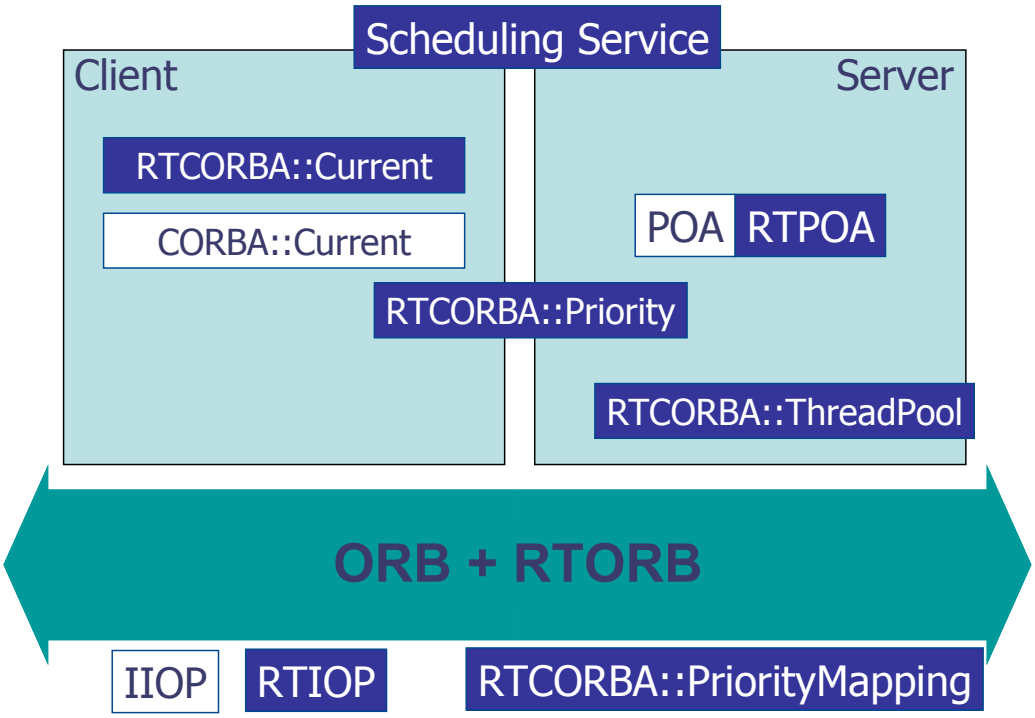


Figure 5: Real-time CORBA extensions provide strong control of resources to both clients and servers.

7.1 Fixed-priority Real-Time CORBA

The first real-time CORBA specification did not address the problem of dynamic scheduling. It only supports fixed-priority scheduling. Regarding end-to-end predictability the specification is defined to mean:

- Respecting thread priorities between clients and servers for the resolution of resource contention during the processing of CORBA invocations.
- Bounding of thread priority inversions during end-to-end processing.
- Bounding the latencies of operations invocations.

The specification relies in an extension of key CORBA entities and on the services provided by them.

7.1.1 RT-ORB

The real-time ORB is an extension of the CORBA::ORB interface that provides operations to create and destroy other constituents of a real-time ORB. There is also the possibility to configure the ORB on start-up to use a certain range of priorities during execution.

7.1.2 Real-time POA

The Portable Object Adapter for Real-time CORBA is defined in the module `RTPortableServer`. The Real-Time POA is a subtype of the CORBA `PortableServer::POA`. The POA for real-time has two different characteristics from that of CORBA.

- It should understand the policies specified in the real-time extension.
- It provides an additional set of operations to support object level priority settings. The Real-Time POA groups a set of operations designed to override the server declared priority on a per-object reference basis. Examples of these operations are `create_reference_with_priority` or `activate_object_with_priority` that allow setting an execution priority for CORBA objects or interoperable references.

7.1.3 CORBA Priorities

Different RTOS show different native thread priority schemes. As a result of RTOS heterogeneity different native priorities exist. Real-Time CORBA solves this problem defining a CORBA Priority which is valid for the entire system. As in the RTOS native priority schemes, CORBA Priority is

a set of integer values that map the native priority scheme of a specific RTOS to a uniform scheme which is accepted system-wide. Priority for real-time is defined in the RTCORBA module.

Real-Time CORBA also defines a PriorityMapping which is defined as an IDL native type. This is a programming language object (an instance of the `RTCORBA::PriotityMapping` class in C++) instead of a CORBA object. Consequently mappings to programming languages are also provided in the submission for Real-Time CORBA. Figure 6 shows a mapping of priorities between Real-time CORBA and a particular operating system. Notice that not all the priorities have to be mapped from CORBA to the OS. The priority mapping interface allows mapping CORBA priorities to the native operating systems and from the operating system to CORBA.

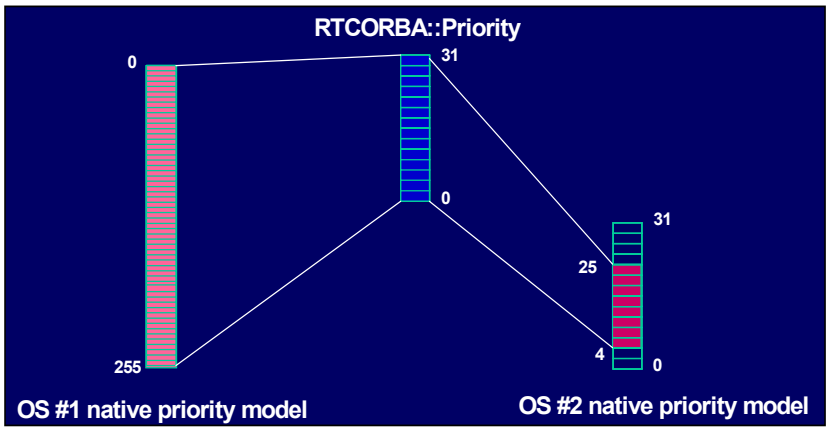


Figure 6: CORBA Priority mapping.

7.1.4 Real-time Current Interface

Real-Time CORBA derives the real-time Current interface from the `CORBA::Current` interface. The specific objective of Current for real-time is to obtain the CORBA Priority of the current thread. The `RTCORBA::Current` object contains a priority attribute which can be set and consulted. As a result of setting the priority attribute, the ORB sets the native priority of the thread to the value returned by `PriorityMapping::to_native`.

7.1.5 Real-time CORBA Priority Models and Transforms

In the context of a real-time distributed system there must be resources that allow applications to enforce eligibility of execution on remote objects. For this purpose, Real-Time CORBA supports two types of priority models to coordinate priorities across systems (see Table I).

Priority Model	Description
Client Propagated Priority Model	Priority is carried with the CORBA invocation and is used to ensure that all threads subsequently executing on the invocation run at the appropriate priority.
Server Declared Priority Model	In this model objects publish their CORBA priorities in the object references. This lets clients know the priority of invocation execution in the servant's code. The Real-Time CORBA Priority of an invocation needs not to be passed from client to server in an invocation.

Table I: Priority Models in Real-Time CORBA

Additionally, it is possible to override a Server Declared Priority on a per-object reference basis. The Real-Time CORBA POA interface provides four operations for doing this. The server priority can be changed at the time of object reference creation or at the time of object activation.

Priority Model Policies must be applied to POA objects at the time of creation. To let clients know the policy models supported by CORBA objects, the policies are propagated from servers to clients within IORs. The mechanism of propagation is defined in the Quality of Service framework of the CORBA Messaging specification. Figure 7 and Figure 8 show how the Client and Server Priority Models work.

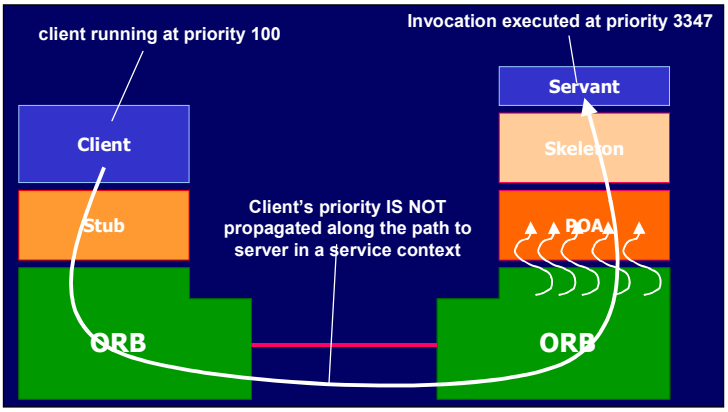


Figure 8: Server Declared Priority Model.

Transforms are user-defined Priority Transforms that modify the CORBA Priority associated with an invocation. The transforms take place when the invocation is processed by a server. Priority Transforms map Real-Time CORBA priorities to other priorities. These can be used to implement specific priority protocols. As `PriorityTransform` is an IDL native, language mappings are provided for different programming languages. There are two types of priority transforms inbound and outbound. Inbound transforms take place when an invocation is received by a server but before it is processed by a servant. Outbound transforms are applied to ongoing requests from a servant to other CORBA objects.

7.1.6 Synchronisation

Synchronisation is the satisfaction of restrictions that exist in the interleaving of actions of different processes or threads. One of the most difficult parts of building a concurrent application is dealing with process

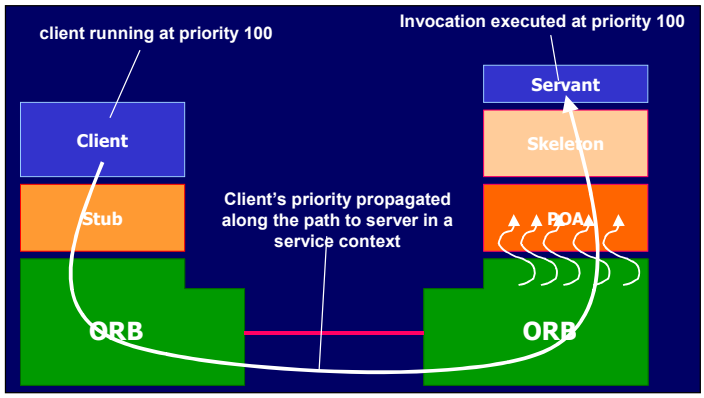


Figure 7: Client Propagated Priority Model.

or thread interaction. Threads and processes are not independent of each other and system's behaviour depends on their synchronisation and communication. For our purposes, communication can be understood as the passing of information between different processes or threads. This can be achieved by means of shared variables or message passing.

Real-Time CORBA provides a mutex interface to allow threads execute in regions protected by mutex objects. Objects of mutex type provide the degree of synchronization needed to protect a critical section (this is known as mutual exclusion and the *MutEx* word reflects that name). Real-Time CORBA provides two operations in the RTORB interface to create and destroy mutexes so the same mechanisms than the broker uses for synchronisation can be used by the application on top of the ORB. This helps to reduce priority inversion as all the mutexes will use the same protocol (e.g. a priority ceiling protocol).

7.1.7 Handling Concurrency

In the real world things happen in parallel. The term concurrency refers to an expression of the parallelism present in the world. In the computer, parallelism is achieved by concurrent programming. The term "concurrent" does not mean parallel but "potentially parallel". This is because concurrent programs or applications are formed by a set of sequential processes that are (logically) executed in parallel. Parallelism depends on how the collection of processes is executed.

- The execution is multiplexed in one processor.
- The execution is multiplexed in a multi-processor system where memory is shared (parallel computing).
- Execution is multiplexed in several processors and no memory is shared (distributed system).

Concurrency can be expressed either at the level of processes (programs) or inside the program (threads). In the latter case memory is shared by the different threads of the program. OS APIs to create threads are not always standard and a model of thread interface (a wrapper class) is useful when developing for several platforms.

Real-Time CORBA introduces the threadpool model which in a literal and straightforward definition is understood as a pool of threads. There are two possible styles for the threadpool: with or without lanes (Figure 9). A lane is a subset of the threadpool in which all the threads have the same `RTCORBA::Priority` value. Different lanes in the threadpool have different priority values.

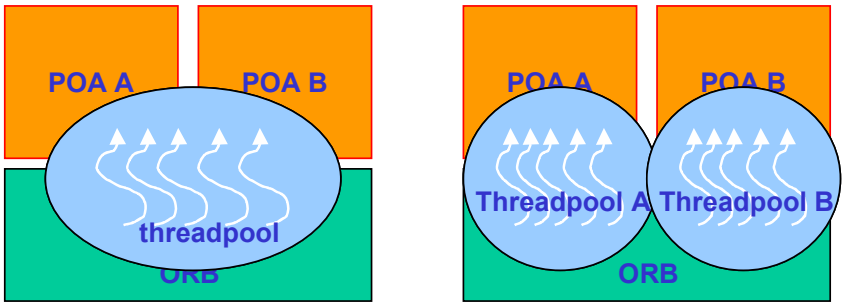


Figure 9: Threadpools without and with lanes.

The operations for the creation of threadpools allow to configure the stacksize and request buffering for the threadpool. `Stacksize` is important because operation arguments are stored in the thread stack and enough space must be allocated for this purpose. As with other policy types a threadpool can be associated with several POA at creation time by using a `ThreadpoolId`. Other interesting features can be set in the threadpool creation operations.

- **Static threads:** If lanes are not being used, it is the number of threads to be assigned to the threadpool. In a threadpool-with-lanes style it designates the number of threads in each lane.
- **Dynamic threads:** It is the number of threads that can be created dynamically and that are allocated either to the threadpool or to the lane.
- **Priority:** Without lanes it is the CORBA priority assigned to the static threads in the threadpool. In this case, dynamic threads can be created at the priority required to handle the invocation they were created for. If lanes are being used, the CORBA priority is assigned to all the threads (statically and dynamically allocated) in the lane.

In the styles with lanes, borrowing of threads from lower priority lanes can also be specified.

7.1.8 Handling of Connections

Predictability is a main issue in a real-time system. In a conventional ORB, binding does not occur until the first invocation on an object is made. In CORBA this is known as *implicit binding*. The binding is a source of overhead that puts a penalty on the first invocation. Real-Time CORBA anticipates this problem by making allowance for *explicit binding*. Client applications can control the time when the binding on an unbounded object reference is made by means of the `validate_connection` operation.

Real-Time CORBA also foresees the use of Priority Banded Connections. A Priority Banded Connection is a connection with an assigned set of PriorityBands. Priority bands assigned to a connection cannot be reconfigured during the lifetime of the connection and no priority may be covered more than once. As Figure 10 shows, it is also foreseen that non-contiguous ranges can be formed and that it is not necessary to cover all CORBA priorities.

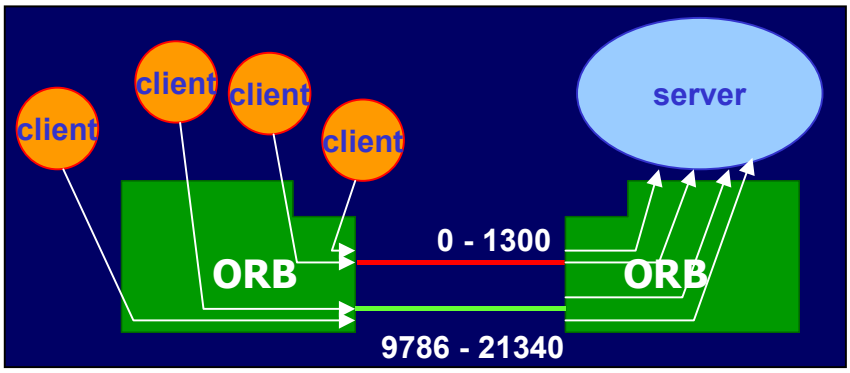


Figure 10: Priority Banded Connections.

The idea of banded connections is to allow clients communicate with servers using multiple connections reserved for invocations made at different CORBA priorities (Figure 10). If the priority of invocations is not respected by the transport it will become a source of priority inversion.

The connection is chosen depending on the target object priority model. In the case of the client propagated priority model the band is chosen using the priority specified by the client. If the ServerDeclared priority model is being used, its priority is published in the IOR and its value is used to select the band.

Banded connections are configured by clients and the policies are applied to the client side only. By default, the ORB provides a multiplexed connection for client/server connections (Figure 11). However, it is possible to request a private transport connection (Figure 11) by means of the `PrivateConnectionPolicy`.

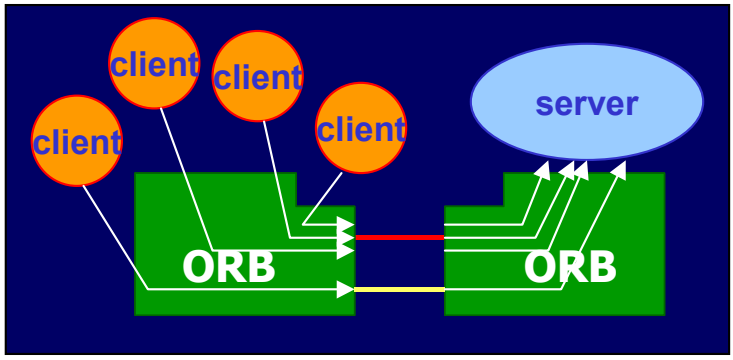


Figure 11: Private and Multiplexed Connections.

7.1.9 Invocation Timeout

Bounding the time in which a reply from a server must be obtained is a useful tool for development. Predictability can be improved if system developers know the upper bound of time an invocation will be blocked waiting for the server to answer. This functionality is achieved setting timeouts. Real-Time CORBA uses the `Messaging::RelativeRoundTripTimeoutPolicy` to set the timeout for the receiving of a reply to an invocation.

7.1.10 Protocol Configuration

Real-Time CORBA applications may find that best-effort QoS requirements do not meet their needs. Real-Time CORBA specifies an API to select and configure the underlying communication protocol. In CORBA, IOP instances contain an ORB protocol and a mapping to an underlying transport protocol. Real-Time CORBA defines an interface that allows applications to control protocol properties. It is possible to

configure protocol properties either at the client-side or at the server-side of an RTORB.

Protocols are specified by means of a protocol list which can be applied to POAs. The list indicates the protocols supported by a certain POA and the order of the protocols in the protocol list indicates the order of preference for the usage of protocols. The policy is client-exposed, meaning that it is encapsulated in IORs to be consulted by clients making invocations.

On the client-side, Real-Time CORBA defines a similar interface for the client protocol policy. The difference is that the policy is applied at the time of binding to an object reference. In the server side it was applied at the time of POA creation and the policy was propagated from client to server in the IORs.

7.1.11 Real-Time Scheduling Service

The real-time scheduling service is a tool that enforces a certain scheduling policy to be used across the whole real-time CORBA system. It is useful as specifying the appropriate configuration parameter in all parts of the system may be a complex task. The scheduling service provides a form of central repository from where a uniform scheduling policy for the whole system can be obtained by CORBA objects.

7.2 Dynamic-Scheduling Real-Time CORBA

In order to generalize the Real-time CORBA specification and meet the requirements of a greater field in real-time computing, another specification (dynamic scheduling see [OMG 01b]) has been created. The Real-time CORBA 2.0 specification tries to address static and dynamic distributed systems. Dynamic distributed systems are those in which the processing workload of the system is not well known before-hand or no bounds can be put to it so it is not possible to perform an off-line analysis of the system. The specification generalizes the concepts of distributed system scheduling and that of a distributable thread in order to allow the application or the ORB have control on the following features.

- Any scheduling discipline may be employed
- the scheduling parameter elements associated with the chosen discipline may be changed at any time during execution

- the schedulable entity is a distributable thread[7] that may span node boundaries, carrying its scheduling context among scheduler instances on those nodes.

In order to provide more control over the temporal behaviour of the application interacts with the scheduler than can use one or more scheduling disciplines such as Fixed Priority Scheduling, Earliest Deadline First, Least Laxity First, Maximize Accrued Utility. The specification provides IDL interfaces for all the above scheduling disciplines but it is possible to establish any other scheduling discipline. The goal of the scheduler is to determine how best to meet the schedule given a predicted use of system resource by the application in a certain instant of time. The specification provides a framework in the form of IDL interfaces that allow the development of portable schedulers.

7.2.1 Distributable Thread

In dynamic Real-time CORBA the notion of *activity* from Real-time CORBA 1.0 has been replaced by that of *distributable thread*. For dynamic systems and in order to achieve end-to-end timeliness a trans-node application behaviour must be enforced. This can be done by using the time and resources related parameters in a consistent system-wide manner for allocation of resources. A trans-node application behaviour abstraction is defined (the distributable thread) for this purpose. A distributable thread is a programming model abstraction. It is a thread that can execute operations in objects without regard for the physical node boundaries (Figure 12). The distributable thread is the schedulable entity in this specification.

[7] Much in the line of the DRTSJ.

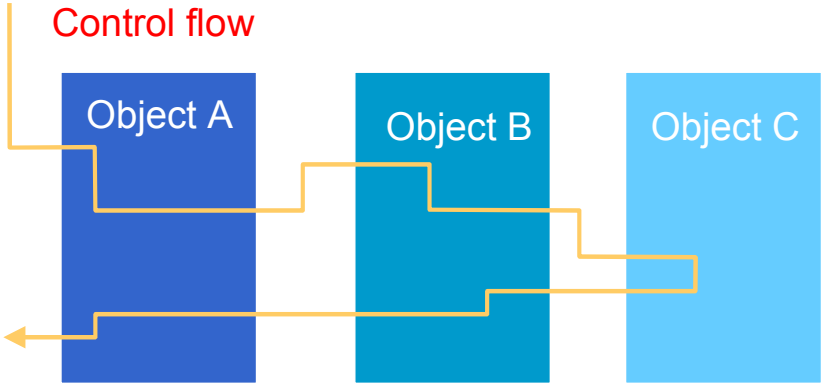


Figure 12: Control flow in a distributed processing systems.

Each distributable thread may have several execution parameters (e.g. priority, deadlines, utility functions, etc.) to specify the end-to-end timeliness to complete the set of sequential operations in objects that may reside in different physical nodes. The distributable thread transports scheduling information across the distributed system.

Distributable thread forking

Forking is the part of a concurrency model that deals with the creation of a new execution context. The specification allows for explicit forking of a distributable thread by the use of the spawn operation.

An example of forking is that of `oneway` invocations as the distributable thread making the invocation is not blocked waiting for the servant to process the invocation.

Scheduling Segments

A distributable thread consists of one or more *scheduling segments*. A scheduling segment represents a sequence of control flow related to a set of scheduling parameters. A scheduling segment has only one starting point and one ending point which may span processor boundaries.

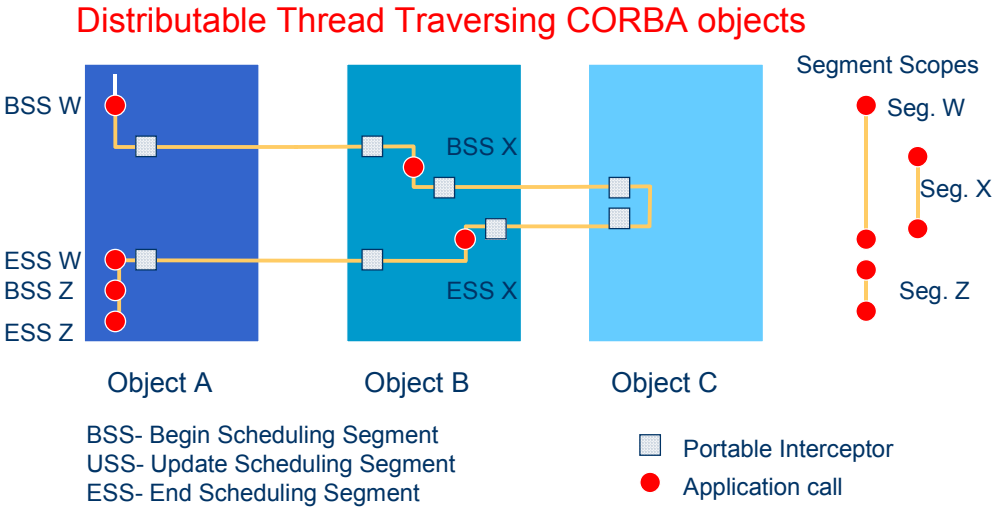


Figure 13: Distributable thread with nested scheduling segments.

Figure 13 shows an illustration of a distributable thread with nested scheduling segments. Where the segment X begins, the scheduling context of segment W is put aside and segment's X scheduling parameter elements are used for the distributable thread. When segment X ends, the scheduling parameter for segment W are used until it ends.

7.2.2 Scheduler

The scheduler is an extension to Real-time CORBA that manages the scheduling requirements and parameters of the applications that run on top of the broker. The scheduler decides on the order of execution of the applications on the distributed nodes of the CORBA system. To decide on the execution elegibility in the CORBA systems the scheduler is based on the following characteristics.

- The scheduler responds to application requests (to define scheduler elements) and in response to application actions (e.g. such invocations by using the Portable Interceptor interfaces).
- The scheduler uses the information provided by the application to decide on the eligibility of threads.
- The scheduler architecture is based upon the concept that the distributed system can be considered as a set of distributable threads.

- It is supposed that the schedulability of the system can be addressed by managing the allocation of resources to distributable threads.
- The distributable threads and the scheduler interact at specific scheduling points such as in transitions to new processors where scheduling information must be re-interpreted.
- The scheduler is a pluggable scheduler. If an ORB has a scheduler installed, all applications that run on that ORB use that scheduler.

Scheduling points

The scheduling points are the points in time and/or in code where the scheduler is run. This may result in a change of the current schedule. The defined scheduling points are shown below.

- Creation of a distributable thread.
- Termination or completion of a distributable thread.
- Beginning of a scheduling segment.
- Update of a scheduling segment.
- End of a scheduling segment.
- A CORBA operation invocation, specifically the request and reply interception points provided in the Portable Interceptor specification.
- Creation of a resource manager.
- Blocking on a request for a resource.
- Unblocking as result of the release of a resource.

Schedule-aware Resource

The specification allows the creation of schedule-aware resource via a *Resource Manager*. The resources can have scheduling information associated with them.

7.2.3 Tip of Advice about Real-Time CORBA

Originally CORBA was not been planned for hard real-time systems. Therefore several sources of non-determinism can be identified that could cause the system to miss its deadline: marshalling of parameters at the clients side, the protocol queues of the client, delays on the transport

media, the protocol queues on the server, dispatching of threads and requests, and unmarshalling of parameters at the server side.

Nevertheless there is no clear way of reducing complexity by subdividing a system into subsystems and retaining the temporal behaviour. Thus the implementer is left with the complexity of the whole system. Introducing composability while preserving end-to-end predictability is still an open issue. Especially analysis of a RT-System according to the real-time CORBA specifications is complicated by features like borrowing threads among threadpools.

8 Extensible Transports Framework

The GIOP-protocol can be mapped onto any connection oriented protocol that reliably delivers a stream of bytes, provides some reasonable notification for disorderly connection loss, and can be mapped onto the general connection model of TCP/IP. It includes seven message formats that provide support for opening and closing connections and transferring data as well as migration of dynamic objects.

This specification [OMG03a] targets at a standardized interface between the ORB and the transport layer in order to allow replacing the transport-plugin without changing the ORB itself.

As a brief overview this specification allows to establish, use (read or write), and close connections as well as listening for incoming connections. The transport protocol framework is responsible for the creation of the acceptor and connector objects which in turn provide service handlers to carry out communication through a given network protocol. A detailed discussion can be found in HRTC D2.2.

For control systems engineering it is necessary to define further functions (like providing a global time) or support for periodic transmission of state information with low jitter. On the other hand some requirements can be relaxed (e.g., reliable transport of messages).

A pluggable protocol framework can be composed of two different levels of components: The *pluggable message protocol* and the *pluggable transport protocol*.

The message layer is the ORB message layer, i.e. the GIOP message layer. It is not intended to introduce a new pluggable framework for the message layer since interoperability with other ORBs is likely to be lost otherwise.

The pluggable transport protocol is placed under the message protocol layer (the GIOP message layer) and this is the place where it is proposed to introduce protocol plugins for hard real-time communications. The transport protocol layer directly interacts with the network protocol and provides a way to hook a transport protocol to the broker with independency of its developer.

9 Fault-tolerant CORBA

9.1 Introduction

There are various kinds of applications with a need for fault tolerance beginning from large critical systems (such as air traffic control systems) to smaller critical systems (such as medical systems) and embedded applications (such as manufacturing control applications).

A standard that attempts to meet all of the requirements of this wide spectrum of applications might satisfy many needs only poorly, or might be too complex to implement. The Fault-tolerant CORBA specification [OMG 02b] therefore represents a number of compromises in order to support most of these systems.

Fault tolerance depends on entity redundancy, fault detection, and recovery. The entity redundancy by which this specification provides fault tolerance is the replication of objects. This strategy allows greater flexibility in configuration management of the number of replicas, and of their assignment to different hosts, compared to server replication. Replicated objects can invoke the methods of other replicated objects without regard to the physical location of those objects.

The standard supports a range of fault tolerance strategies, including request retry, redirection to an alternative server, passive (primary/backup) replication, and active replication which provides more rapid recovery from faults.

Applications that require the Fault Tolerance Infrastructure in order to control the creation of the application object replicas are supported as well as applications that control directly the creation of their own object replicas and applications that require the Fault Tolerance Infrastructure to maintain Strong Replica Consistency, both under normal conditions and under fault conditions.

Support for fault detection, notification, and analysis for the object replicas is supported. Thus allowing applications that require the Fault Tolerance Infrastructure in order to provide automatic checkpointing, logging and recovery from faults as well as applications that handle their own fault recovery.

The standard aims for minimal modifications to the application programs, and for transparency to replication and to faults and it defines minimal modifications to existing ORBs that allow non-replicated clients to derive fault tolerance benefits when they invoke replicated server objects.

9.2 Grouping Abstractions

In Fault-Tolerant CORBA there are two types of grouping abstractions; the object groups and the fault tolerance domains.

Object Groups

In order to achieve CORBA objects fault tolerance, several replicas of the object are created and grouped in an *Object Group*. The object group can be addressed as a whole by the use of an *Interoperable Object Group Reference* (IORG) which is exported by the server to be used by CORBA clients. The IOGR is created by the Replication Manager (see next sections). The clients invoke their requests on the object group and are processed by the object members of the group. Each individual replica has keeps its own IOR but the abstraction of the object group provides two advantages from the client point of view.

1. **Replication Transparency**: The client objects are not aware that the server objects are replicated.
2. **Failure Transparency**: The client objects are not aware of fault in the server replicas or of fault recovery.

Fault Tolerant Domains

FT CORBA also deals with large applications that have a need for fault tolerance. Such applications manage thousands of objects and span several locations so it is inappropriate to consider them as a single entity. To address this problem fault tolerant CORBA defines the concept of *fault tolerance domains* (Figure 14). A fault tolerance domain usually contains several hosts and many object groups being possible for a single host to support several fault tolerance domains.

All the objects and groups in a fault tolerance domain are managed by a single Replication Manager. Fault tolerance domains do not isolate objects in one domain from those of others. Objects in a domain can invoke to and be invoked by objects in other domains.

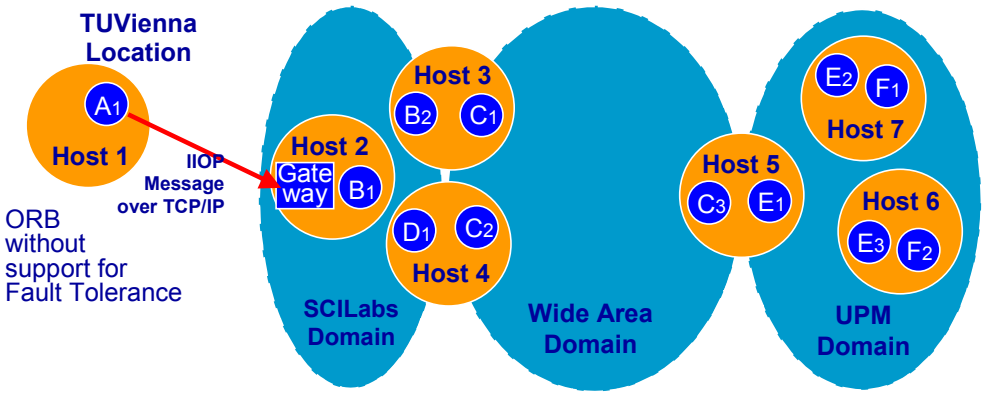


Figure 14: Fault tolerance domains.

Figure 14 shows an example of fault tolerance domains which are filled in light blue in the figure. Hosts are shown in orange and members of an object group are shown in dark blue. The members of object group B are denoted as B1, B2. The same notation is valid for groups A, D, C, E and F.

The fault tolerance domains allow applications to be arbitrary size, putting no limit to application scale. This is achieved by letting replications managers handle a smaller number of objects than that of the whole system.

Fault tolerance properties can be assigned either to object groups or to fault tolerance domains. Number of replicas or replication style (passive, warm passive or active) and other properties can be applied to all the object of a group, domain or to all the object groups of a specific type.

9.3 Architectural Overview

A fault tolerant CORBA system needs the infrastructure shown in Figure 15. A Replication Manager, Fault Notifier and Fault Detector object are implemented as CORBA objects. Logically, only a single instance of this objects exist in a fault tolerance domain but they are physically replicated for fault protection as all the other objects of the application. In the figure,

it can be seen that the Replication Manager inherits the Property Manager, Object Group Manager and Generic Factory interfaces.

The `PropertyManager` interface allows users to specify fault tolerance properties of object groups. Replication Management is controlled by the use of the `GenericFactory` and the `ObjectGroupManager` interfaces. The `GenericFactory` interface is able to create replicated objects on application demand. The `GenericFactory` is not used directly; it is the Replication Manager who will invoke the factories on the hosts where the replica is to be created. The `ObjectGroupManager` operations are designed to add, remove or control de location of member of an object group

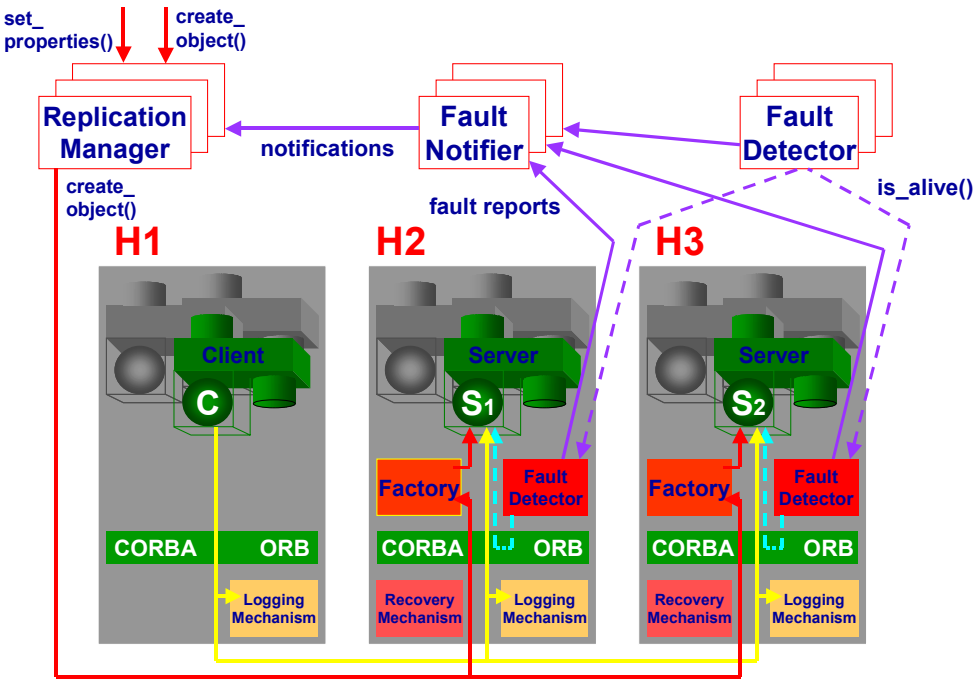


Figure 15: Architectural overview of a fault tolerant system.

The figure shows three hosts H1, H2 and H3. The client C on H1 is invoking a replicated server with two replicas S1 on host H2 and S2 on host H3. The Factory and Fault detector objects in each host are not replicated as occurs with the service objects on top of the figure.

All the application objects inherit a `PullMonitorable` interface that a Fault Detector invokes. It is a kind of watchdog invoking an `is_alive()`

operation. Faults detected by in-host Fault Detectors are notified to the Fault Notifier which passes the notifications to the Replication Manager.

In the case the passive or warm passive replication styles are used, only one member of an object group executes the requests and sends the replies. On a faulty condition, the Replication Manager can restart the primary member of the object group or can promote a backup member to primary member.

In the case of active replication all the members of an object group execute invocations independently and in the same order so as to keep the same state. When a fault occurs in one member the application continues with the results of other member without waiting for fault detection and recovery. There is a message handling mechanism that detects and suppresses all replies except one which is sent to client.

10 UML

The Unified Modelling Language (UML) is a textual and graphical notation used to quantify and formalize the understanding of systems. It can be used for business systems or computer systems that are an abstraction (i.e. simplified representation) of those systems as well as many other domains. More details than provided in this overview are available in the more than 1000 pages of the specification (see [OMG 03d]).

The UML has its roots in a branch of the computer software development industry known as Object Oriented Analysis and Design but also has applications beyond this field, especially – but not limited to – in the field of understanding and documenting business processes. Among the benefits of object-oriented analysis and design are:

- required changes are localized and unexpected interactions with other program modules are unlikely
- inheritance and polymorphism make OO systems more extensible, thus contributing to more rapid development
- object-based design is suitable for distributed, parallel or sequential implementation
- objects correspond more closely to the entities in the conceptual worlds of the designer and user, leading to greater seamlessness and traceability
- shared data areas are encapsulated, reducing the possibility of unexpected modifications or other update anomalies

Object-oriented analysis and design methods share the following basic steps although the details and the ordering of the steps vary quite a lot:

- find the ways that the system interacts with its environment (use cases)
- identify objects and their attribute and method names
- establish the relationships between objects
- establish the interface(s) of each object and exception handling
- implement and test the objects
- assemble and test systems

Experiences with systems modelled by average developers have demonstrated that data-driven approaches of modelling techniques usually lead to a few objects that concentrate the whole behaviour of the system and other objects that act similar to normalized database tables. In opposition to models created according to responsibility-driven approaches these models tend to have less reusable classes as a consequence. The UML authors promote a development process that is use-case driven, architecture centric, and iterative and incremental. The two basic principles of object orientation are very important and cannot be overstressed: encapsulation and inheritance.

For modelling in UML there are nine types of diagrams that address different aspects of an application:

- **class (package) diagrams**: describe the static structure of the system; package diagrams are a subset of class diagrams and organize elements of a system into related groups to minimize dependencies between packages
- **object diagrams**: describe the static structure of the system at a particular instant
- **use case diagrams**: model the functionality of the system using actors and use cases
- **sequence diagrams**: describe interactions among classes in terms of exchange of messages over time
- **collaboration diagrams**: represent interactions between objects as a series of sequenced messages thus describing both, the static structure and the dynamic behaviour, of a system
- **statechart diagrams**: describe the dynamic behaviour of a system in response to external stimuli (especially useful in modelling reactive objects whose states are triggered by specific events)
- **activity diagrams**: illustrate the dynamic nature of a system by modelling the flow of control from activity (operation on some class in the system that results in a change in the state of the system) to activity
- **component diagrams**: describe the organization of physical software components, including source code, run-time (binary) code, and executables.
- **deployment diagrams**: depict the physical resources in a system, including nodes, components, and connections.

One of the major strengths of UML is the possibility of extension in case that something else is required. This can be done with profiles that

provide particular extensions for a particular domain (e.g., real-time) or with free notes that are available for each construct.

In the "UML Profile for Schedulability, Performance & Time" are some definitions related to the domain of real-time systems:

- Modelling Resources
- Modelling Time
- Modelling Schedulability
- Modelling Performance
- Modelling Concurrency
- Modelling Processing

With the help of these building-blocks a control systems engineer can model important aspects of hard real-time systems.

11 CCM

In the CORBA Component Model (CCM) as defined in [OMG 01a] the component type is a specific, named collection of features that can be described by an IDL component definition or a corresponding structure in an Interface Repository (IR). It encapsulates its internal representation and implementation. Although the component specification includes standard frameworks for component implementation, these frameworks, and any assumptions that they might entail, are completely hidden from clients of the component.

There are two levels of components: basic and extended. Both are managed by component homes, but they differ in the capabilities they can offer. While basic components essentially provide a simple mechanism to "componentize" a regular CORBA object, extended components, on the other hand, provide a richer set of functionality.

In OMG Terminology a component may support a variety of surface features through which clients and other elements of an application environment may interact with a component. These surface features are called ports. The component model supports the following basic kinds of ports:

- **facets:** distinct named interfaces provided by the component for client interaction
- **receptacles:** named connection points that describe the components ability to use a reference supplied by some external agent
- **event sources:** named connection points that emit events of a specified type to one or more interested event consumers, or to an event channel
- **event sinks:** named connection points into which events of a specified type may be pushed
- **attributes:** named values exposed through `accessor` and `mutator` operations. Attributes are primarily intended to be used for component configuration, although they may be used in a variety of other ways.

While extended components may offer any type of port the basic components may only offer attributes. Thus it is very similar to an Enterprise Java Bean (EJB).

Using components allows defining typical patterns in an easy way. Among them are persistence or transactions of objects. Thus the programmer can focus on the "real" problem and does not have to deal with these problems.

12 Data-Distribution Service

Many real-time applications have a requirement to model some of their communication patterns as a pure data-centric exchange, where applications publish "data" which is then available to the remote applications that are interested in it. This specification (see [OMG 03b]) targets at solving the following problems:

Predictable distribution of data with minimal overhead is of primary concern to these real-time applications since it is not feasible to infinitely extend the needed resources.

The need to scale to hundreds or thousands of publishers and subscribers in a robust manner is an important requirement. This is actually not only a requirement of scalability but also a requirement of flexibility. Data-centric communications decouples senders from receivers; the less coupled the publishers and the subscribers are, the easier these extensions become.

The Data-Centric Publish-Subscribe (DCPS) model has become popular in many real-time applications. Compared to distributed shared memory which is a classic model that provides data-centric exchanges this model is easier to implement efficiently over a network and allows the required scalability and flexibility. It builds on the concept of a "global data space" that is accessible to all interested applications. Applications that want to contribute information to this data space declare their intent to become "Publishers." Similarly, applications that want to access portions of this data space declare their intent to become "Subscribers". Each time a Publisher posts new data into this "global data space", the middleware propagates the information to all interested Subscribers.

Another common need is a Data Local Reconstruction Layer (DLRL) that automatically reconstructs the data locally from the updates and allows the application to access the data 'as if' it were local. In that case, the middleware not only propagates the information to all interested subscribers but also updates a local copy of the information.

It defines a Platform Independent Model (PIM) for Publisher and Subscriber and the mapping rules to the Platform Specific Model (PSM). Since there is a separation between the publish sides and the subscribe sides an application process that only participates as a publisher can embed just what strictly relates to publication and vice versa.

13 Other Domain Specifications

13.1 Enhanced Views of Time

This specification (see [OMG 02a]) defines a format for representation of time and related terms (e.g., intervals) in CORBA as well as synchronization of clocks or periodic execution of tasks.

The basic format supports a granularity of 100 ns and allows a range of about ± 30000 years. The Clock Service makes no assumption about the accuracy of underlying time sources. It provides, however, means for characterizing the properties of each available time source, so that applications may select among them.

It allows to specify several characteristics for each clock:

- resolution: the granularity of the clock
- precision: the number of bits provided in the clock readout and their scaling
- stability: the ability of a clock to "tick" at a constant rate

In addition for a set of clocks the following parameters may be specified:

- offset: The difference between two clocks at a particular instant
- skew: the rate of change (first derivative) of the offset between two clocks
- drift: the rate of change of skew (second derivative of offset) between two clocks

Further for a ensemble of clocks that is synchronized with a reference the following parameters may be specified:

- coordination time scale: the time scale directly (through an external time source) or indirectly coordinated with
- coordination strata: an indication of "directness" of the coordination with the ultimate time source

- coordination source: the source of coordination.

The format supports more than ten time-formats (e.g., UTC, TAI, GPS) as well as time-displacement and functions for comparing timestamps and intervals against each other.

For special applications there are interfaces for controllable clocks, that can be paused, resumed, reset, or otherwise controlled and support for delayed execution.

This specification defines a lot of features that could be useful for control systems engineering. However this versatility could be a problem in embedded systems with limited resources. Since there is support for several time-formats and support for time-displacement there are several representations for one and the same instant. Choosing a time-format with leap-seconds (e.g., UTC) could lead to sporadic problems that are very hard to track since they occur rarely. Further the conversion of timestamps in systems without a hardware-floating-point-unit could require noticeable CPU time.

13.2 Smart Transducers

This specification (see [OMG 03c]) defines an abstract interface for a cluster of STs (smart transducers; small compact devices containing a sensor and/or actuator element, a microcontroller, and a communication controller) that allows to encapsulate the internal details and thus lower the complexity of the system.

It defines three different interfaces intended for different type of service levels: the time-critical real-time service (RS) interface, the non real-time diagnostic and management (DM) interface, and the configuration planning (CP) interface. Using simple and understandable orthogonal concepts is another key principle for reducing complexity.

These interfaces allow access to a distributed interface file system (IFS) that contains all values that should be visible to the outside while internal details are hidden by not mapping them into the IFS. Access to an ST is performed as read, write, or execute operation to the IFS. This means that also real-time data is available as state information in the IFS.

This specification also defines a format for timestamps that is especially suited for embedded systems with few resources because it consists only of an 8-byte integer value with a granularity of 2^{-24} and a precision field.

This allows a granularity of about 60 ns and also allows access to the full second with bit-shift-operations that are available on every microcontroller which makes synchronization with, e.g. GPS, very easy. By setting the precision (the number of valid bits in the time-stamp) setting to an appropriate value it is possible to specify timestamps in a cluster with imprecise clocks or bad synchronization.

Since this specification is especially targeting at small embedded systems (prototype implementations required an 8-byte controller with 4 kb ROM and 64 bytes of RAM) it presents some interesting approaches for equipping small (in terms of CPU power) sensors with CORBA.

Part 3
Software and Hardware

This page has been intentionally left blank.

14 ORBs

14.1 Implementations of CORBA ORBs

There are many implementations of CORBA ORBs currently available; they vary in the degree of CORBA compliance, quality of support, portability and availability of additional features. There are even fully compliant public domain implementations. Subsections below describe the most widely available ORBs.

14.1.1 TAO

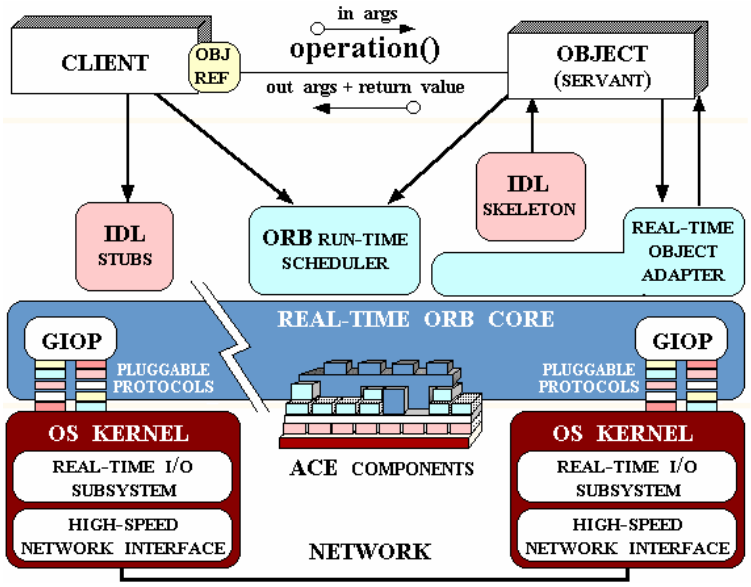


Figure 16: The ACE ORB (TAO)

TAO is the real-time ORB from the Distributed Object Computing (DOC) Group. The DOC is a distributed research consortium consisting of the Center for Distributed Object Computing in the Computer Science department at Washington University and the Laboratory for Distributed Object Computing in the Electrical and Computer Engineering department at the University of California, Irvine. In addition, the DOC Group also includes members at Siemens ZT in Munich, Germany, Bell Labs in Murray Hill, New Jersey, and OCI in St. Louis, MO. The purpose of the DOC group is to support advanced R&D on

distributed object computing middleware using an open source software development model.

TAO is based in the Adaptive Communications Environment (ACE) which is a network programming environment written in C++. This is also an open-software framework based on patterns for concurrent communication. The TAO acronym stands for "The ACE ORB" which reflects that TAO is based on ACE.

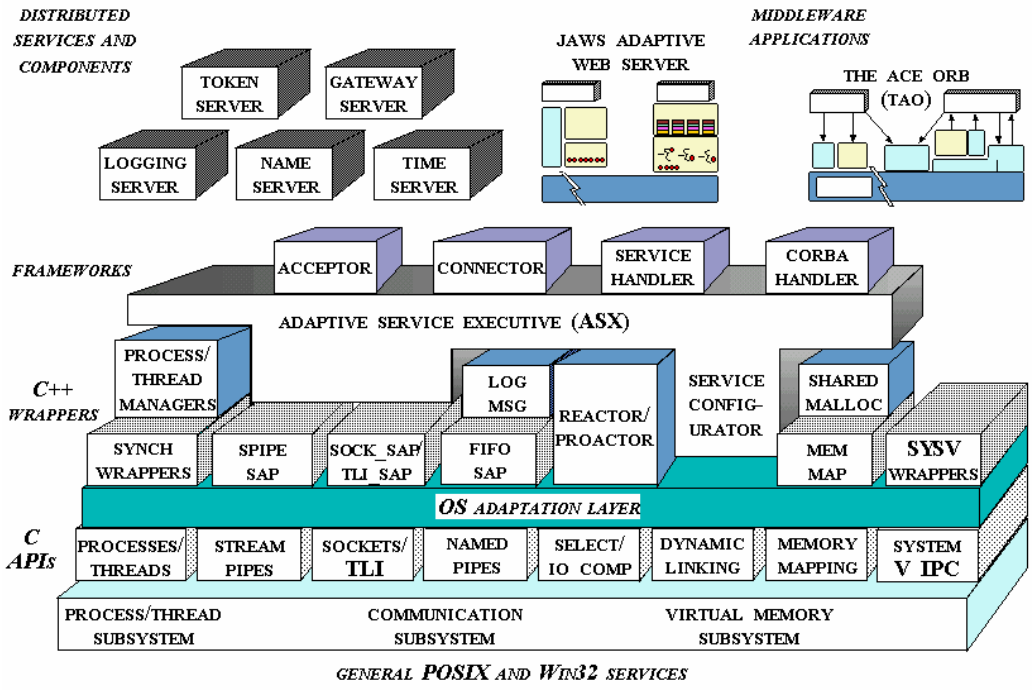


Figure 17: The Adaptive Communication Environment (ACE)

ACE was built with the objectives of providing portability, software quality, efficiency and predictability and ease of transition to higher-level middleware like TAO. ACE and TAO have been funded for over a decade by the DARPA Quorum program, NSF and several industrial sponsors (Lockheed Martin, Motorola, Microsoft, Nortel, Nokia, Boeing, Siemens, Raytheon and many more). ACE and TAO are commercially supported by Riverace, OCI and Prism Technologies on a open-software business model.

TAO is a real-time CORBA broker that is compliant with most of the services and features of the CORBA 3.x specification (including the Real-Time CORBA specification). TAO runs on Windows and on many UNIX systems and real-time operating systems. The main problem of TAO is its heavy footprint. In the minimum TAO configuration which lacks lots of features it has a footprint of 1,3

Mb (in a release version with no debugging information) which is an enormous footprint for any embedded system.

In addition, TAO provides many of the standard CORBA services as audio/video streaming service, Concurrency service, Event service, Lifecycle service, Naming service, Property service, Logging service, Persistent State service, Trading service, etc.

Regarding real-time there are other services as the Real-Time Event service, Load Balancing service and Scheduling service specially developed for real-time systems.

The pricing strategy for TAO is simple, it is a free broker. There are companies (Prismtech and OCI) that provide support and maintenance for TAO and access to the more up-to-date versions of the product.

14.1.2 Orbix

Orbix is the CORBA broker for IONA. IONA's story began in 1983 in the computer science department of Trinity College in Dublin. Chris Horn, Annrai O'Toole and Sean Baker spent much of the decade researching the ability to make computers, and the software that runs them, work together collaboratively.

In the years that followed, Horn, O'Toole and Baker continued the distributed computing research that would eventually become patented technology used by almost every Global 2000 company today. New funding came in from the European Union, the Irish Government, and in due course from Sun Microsystems of California (who would later sell their position after a 100-fold increase).

In 1993, IONA shipped its ORBIX product and left the Trinity College campus and began opening offices around the world. In 1995, the company opened its U.S. headquarters in Boston. Two years later, IONA "went public" on the NASDAQ exchange, in what was then the 5th largest software IPO ever. Currently, IONA employs more than 900 people in 30 offices world-wide, generating in excess of $150-million dollars in annual revenue and has been profitable every year since its foundation.

Still led by Chairman Chris Horn and CEO Barry Morris, IONA has become one of the leading software companies in the world. Over the past ten years, IONA has proven to more than 5,000 customers that it is capable of solving their integration problems and currently controls 40%.of the

CORBA market. The company's products have been used in a wide range of organisations including:

- Telecom
- e-commerce
- Manufacturing
- Financial
- Petroleum
- Research
- Defense
- Multimedia

Customers include Ford Motor Company, Southwest Airlines, Boeing, Deutsche Bahn AG, Lufthansa, Credit Suisse, ABN AMRO, Chicago Stock Exchange, Compaq, Silicon Graphics, Cisco Systems, Sun Microsystems, BSA Consulting, KPGM, Matra Systèmes & Information, Baxter Healthcare, Telefónica I+D, NOKIA Telecommunications, Nortel, etc.

Despite all the company software products and services it lacked products for embedded or real-time development. On February 2, 2001 the company announced the acquisition of Object Oriented Concepts, Inc. (OOC) to add embedded/real-time functionality to the Orbix family. This effort resulted in the ORBIX/E product (previously Orbacus/E from OOC) focused towards the embedded market. The acquisition of OOC brought to IONA more than 2000 new developer licenses, 350 new customers from OOC and more than 20 technical engineering experts.

IONA markets its products world-wide, primarily through a direct-sales organisation. They have their headquarters in Dublin and regional offices at Waltham, Santa Clara, Carlsbad, St John's, Denver, Alpharetta, Chicago, New York, Austin, Addison, Reston, Bellevue, Utrecht, Wokingham, Beijing, Espoo, Frankfurt, Munich, Karlsruhe, Hong Kong, Powai, Milan, Rome, Akasaka, Kangnam-ku, London, Paris, Perth, Madrid, Duebendorf, Stockholm, Sydney and Melbourne. IONA also maintains a partner program with solution and technology providers with benefits such as benefiting from the IONA brand recognition through co-marketing efforts.

Orbix 3 is the CORBA broker from IONA. It is compliant with the 2.1 CORBA specification and it is maintained by the company as a link with older CORBA systems. Orbix E2A is the current CORBA product sold by the company as an application server and heterogeneous system integration solution. It is mainly focused towards enterprise/business

applications and its features are compliant with CORBA 2.4. The company sells Orbix/E as its product for embedded systems.

Orbix/E is a lightweight CORBA v2.3 ORB. Lightweight means that only a subset of the CORBA specification has been implemented, allowing the product to present a small memory footprint. Orbix/E is not a real-time ORB but its footprint (100 kb for clients and 150 kb for servers in a low profile configuration) make it a good choice for some types of embedded and real-time applications.

Orbix/E is commercialised in a per-developer name, one license per platform and language fashion. There are also runtime license fees (as much as 3000$ per runtime license for small quantities).

14.1.3 Visibroker

Visibroker is the CORBA ORB from Borland. Borland Software Corporation is one of the leading providers of technology that helps Global 1000 companies develop, deploy, and integrate software applications. Delivering some of the best-in-class solutions dedicated to interoperability, Borland allows enterprises of all sizes to move into Web-based computing while continuing to leverage the benefits of legacy systems. With more than 1,100 employees worldwide and operations in more than 20 countries, Borland is a technology innovator that has been serving global customers with best-in-class technology solutions for more than 19 years.

Since 1983, Borland has been simplifying and speeding the process of application development. As a pioneer in this space, Borland launched one of the first PC development environments, Turbo Pascal, ®, which made possible the commercial development of PC applications. Through the years, Borland has anticipated the need of millions of software professionals around the world; the company has continually refined its technology to meet the evolving demands of business environments. In 1996, Borland began expanding its offerings to serve a broader range of customers: the company launched a Java™ development environment with the award-winning Borland JBuilder, ™ which is now the industry leader in the expansive Java development space. In 1997, the company acquired VisiGenic Software, enabling Borland to extend its application development expertise to enterprise application deployment. The year 2001 brought the launch of Borland Kylix, ™ a development environment

for the Linux® platform. Within six months, Kylix became the industry leader in this space.

Today, Borland continues to deliver on its mission to help customers embrace the future without abandoning the past. Borland carries out this mission by supporting the major software architectures used to develop, deploy, integrate and manage enterprise e-business applications. These include the Sun® J2EE™ platform and the Microsoft® .NET™ framework. Keeping pace with the rapid evolution of enterprise information technology, Borland has emerged as a key player in delivering development and deployment solutions for robust, standards-based Web Services and wireless applications.

Borland offers its Visibroker product in two different flavours; Borland Enterprise Server, Visibroker Edition and Visibroker-RT. The enterprise server is a unified, cost-effective software platform for deploying and managing a wide range of e-business applications and Web Services based on the Visibroker ORB. All Visibroker CORBA features are version 2.5 compliant.

Visibroker-RT is the CORBA solution from Borland for the development of distributed applications that incorporate embedded computers, including communications equipment, defence electronics, instrumentation, and process control systems. Visibroker-RT was developer by HighLander Engineering which was acquired by Borland in 2002. This solution from Borland is intended for the development of distributed real-time software that integrates with most real-time operating systems. Visibroker-RT offers either a complete embedded implementation of the CORBA specification or a Minimum CORBA subset. In both configurations the real-time CORBA extensions are supported. Visibroker also provides embedded implementations of the Naming service and of the Event service. Additionally, it also provides proprietary extensions for high availability. It is possible to have backup implementations for CORBA objects. If an object is no longer accessible requests automatically fail-over to a backup. Objects can also be visible on several networks simultaneously so communication can be re-routed in case of failure.

14.1.4 e*ORB

e*ORB is the real-time ORB from Prism Technologies. PrismTech, founded in 1992, is a privately held company, with both US and European

operations. The company is an acknowledged leader in the provision of standards-based middleware to a list of multinational customers world-wide, operating primarily in the Telecom, Defence, Financial Services and Manufacturing sectors. The customer list includes companies such as ABN Amro, Alcatel, AT&T, Bank of America, Boeing, Cisco Systems, Deutsche Bank, Ericsson, France Telecom, JP Morgan Chase, Lucent Technologies, Marconi, NEC, Nokia, Nortel Networks, Raytheon Systems, SBC, Siemens, Sonera, Sprint, Telcordia Technologies, UBS, US West and many more. PrismTech has rapidly grown its customer base since the launch of OpenFusion in 1999. Its market share has accelerated since the launch of the "Total CORBA Solution" product suite in 2001 that elevated PrismTech to a "full-service" CORBA vendor. PrismTech intends to become the market leading CORBA vendor by the end of 2003. During 2002/3 PrismTech also intends applying its business model to embrace J2EE and Web Services middleware opportunities.

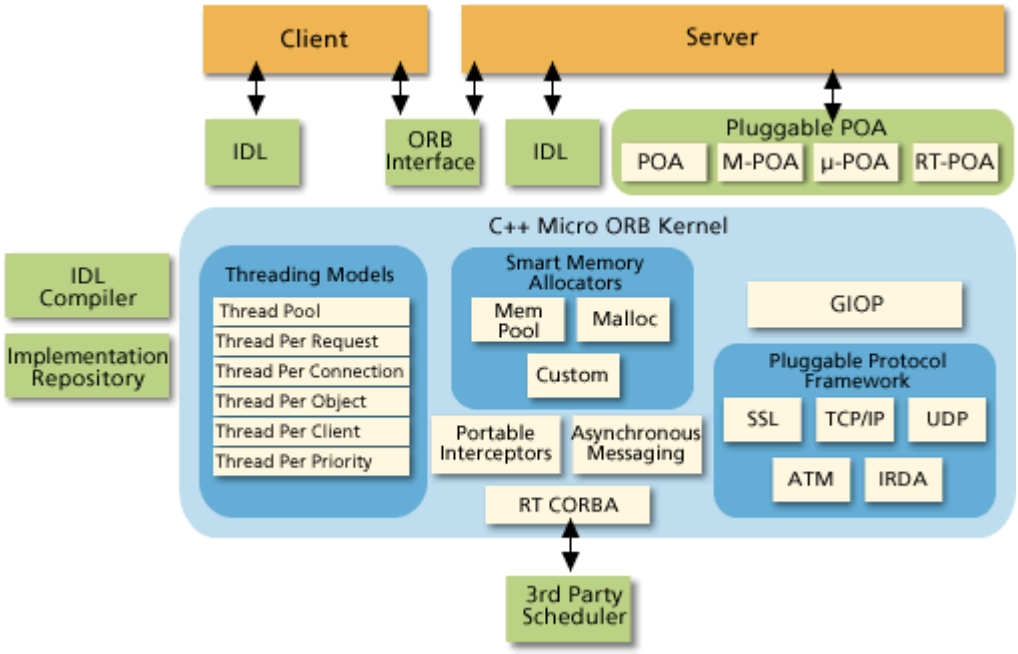


Figure 18: Open Fusion e*ORB architecture

e*ORB claims to be the fastest and smallest ORB in the world. It is a lightweight flexible and modular implementation of the Minimum CORBA specification v2.4 (fully compliant). Among its advantages is its availability in multiple configurations, capability of user-defined memory allocation and de-allocation, the extensible server-side threading

framework and the proprietary pluggable protocol framework (PPF). The ORB also provides a set of different POAs with different levels of functionality which can be used depending on the resource requirements of the target applications.

Additionally, e*ORB support logging, tracing and dumping for application debugging and analysis and also Naming, Telecom Logging, Notification and fault tolerance services.

The pricing strategy for e*ORB is based on licenses per developer/runtime model and both are per product or per project.

14.1.5 ORB Express from Objective Interface

ORBExpress is the real-time CORBA ORB from Objective Interface Systems (OIS). The company develops real-time, embedded and high performance software for use in products throughout telecom/datacom, military/aerospace, medical, process control and aerospace industries since 1989. OIS is one of the pioneers in the development and implementation of a real-time CORBA ORB and it is actively involved in the Object Management Group.

Among their customers there are CERN, National Lawrence Livermore Laboraty, Boeing, Nortel Networks, Ericsson, Daimler-Benz Aerospace, Lockheed Martin, etc.

OIS also claims (as PrismTech) to have the fastest ORB in the market. It is an ORB for the real-time and embedded market. ORB Express implements the Real-Time CORBA standard and provides additional features as plug-in transports. ORBExpress comes in three different flavours: ORBExpress RT, ORBExpress ST and ORBExpress GT.

ORBExpress RT is the flagship product from OIS and its features go beyond those of real-time CORBA. It is designed for hard and soft real-time applications and supports plug-in transports, transport quality of service and fail-over fault resilient connections.

ORBExpress ST is basically the high-performance core of the family of ORBs from OIS. It is a fully multithreaded ORB that always multiplexes connections in order to save resources. It also supports fail-over for connections.

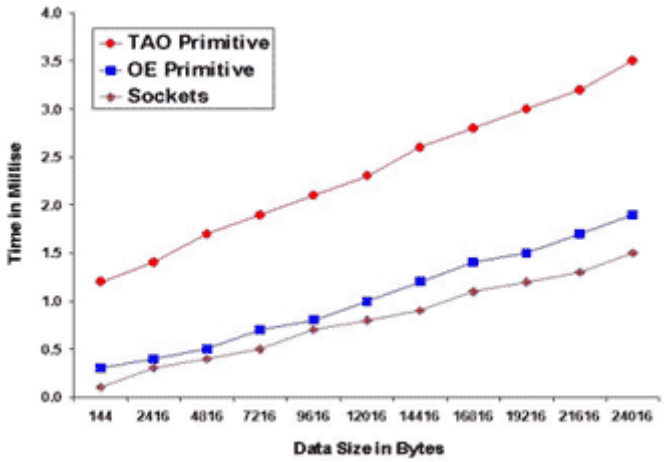


Figure 19: OE benchmark vs. TAO and raw sockets

ORBExpress GT provides more features to those of ORBExpress ST for the development of embedded systems. The ORB has a segmented architecture in which features can be used depending on the requirements of the target application and can be deployed in three different configurations (standard, small and tiny).

ORBExpress is licensed per developer name with no charge for runtimes. Volume discounts are available.

14.1.6 ICa from SCILabs Ingenieros

ICa is the real-time ORB from SCILabs Ingenieros. SCILabs was founded in 1998 with a clear interest in distributed control systems. The Integrated Control Architecture (ICA) ORB offers an extensible framework for architecture based development of product lines in the complex system control area.

ICa product family is available in two different formats. The Real-Time ICa for real-time systems and the Real-Time minimum ICa for real-time embedded systems. ICa is available for a wide range of operating systems

and hardware platforms. It has been successfully deployed in distributed control Remote Terminal Units (RTU), network device servers and time-triggered network devices for high predictability systems. ICa is compliant with the real-time CORBA specification and exhibits a small footprint (≈ 200 Kb in its minimal configuration).

ICa is licensed in a per developer/ per runtime model with volume discounts. Runtime licensed vary in a sliding scale and maintenance is based on a yearly fee.

14.2 Qualitative Feature Comparison

In the table below a qualitative comparison of the previous brokers has been made. Several features of the ORBs are compared; CORBA and Real-Time CORBA degree of compliance to the specification, suitability for embedded applications, availability for different platforms and operating systems, additional proprietary features and CORBA services, level of tech support, maintenance, and pricing strategy. Finally a global score is given for all the ORBs.

	CORBA	Real-Time CORBA	Embedded	Performance	Platforms & RTOS	Additional features	Technical Support	Pricing	Global score
TAO	★★★★	★★★★★	★	★	★★★★★	★★★★★	★★★	★★★★★	★★★★
Orbix/E	★★★	-	★★★★★	★★	★★★	★★★	★★	★★★	★★★
Visibroker-RT	★★★★★	★★★	-[8]	★★	★★★★	★★★★	★★	★	★★★
e*ORB	★★★★	★★★★	★★★★★	★★★★	★★★★	★★★	★★	★	★★★★
ORBExpress	★★★★	★★★★★	★★★★★	★★★★	★★★★	★★★	★★	★	★★★★★
ICa	★★★★	★★★★★	★★★★	★★★★	★★★	★★	★★	★★★	★★★

Notice that TAO which is a free research real-time ORB gets a very good qualification. This is because it is free, implements most of CORBA and real-time CORBA features and has lots of additional features (mainly CORBA services). Nevertheless, it must be understood that its suitability for real-time/embedded systems depends on the needs of the system. For instance the lowest memory footprint for TAO is 1,3 Mb and regarding performance a simple roundtrip request takes as much as 1,3 ms. Its

[8] There is no information on the minimum footprint for Visibroker

competitors footprint can be as small as 100 Kb and regarding performance they can be several times faster.

15 Design Tools

15.1 UML

UML stands for Unified Modelling Language. It helps specify, visualise, design, construct and deploy software-intensive artefacts. It was first introduced in 1997 and provides a standard notation to express a system's blueprint. UML allows to express either conceptual things as processes or functionality or concrete things as programming language classes or database schemas. UML started to form when Grady Booch, Ivar Jacobson, and James Rumbaugh began to adopt ideas from each other's methods (Booch, Jacobson's OOSE and Rumbaugh's OMT). Towards by the mid 1990s, they realised that their methods were already evolving toward each other independently and they decided it made sense to continue that evolution together rather than apart.

To express the semantics of CORBA a UML profile for CORBA specification was designed by the OMG. This has the advantage that an standardised set of UML extensions can be used among all stakeholders. As can be seen in the example of Figure 20, the UML profile for CORBA is greatly based upon the UML concept of Stereotype. In the UML metamodel, a Stereotype extends an element or elements of the metamodel. For example the stereotype <<CORBAException>> extends the UML metamodel Exception element. The aggregation of members into constructed types in CORBA types is always modelled as an aggregation Association with navigability away from the aggregate.

The UML profile for schedulability, performance and time enables the construction of models that can be used to make quantitative predictions regarding these characteristics. This is part of al larger initiative by the Real-Time Analysis and Design Group in the OMG to provide a solution for the modelling of real-time systems. In this case, the problem is tougher than that of the UML profile for CORBA. It covers different aspects of the

design and implementation of real-time systems; UML metamodel, real-time domain analysis, and schedulability analysis.

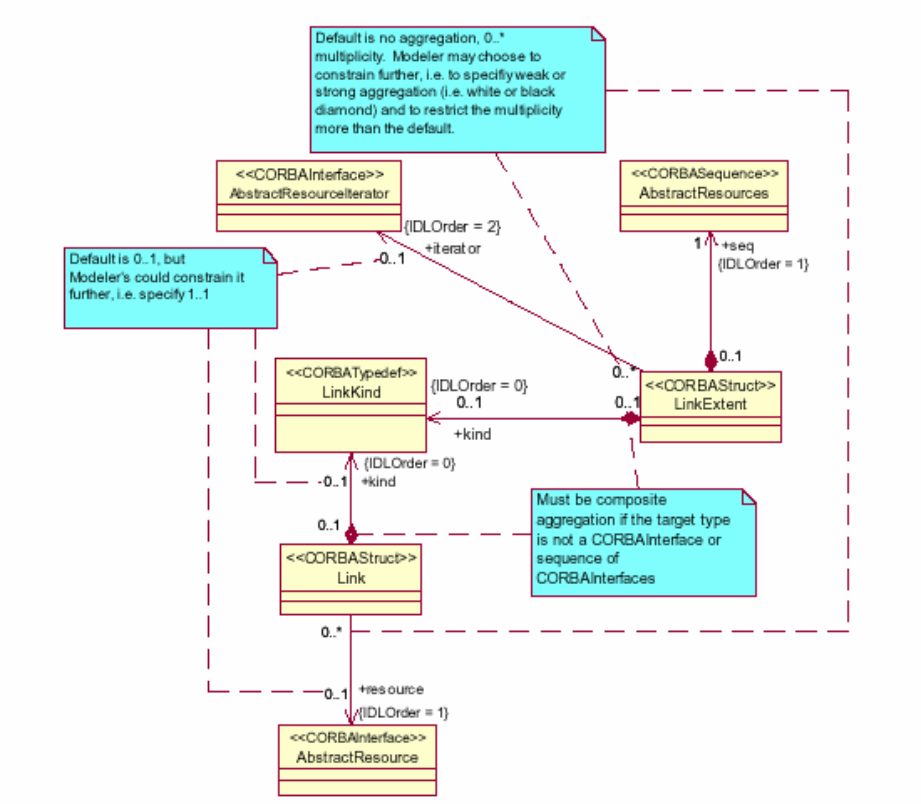


Figure 20: Example of the use of the CORBA UML Profile

The UML profile for schedulability performance and time is the result of the lack in UML of a quantifiable notion of time and resources which was an impediment for its use in broader range of real-time an embedded applications. The aim of the profile is to extend the UML metamodel in order to provide capabilities that allow designers to determine the schedulability of a piece of code before beginning to write a single line of it. This poses the need to model Quality Of Service aspects of the planned system.

The profile does not make any assumptions about real-time modelling concepts and leaves to the developer full control of UML modelling features to model a specific real-time system. This has been a consequence of the wide range of real-time and embedded systems and their varying demands; fault-tolerance, safety/mission critical, soft and hard real-time, etc.

The profile recognises the importance of modelling for analysis in this type of systems so predictability checks can be done automating the process with analysis tools. Different analysis methods focus on different aspects of a model. Analysis methods need a simplified view of the model in order to perform its task. The different perspectives of a model are called *analysis views*. An analysis view is a simplified version of the complete model and is extracted on the basis of a particular analysis or *domain viewpoint* representative of a specific analysis method (e.g. the "schedulability analysis viewpoint").

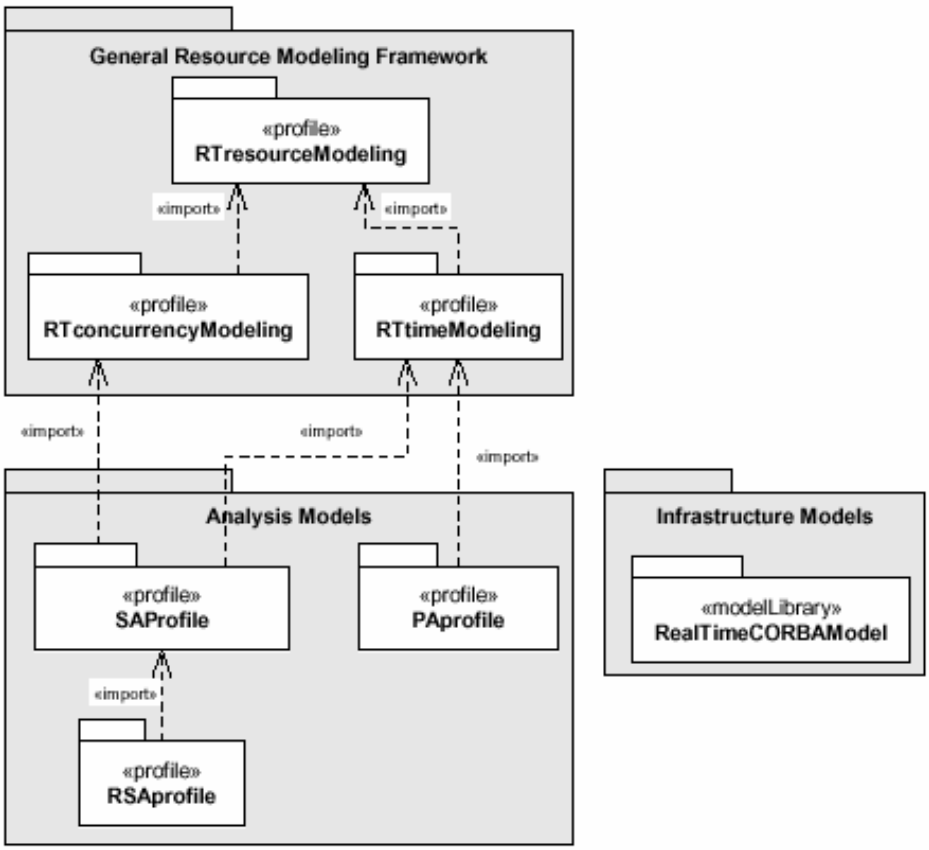


Figure 21: Structure of the UML Profile for performance, schedulability and time

The profile is organised so it is possible to make modelling for analysis or modelling process. Modelling for process means modelling for analysis plus synthesis. The profile defines a framework which is suitable for the modelling from the analysis or synthesis perspectives. The general structure of the framework is depicted in Figure 21.

The framework is modularised so only elements of the profile needed can be used. The profile is structured into the General Resource Modelling Framework and into the Analysis models.
The resource framework is partitioned in a set of sub-profiles for resource modelling. These include core resource concepts and specific sub-profiles for concurrency and time as these are basic requirements behind the UML profile.

The Analysis models package is split into three different sub-profiles for analysis which are based on the general resource modelling framework. One sub-profile is dedicated to performance analysis while another is for schedulability analysis. The schedulability analysis sub-profile is also further specialised for schedulability analysis of Real-Time CORBA applications.

Finally, the profile specification contains a model library with a high-level UML model of Real-Time CORBA. The intent of this model is to serve as the basis for more complex models where it is necessary to model parts of the system infrastructure (Real-Time CORBA in this case) as it is usual in these applications (e.g. fault tolerance).

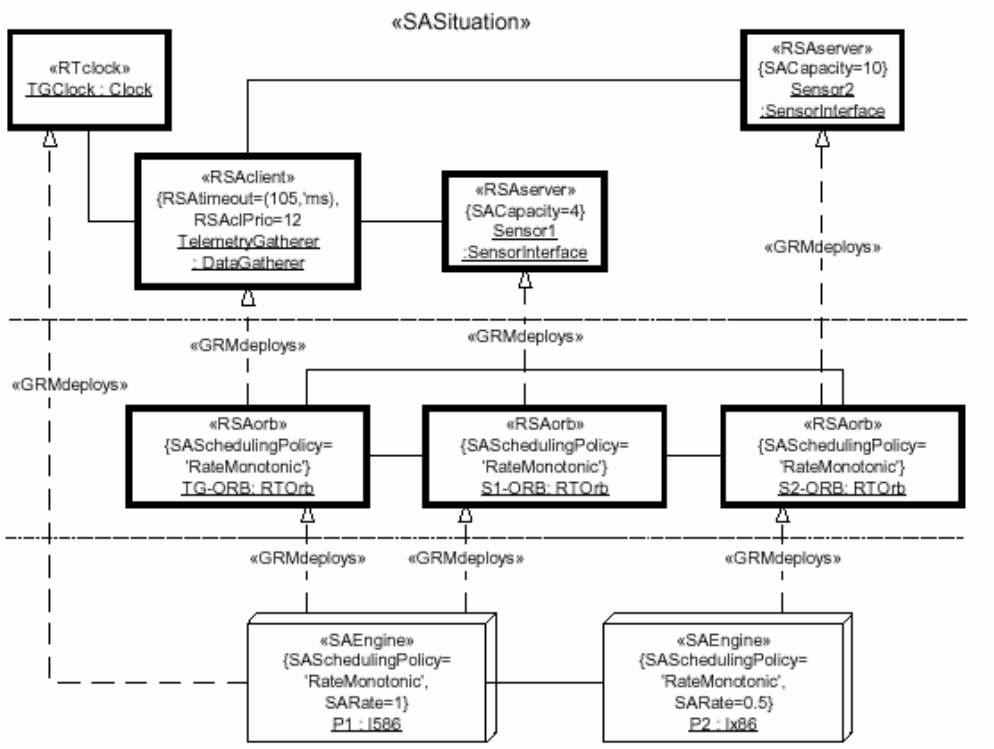


Figure 22: Example Real-Time CORBA Application

Figure 22shows an example on the use of the stereotypes for a real-time application with one client and two servers. The figure shows three layers separated by the horizontal lines. The layer at the top is a logical layer which is independent of the technology used. The middle layer is more related to the application realisation; in this case it is a real-time CORBA applications in which different ORBs are involved. The bottom layer is the hardware layer which shows the different processors of the system and the Real-Time CORBA functionality assigned to them.

15.2 IBM Rational

Rational is a family of products that use the industry's de facto standard language (UML) for the modelling of system software architecture and design models. Rational products range from model-driven design (IBM Rational Rose Real-Time), target-based component testing and runtime analysis (IBM Rational Test Real-Time and IBM Rational Purify Plus Real-Time) and life-cycle support (the Rational Suite Team Unifying Platform).

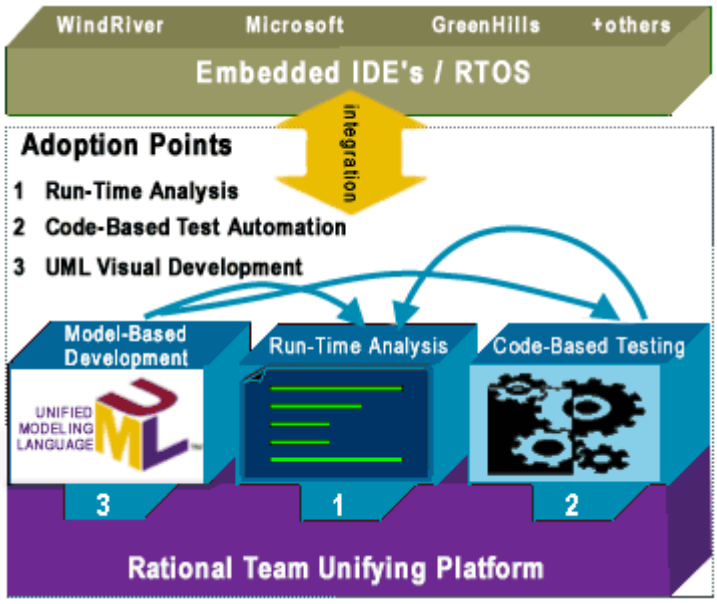


Figure 23: IBM Rational Product Family for real-time/embedded systems

15.2.1 IBM Rational Rose Real-Time

The tool is a complete lifecycle UML development environment targeted at real-time/embedded systems. This tools is specifically suited to deal with the problems of concurrency and distribution. Rose Real-Time provides from requirements capture through code generation, testing and debugging for real-time operating system targets.

15.2.2 IBM Rational Test Real-Time

Rational Test Real-Time is a solution for cross-platform real-time and embedded product testing. Test Real-Time allows to test, analyse and debug during development on host and target platforms and provides mechanisms for component and system testing as well as memory, performance and thread profiling. The tool also has the ability to perform code coverage analysis and runtime tracing.

15.3 I-Logix Rhapsody

Rhapsody from I-Logix is another family of tools targeted at real-time and embedded systems. These family of tools also follow a Model Driven Development approach as those from IBM Rational. The Rhapsody family is composed by the three tools below.

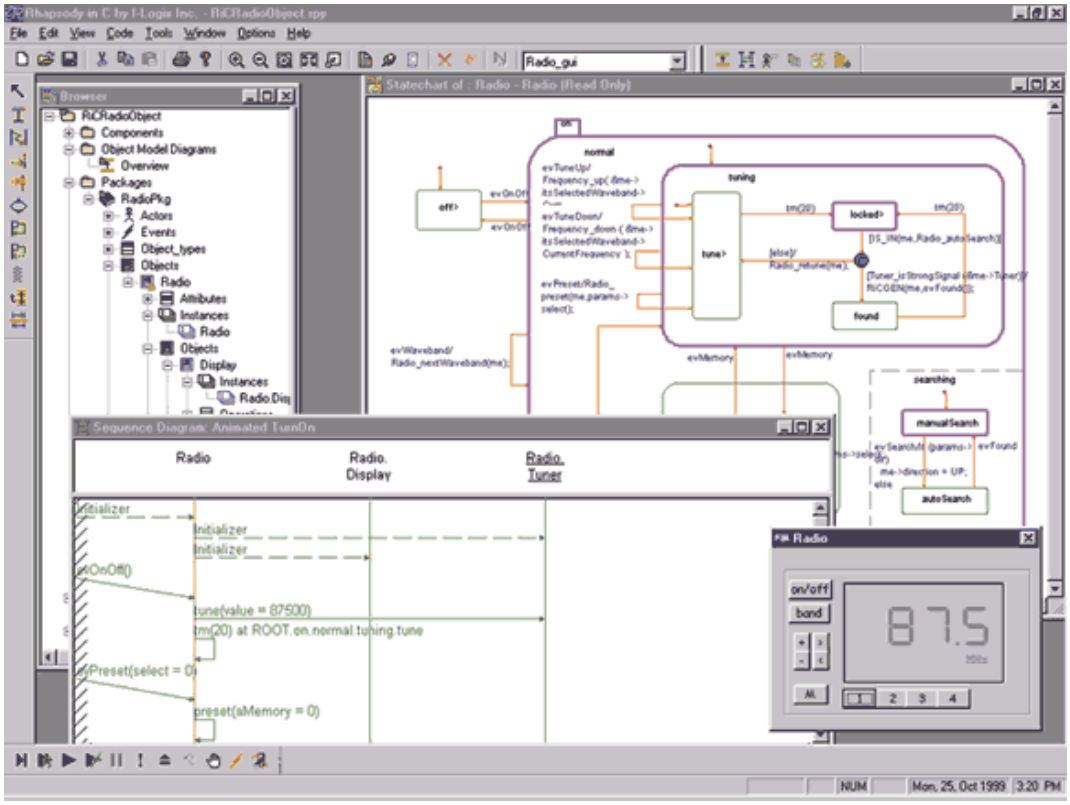


Figure 24: Design-level debugging during runtime using Rhapsody.

- **Rhapsody Architect**. This product is the base of the Rhapsody family. It is a MDD environment based on UML which performs requirements analysis, design and documentation of real-time embedded applications.
- **Rhapsody Designer**. The Designer enables executable validation of the model-based designs on the development host platform. The tool provides the features of the Architect package plus design-level debugging to prove behaviour and functionality and to validate analysis models.
- **Rhapsody Developer**. Rhapsody Developer is the flagship product of the family and encapsulates the functionality of the previous two tools. It also provides an environment for testing and deployment, and production code generation together with an execution platform for deployment in the target hardware.

15.4 Artisan Software Real-Time Studio

Artisan tools for real-time and embedded systems is one of the collaborators of the UML Profile for Schedulabilty, Perfomance and Time. The profile's proof of concept used the analysis tools from Tri-Pacific and TimeSys and made them interwork with the UML modelling tools from Artisan Software and IBM Rational.

Real-Time Studio is a multi-user suite of tools specifically suited for technical systems. The tools provide UML modelling with real-time extensions and design validations. Among the features there are the following.

- **State models simulation of system behaviour**.
- **Generation of test harnesses for behaviour verification**.
- **Front-panel simulation with the Altia Face Plate** (Graphics tools for simulation).

Real-Time Studio is able to animate sequence diagrams, simulate state models and integrate graphic panel displays from Altia. Real-Time Studio also provides CORBA support with the generation of IDL files for CORBA interfaces.

15.5 PrismTech OpenFusion CORBA Explorer

CORBA Explorer is basically a user interface for CORBA distributed computing systems. CORBA Explorer is formed by a set of four tools to explore CORBA systems.

- **Object Explorer**. Gives direct access to object whose interface is available in an interface repository. The object interface is exposed and operation can be invoked. Values can be assigned and return values can be inspected. It is a tool thought to explore object

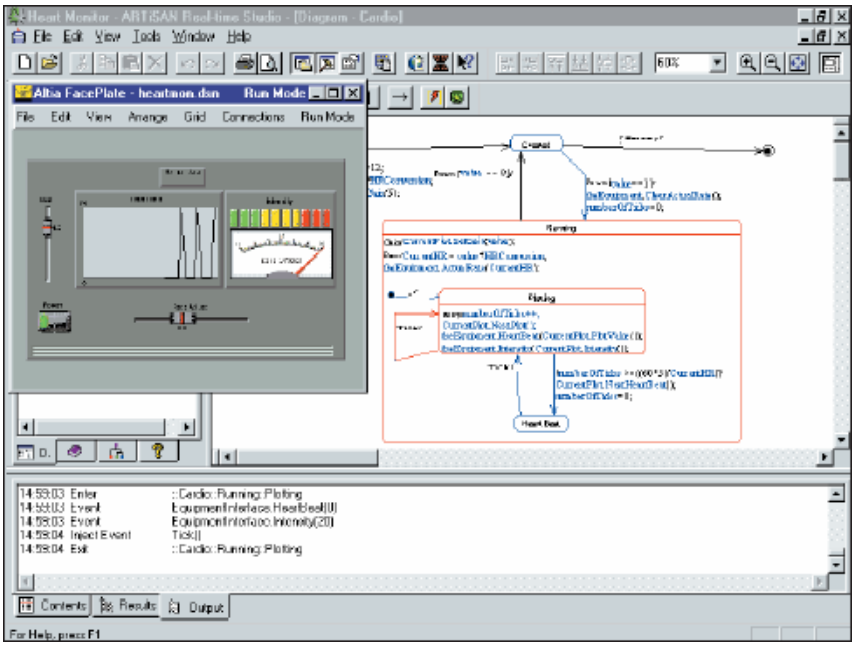


Figure 25: Artisan Real-Time Studio model simulation

implementation or to test prototype objects.

- **Interface Repository Explorer**. It is a graphical user interface for browsing a CORBA Interface Repository. The Interface Repository is shown as a hierarchical tree and objects are shown via their IDL interfaces.
- **Name Service Explorer**. It is a browser for a CORBA Naming Service. CORBA Names and Contexts are shown on a tree-view pane while object references are shown on a list. It also is able to resolve names, create or destroy contexts and to bind or unbind object names.
- **CORBA Shell**. Provides a command programming language and user interface to a CORBA system. The CORBA shell can process scripts and is able to simulate a CORBA environment by scripting clients and servers.

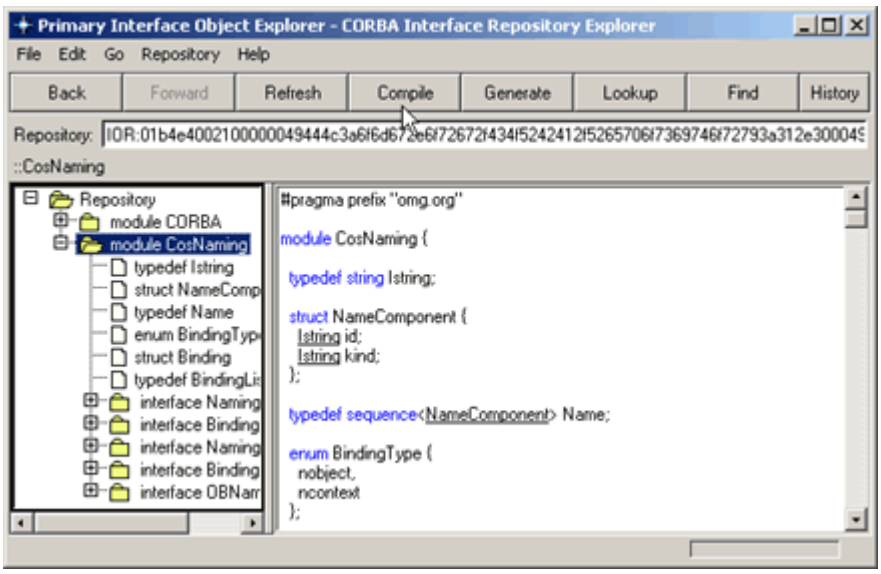


Figure 26: OpenFusion CORBA Explorer

15.6 No Magic MagicDraw UML

MagicDraw UML from No Magic is a visual UML modelling tool with teamwork support. MagicDraw is one of the cheapest options for UML diagramming available in the market. It is not intended for real-time embedded systems and does not support real-time UML extensions (although the metamodel can be extended via stereotypes). MagicDraw generates CORBA IDL interfaces code automatically and UML 1.4 notation and semantics. The CORBA IDL support also provides reverse engineering from IDL sources. It is written in Java which makes slow as runs over the Java virtual machine.

15.7 Microsoft Visio

Microsoft Visio is a general purpose diagramming tool from Microsoft that supports UML 1.2 model diagramming. Visio also allows reverse engineering of Microsoft specific language development environments as MS Visual C++ or Microsoft Visual Basic. Visio provides model error checking and code generation for the Microsoft developer tools.

15.8 Design Tools Comparison Chart

The table below shows a feature comparison chart for several UML modelling tools.

Tool	Manufacturer	UML	Supported Diagrams								Code Generation					XMI
			Use-case	Class	State	Activity	Sequence	Collaboration	Component	Deployment	Real-Time extensions	CORBA IDL	C++	Java	Ada	
Rational Rose	IBM	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Rhapsody	I-Logix	UML v2.0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Real-Time Studio	Artisan Software	UML v2.0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
MagicDraw UML	No Magic	UML v1.4	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓
Visio	Microsoft	UML v1.2	✓	✓	✓	✓	✓	✓	✓	✓			✓			
Describe	Embarcadero	-	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓
TAU-UML	Telelogic	-	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
Visual UML	Visual Object Modelers	UML v1.4	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓		-[9]

[9] There is only support for import/export of model elements, not for diagrams.

16 Platforms

This section is a non-exhaustive list of hardware platforms where real-time CORBA has been run. The hardware platforms described can be used in systems with different demands from embedded soft real-time systems to fault-tolerant embedded hard real-time systems.

16.1 Embedded industrial PC Boards

This type of boards usually have a small size factor (e.g. PC104 or HDD3,5" form factors) and provide interfaces for special equipment (e.g. GPS). Usually this type of boards is equipped with low power consumption processors and common interfaces to external equipment (serial, parallel and ethernet). For obvious industrial reasons this type of boards usually comes with compact flash memory storage device which serves the purpose of hard disk. Figure 27 and Figure 28 show two industrial PC boards with different form factors from Lanner Electronics.

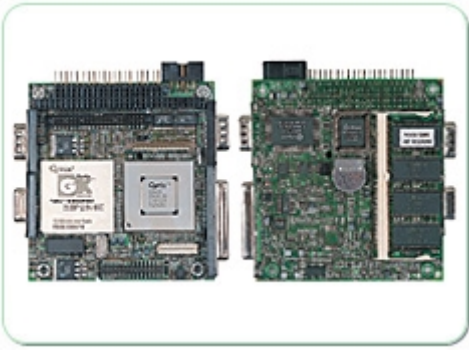


Figure 27: A PC104 form factor industrial PC board

Figure 28: A HDD3,5" form factor industrial PC board

16.2 Networked Device Servers

Networked device servers are general purpose boards that use specialised hardware and processors. As industrial devices, this type of server is

enclosed in sturdy casing and complies to immunity, emission and safety standards.
The network device servers use flash memory for the software system storage and are fully programmable. The usefulness of a device network server relies in the variety of their interfaces. One or more ethernet/fast ethernet ports and several serial RS232 and parallel and USB ports are common configurations for this type of devices. RS485/422 are also commonly found as they provide serial communication over longer distances to devices. **Figures 19 and 18** show a network device server board from AXIS Communications.

Figure 30: AXIS network device server (front side)

Figure 29: AXIS network device server (back side)

16.3 Control Units

Control Units are small computers used for automation of processes such us control of machinery or assembly product lines. They have either modular or integral input/output circuitry that monitors field sensors and controls output actuators according to the programmed control strategy of the unit. They are usually integrated into DCSs.

Figure 31: A control unit from ELIOP

16.4 Time-Triggered Hardware

TTP is a time-triggered network protocol based on a TDMA bus access scheme. There are several classes of TTP protocols raging from TTP/A for low-cost systems to TTP/C for high speed network with high-dependability requirements. TTP provides the predictability needed for hard real-time applications while keeping fault-tolerance capabilities.

Figure 32: A TTP-by-wire box

Figure 33: A TTP development cluster

The TTP-Development Cluster hardware shown in Figure 33 is based on TTP-Powernodes mounted in a rack and with one TTPMonitoring Node for real-time TTP bus monitoring and download. Each TTP-Powernode is equipped with the TTP-C2 controller (AS8202). In addition to TTP, a broad

variety of interfaces is supported: ISO 9141 (suitable for TTP/A, LIN, and ISO-K), CAN, digital I/O, and analog inputs.

The TTP-by-wire box of Figure 32 is a platform for rapid development of TTP-based distributed control systems in advanced automotive applications with high-power actuators. The box is an actuator control unit that offers hardware and software support for direct control of a brushless DC motor.

16.5 Telecom Equipment

Telecom hardware platforms have been from the beginning one of the most widely platforms targeted by real-time CORBA. The reason behind it is that telecom companies constantly suffer the nightmare of dealing with multiple network protocols and CORBA provided them a transparent platform for interoperation.

Figure 34 shows the CISCO ONS 15454 Optical Transport Platform. It is the first optical transport platform that enabled service providers to vary the capacity of an optical network between 155 Mbit/s and 10 Gbit/s. This platform runs a real-time ORB for its management software.

Figure 34: A CISCO Optical Transport Platform

Part 4
Core Methodology

This page has been intentionally left blank.

17 A Methodological Approach

17.1 Methodology rationale

The search of quality and cost/time reduction leads to the definition of coherent methodologies for system development. As a starting point for the discussion, a definition of methodology is provided:

> **meth.od.ol.o.gy** *n, pl* **-gies** [NL *methodologia*, fr. L *methodus* + *-logia* -logy] (1800) **1**: a body of methods, rules, and postulates employed by a discipline: a particular procedure or set of procedures **2**: the analysis of the principles or procedures of inquiry in a particular field.
>
> *Merriam-Webster's Collegiate Dictionary*

What we need is a body of methods, rules and postulates (a procedure) to build complex controllers.

Finding the *correct procedure* to build complex process control systems is almost a no hope task. Heterogeneity in process problems leads to a high degree of variety in application structure and technology. Some of the reasons for the complexity of the search of a methodology are:

- Use of heterogeneous software technologies and heterogeneous platforms.
- Need of integration with legacy systems.
- Use of non deterministic computational methods and platforms; and in particular artificial intelligence technologies, which are inherently unpredictable.
- Knowledge based processing.
- Knowledge extraction, representation, sharing and coherence problems.
- Application structure dynamics.
- Exploratory programming typically used in contoller implementation.
- High level of novelty. Most systems are of the one-of-a-kind type.
- Strong coupling between development phases: inherent life cycle feedback.

- Postponed specifications and designs.
- Complex non-hierarchical development teams.
- Multidisciplinarity of the expertise needed.

The complexity of the tasks the complete methodology must address is enormous:

- ***Software development methodologies***: There is a need for a global application design, process, distributed system construction, real-time, artificial intelligence, validation and verification, product line management, etc.
- ***Control systems methodologies***: The heterogeneity of elementary controllers lead to specific (sub)methodologies for all them. Classical controllers, expert systems (knowledge management), fuzzy controllers (uncertainty representation), neural networks (topology, laws), genetic algorithms, learning, heterogeneous system performance (stability, etc.).

Even when an ultimate methodology is not achievable in principle, at the end, what is really needed, is a software methodology coupled with some control submethodologies.

In the CORBA-based control systems domain, we address the issue of complex software controllers focusing in software more than in control aspects. Classical software methodologies offer good alternatives as starting points for a more suitable one for our domain (a *domain specific methodology* address problems in a focused domain).

Some examples of useful methodologies (they offer some ideas valuable for an integrated control software methodology) are:

- Shlaer-Mellor, Booch, OMT, Objectory, RUP: Generic object oriented methodologies.
- ROOM, OCTOPUS, ROPES: Real time distributed systems.
- ODM, FODA: Domain analysis.
- KADS: Knowledge acquisition and elicitation.
- HINT: Heterogeneous control systems.
- AOM[10], GAIA, ROADMAP, Prometheus, ADELFE: Agent oriented methodologies. They are of special interest for the construction of CCS.

[10] Also Aspect Oriented Methodology.

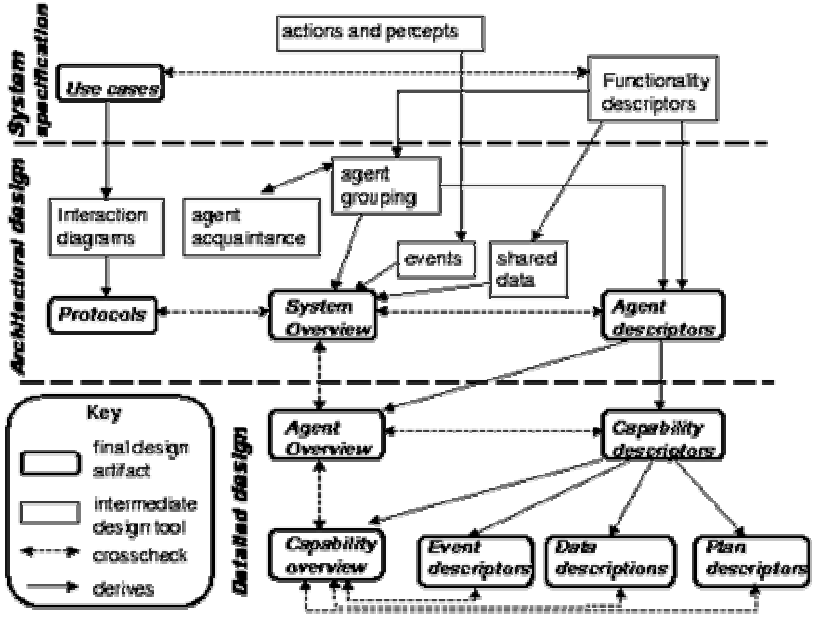


Figure 35: Prometheus methodology phases, artefacts and relationships in the design process.

Each one has it niches, strengths and weaknesses, focusing on specific aspects of the development process or the target system. They are commented below only as a sample of the methodological hodgepodge that managers have available throughout the world of information systems development. There are –literally– thousands of methodologies out there. A question arises: Do we need another one?

17.2 A Methodology for Complex Process Controllers

The methodologies commented are examples of the spectrum of available ones. All them fit to some extent within a complex control methodology targeting one-of a-kind systems.

Methodology based engineering must consider this and provide ways to cope with variability. This will translate in apparent overgeneralization or underspecification of some phases and/or tasks.

The task of complex controller construction can be viewed from two perspectives:

- **Short term**: Oriented towards the construction of a single controller (tackling one control problem); i.e. a classical control system project approach.
- **Long term**: Oriented towards the construction of a successive series of controllers coping with heterogeneous problems in the plant along an extended period of time. This is worth a product line approach.

A comprehensive long-term methodology should provide support for developing a family of integrated control systems for a plant –or related group of plants– following a *product line* approach. This is the best approach to provide a cohesive global control system for the plant.
The base phasing proposed by DIXIT can be seen in the section of DIXIT Methodology.

17.3 Key Concepts for a Methodology

The search for better ways to build software systems is pervasive. Like the slain heads of the Hydra, software engineering threads seem to multiply each time they are defeated. If you are searching the web in seek of relevant literature, some of the key terms for a complex software controller methodology are (in alphabetical order):

- Agents
- Architecture based design
- Artificial intelligence
- Concurrent Engineering
- Reducing design commitments
- Distributed system
- Domain engineering
- Frameworks
- Integration
- Life cycle
- Model based software engineering
- Object oriented programming
- Ontology
- Patterns

- Product Line Engineering
- Real time systems
- Reusability
- Software components
- Software Process

Most of them are quite related (for example *reuse* and *components*, or *domain engineering* and *product line engineering*).

17.4 Divide and Conquer

The strategy of solving complex control problems by decomposing it into partial control problems is called the *divide-and-conquer* approach. This approach basically consists of three steps:

1. *Decomposing* the overall control problem into a complete set of well-defined partial control problems.
2. *Solving* the partial control problems.
3. *Integrating* the partial solutions into an overall solution.

Although this strategy is commonly used to solve complex control problems, few theory and tools have been developed that support this strategy. It has the status of a heuristic method, rather than a structured design method. Traditional control theory is concerned with the *analysis* of the dynamical behaviour of controlled systems, often in terms of differential equations. Therefore, it is well applicable for solving well-defined partial control problems. The divide-and-conquer approach, however, is concerned with the *synthesis* of solutions for complex problems. An important design issue related to this approach is how to deal with the dependencies between the partial control problems when decomposing the overall problem and integrating the obtained partial solutions. The trends in design and implementation frameworks support the divide-and-conquer approach by providing tools for the *structuring* of complex control problems in terms of partial control problems and their interdependencies, and by providing tools for *integrating* partial solutions into an overall solution.

In the case of CORBA-based controller, componentization is granted and composite behaviour can be obtained without sacrificing local functionality by means of proper structuring using the resources provided by the Real-time CORBA specification. Much should be done however, in

the development of tools to support this type of process, because the available design, analysis and development environments fall short of what is needed.

18 Basic processes

18.1 Introduction

The construction process suggested in this document is relatively simple and straightforward. It is based on a series of consecutive phases of development with a domain-centric perspective.

While it is well known that waterfall models of development are not very suitable in general we propose here a stepwise process composed of six phases:

- Early requirements
- Late requirements
- Analysis
- Architectural design
- Detailed design
- Implementation

We consider that CORBA technology enables this simple organisation due to the domain characteristics of component-based complex distributed control systems. A complex distributed controller is composed by a collection of real-time, semi-autonomous, interacting agents.

Complex control systems are typically large-scale, heterogeneous applications, that perfectly match the characteristics that Wooldridge specifies for the target domain of the Gaia methodology [Wooldridge 00]:

- Agents are coarse-grained computational systems, each making use of significant computational resources (think of each agent as having the resources of a UNIX process/thread).
- It is assumed that the goal is to obtain a system that maximises some global quality measure, but which may be sub-optimal from the point of view of the system components[11].

[11] Gaia is not intended for systems that admit the possibility of true conflict.

- Agents are heterogeneous, in that different agents may be implemented using different programming languages, architectures, and techniques. We make no assumptions about the delivery platform.
- The organisation structure of the system is static, in that inter-agent relationships do not change at run-time.
- The abilities of agents and the services they provide are static, in that they do not change at run-time.
- The overall system contains a comparatively small number of different agent types (less than 100).

This means that, to some extent, it is possible to exploit the concepts and models that Gaia proposes in the implementation of the methodology herewith described.

18.2 Early requirements

The early requirements activity in information systems is concerned with understanding the problems faced by an organization and how an information system can help "solve" these problems.

Early requirements elicitation usually begins when the organization identifies some problem and mandates a person or a group of people to investigate possible solutions for this problem.

In the case of control systems in industry this is hardly the case as outsourcing practices limit the amount of detailed knowledge that software analysts can have about the industrial organisation. In other type of control domains (aircraft, military, automobile, etc), they usually have an in-house engineering task force that clearly perceives the situation. This, however, leads to the in-house, domain limited, early requirement elicitation that moves to domain specific solutions.

Early requirements are usually approached through the following steps:

- Define what is needed or wanted by the organization
- Specify as many optional ways as possible for doing what needs to be done
- Specify the potential influence of each option on the other options
- Select among the most interesting or acceptable set of options

Problems, however, are multi-faceted constructs that don't yield to easy analysis. Some of the problems faced by requirements engineers are the following:

- What is needed or wanted is not clear. Among a set of stakeholders there's rarely an agreement on what is needed or wanted.
- Specifying options is not easy
- Selecting, among the options those to implement is even more difficult

The problem of understanding what is needed has been known for many years. Many methods have been proposed to address this problem. The main point of most, if not all, of these methods is the creation of discussions among stakeholders, including developers. The purpose of these discussions is to help stakeholders to understand what is needed and what is technologically feasible. Rapid prototyping, stakeholder workshops, eXtreme Programming, Agile Development etc. have all been developed with the goal of understanding what is needed and what is feasible. The main tool provided for this purpose in UML is the use case. Use cases were used by developers to define how a potential user will interact with the system being defined, but that is typically not precise enough for what control engineers would like to see.

However, in recent years the understanding has developed that during early requirements phases, it is not good practice to think in terms of interactions of users with a system to be built. This is especially important in relation with embedded control systems, where user interaction is minimal. Focusing excessively in humans, may prevent the stakeholders from specifying optional ways of reaching the same expected result. Thus, goal-oriented use cases have been proposed as an alternative.

18.3 Late requirements

Late requirements are detailed requirements that affect the concrete target application.

They are typically captured by means of use-cases and scenarios and lead to the definition of permissions and responsibilities for the roles that constitute the system.

18.4 Analysis

This stage identifies the system features which are essential to the development and implementation process.

In theory, the objective of the analysis stage is to develop an understanding of the system and its structure (without reference to any implementation detail). But in fact this understanding manifests in the form of early design commitments. Typically, this understanding is captured in the system's organisation. An organisation is a collection of roles to be played by CORBA objects, that stand in certain relationships to one another, and that take part in systematic, institutionalised patterns of interactions with other roles.

To analyse a complex system is too hard a task, so it needs to be decomposed into sub-models. Possible models to be used throughout the whole process —not only analysis— are:

- Domain model: it describes the concepts belonging the domain
- System model: it identifies entities of the system
- Roles / Tasks model: it identifies the tasks in the system as well as their relationship
- Object/Agent model: it specifies the objects and their methods

Domain model

It is related to the domain or environment for which the system is developed. This model targets the set of early requierements. It models generically needed classes, attributes and relationships among them. It is important to specify the variables that could posses real-time features. This model is develop simultaneously or in parallel with the development of the other models. It could be modified as needed, once object and domain features are fully specified.

As a conclusion, the former models are just tools to allow in the analysis process. They could consider the system requirements right from the beginning. However, an iterative analysis process could be of better use. Therefore, at a first stage, early requirements can be specified in the analysis phase. Later on, late requirements not previously considered can be added.

18.5 Architectural design

To design consists in defining a solution to fulfil the features described in the analysis phase and its models. The design phase could be divided in two stages: architectural design and detailed design:

- Architectural design: it is a high-level design where objects, methods, relationships, etc are detailed.
- Detailed design: it is a low-level design where the internal structure, attributes, real-time features of the objects are considered.

System model

It allows identifying the objects belonging to the system and those external to it. Being a real-time system, it also allows identifying time-based constraints and relationships. The main tasks to build up this model are:

- To identify use cases and external actors, to describe the system functions related to external entities (end-users, modellers, software engineers, software systems, etc). It could be done extending an UML use case diagram according to RT-UML.
- To identify external events that might affect to the system functioning. To develop a real-time system, the events occurring outside the system may interfere with the system performance. Therefore, a detailed list of events should be made, as well as to specify the event attributes to characterise how the system will react to that particular event. Among those attributes, features such as response time to the event or pattern arrival could be considered.
- To describe the use cases and actors behaviour as a means to characterise the general system behaviour. This could be made by using the UML statechart, activity and sequence diagrams extended as appropriate with RT-UML to express real-time constraints. The statechart diagram allows to model system features as based-time events, concurrency, activities, conditions and actions associated to transitions. The activity diagram allows to model features such as concurrency, synchronisation, branches and transitions. The sequence diagram allows modelling the message sequence between objects and external entities, specifying possible time constraints.

18.6 Detailed design

Agent/Object model

Its purpose is to identify the objects and their methods in the system taking into account some of the elements described in former models.

For each object, elements to be identified are: name, attributes, methods (own and other object methods) and real-time features. This should be an iterative process refine as the system is implemented. It is not, therefore, a static process. Objects and their relationships could be documented and specified using UML class and object diagrams considering real-time features.

Roles / Tasks /Services model

This model is used to identify the roles to be played and the tasks to be performed by these roles tasks that are needed to achieve system performance. From a real-time point of view, three different type of tasks could be considered:

- Hard-time tasks: critical time constraints are applied to achieve the task. If not, it may have consequences on the system.
- Soft-time tasks: soft time constraints are considered in this case. The task should be achieved in a deadline, otherwise it is useless. However, if it is not achieved, there are not major consequences.
- Normal tasks: there are not time constraints to fulfil the task.

Some artefacts could be used to help in identifying tasks and their types. Tasks schemas are used to identify core features of each task. UML statechart and activity diagrams to model the tasks and time constraints. Flow diagrams will allow possible task precedence as well as assigning tasks to objects.

18.7 Implementation

The implementation consists in the construction of the CORBA objects that have the characteristics specified during the design phases.

In theory this should be a straightforward issue; however, CORBA object interaction models do not directly reflect the variety of interactions that can happen in agent communities (see **Figure 36**).

		Topology of Inter-Agent Relationships	
		Centralized (master-slave)	**Decentralized** (among peers)
Information Flow	**Direct** (messages between agents)	**Construction** (Build-time) **Command** (Run-time)	**Conversation**
	Indirect (non-message interaction)	**Constraint**	**Stigmergy** (generic) **Competition** (limited resources)

Figure 36**:** Categories of agent communication [Parunak 03].

However, if we consider the collection of design patterns that is available in the CORBA Services collection, we promptly discover that the implementation of alternative communication models is not only possible but that is already available in CORBA-based platforms.

Concrete implementation details for control agents go beyond what is available in agent architectures. Real-time issues are critical and they are commented in the next chapter.

19 Engineering Objects for Real-time

The engineering process for real-time applications shares all of the elements of the engineering process for non real-time applications. The timing requirements in most cases come in as non-functional or extra-functional requirements in addition to the functional requirements. To begin one has to decide upon whether the timing constraints are hard or soft. Hard real-time constraints are timing constraints that **must** be ensured by the implementation. Soft real-time constraints can occasionally be missed without any fatal consequences. The main issue in this chapter is hard real-time applications.

The hard real-time characteristic can either be imposed by the application or by the designer. The latter case is most common. The hard real-time model is overly restrictive in most applications. In most cases an occasional missed deadline is no catastrophe. However, it can still be a good engineering decision to treat an application as hard, even if that not really is the case. One example can be when it simplifies verification using formal methods.

In order to ensure that a hard real-time application meets its deadlines it is necessary to add schedulability analysis to the analysis phase of the engineering process.

19.1 Schedulability Analysis

In order to at all be allowed to launch a safety-critical hard real-time application one must first ensure that all real-time requirements really are met. Hence, it is necessary to perform the analysis before the application is started or before any new tasks are added to an already schedulable, executing application (dynamic admission control).

A prerequisite for all real-time system scheduling is knowledge of the worst-case execution times (WCET) of the different tasks and of the critical sections within the tasks. There are two main ways of obtaining this information: using measurements and using analysis. When measuring the execution times are measured using code instrumentation or with external measuring devices. The main drawback with this is the risk of being overly optimistic. It is not possible to guarantee that the worst case encountered during measurement really is the true worst case. The second alternative is based on adding the worst-case execution times for the individual native code statements together with the help of a tool. The main problem with this approach is the risk of being overly pessimistic. When applying this technique, generally a number of worst-case assumptions are made one after another. Other problems with the approach are the lack of tools, and difficulties with handling many of the features of modern hardware architectures such as caches, pipelines, and speculative execution.

Several alternative scheduling approaches are available. A main difference concerns whether the analysis is static or dynamic. In a static schedule the complete execution schedule is decided beforehand, typically using some heuristic optimization algorithm. At run-time the task dispatcher simply has to follow the static schedule. Static scheduling is the technique applied for time-triggered system architectures. It has several advantages. The resulting system will have a very high level of temporal determinism, at it is, at every given point in time, known beforehand which task that is executing. It can be used both to schedule the computations in the nodes of a distributed system and the network communication. A large number of constraints on precedence, exclusion, etc can be included in the optimization problem. The resulting schedule can be executed on very small and inexpensive computing platforms, as it typically does not require the functionality of a full real-time kernel. The main drawback with static scheduling is the inflexibility. It is not possible to add new tasks dynamically to the system.

In a dynamic schedule the decision which task to execute is taken on line (dynamically) by the task dispatcher within the real-time kernel. The decision is based on some importance measure associated with the task in most cases a fixed, or static, priority. The kernel always selects the tasks with the highest priority among the tasks that are currently ready, for execution. Fixed-priority based scheduling is the state-of-the-art in today's real-time systems, with a possible exception for safety-critical applications where static scheduling still is employed.

For fixed-priority scheduling a quite mature scheduling theory has been developed during the last 25 years. In rate-monotonic scheduling the priority is assigned to tasks according to their period. A short period means a high priority. The associated scheduling theory covers task communication employing, e.g., the priority ceiling protocol, task context switching overhead, tick-based clock interrupts, and task offsets. The theory also extends to the scheduling of distributed systems based on CAN-bus communication.

Alternatively, the task priority can be based on the task deadline. In addition to the simple periodic task model, the theory has also been extended to cover more complex task models, e.g, multi-frame task models, sub-task models, and offset-based task models. A number of scheduling models task sets that combines hard periodic tasks with aperiodic soft tasks have also been developed. These are typically based on having a periodic task as a server for the aperiodic tasks. A number of approaches have been developed, e.g., the priority exchange server, the sporadic server, the slack server, and the deferrable server.

Within the academic real-time community most of the current research is devoted to dynamic scheduling based on dynamic priorities. The most well-known of these approaches is the Earliest Deadline First (EDF) scheduling approach where the tasks dispatcher always selects the task that is closest to its deadline for executing. Also, for the scheduling model a vast amount of theory has been developed. However, since it is not yet so well-supported by commercial real-time kernels its usage in industry lacks behind.

For soft real-time applications the main focus today is on adaptive and flexible scheduling approaches. These are typically designed to provide best-effort guarantees for task sets that are characterized by uncertainties in resource requirements and where tasks may arrive dynamically. Schedulability analysis for stochastic task sets and for tasks sets that can be modeled as consisting of tasks with mandatory and optional parts (imprecise scheduling models) has been derived. The use of feedback control as means of handling uncertainty and providing flexibility is receiving increased attention (feedback scheduling). This is often combined with quality-of-service approaches where the different performance metrics are treated as quality parameters that the user or the application can provide the desired values for.

19.2 Engineering of real-time control applications

Engineering of real-time control applications adds a number of control related phases to the engineering process. To begin with it must be pointed out that control applications can mean a large variety of different things. One important class of control problems is discrete or event-based control. These control problems are often modelled and implemented using state-transition based mechanisms such as Statecharts, Grafcet, Sequential Function Charts, or ordinary state machines. However, also here the actual implementation is often based on periodic tasks, where the requirement on sampling intervals and deadlines arises from the requirements on response times in the particular application. However, the control problems that are most often associated with computer-based control are continuous control problems where a control algorithm is used to calculate new control signals as a function of the measurement values with a constant sampling interval. This is what we primarily will consider in the sequel.

The first of the design phases is the actual control design. The inputs to this are the requirement analysis that decides what needs to be controlled, and in most cases what the control signals should be. Other inputs to the control design could be a dynamic model of the process to be controlled or input-output data from which a model could be derived using system identification methods. Once the model is available and the specifications on the closed loop are available the actual design process can begin. Depending on the available model knowledge, the inputs available for control, the disturbances acting on the process, and the level of uncertainty involved in the process, a certain control design approach and controller structure is selected, The controller is sometimes derived through the solution to an optimization problem, e.g., in LQG control (Linear Quadratic Gaussian) control or in MPC control (Model Predictive Control). In other cases a certain controller structure is pre-specified as the controller parameters are tuned to obtain a certain closed loop dynamics, e.g. in pole-placement control using state feedback or using output feedback, or in PID (Proportional Integral Derivative) control.

The design of computer-based controllers can further be distinguished with respect to whether the design is performed in the continuous-time domain and then later approximated to a discrete-time design, or whether a discrete time controller is postulated to begin with. In both cases the nominal sampling interval is selected using rules of thumb. In the discrete time case the rules of thumb typically involves the dynamics of the desired closed loop system, whereas in the continuous time design the rules of

thumb typically involves the bandwidth of the open loop system. In most cases the control design is performed assuming a negligible input-output latency. It is assumed to be so small in relation to the sampling interval that it can be safely be ignored. However, especially in networked control applications this is far from always being true. Another alternative is to attempt to implement the system in such a way that the latency is constant. In that case the input-output latency can be modelled and compensated for at the design stage in the same way as if the latency originated from a constant transport delay within the controlled plant. This approach is especially suitable if a time-triggered network protocol such as TTP/C is used, where the network latency jitter is very small. However, delays always have a negative effect on control performance. For example, it is in many cases better with a varying but short latency, than with a longer but constant latency. This is often true also if the constant latency is compensated for. Even better performance can be achieved if the compensation is performed for the average value of the time-varying delay.

If it is possible to measure the actual latency from sample to sample then it is possible to, to a certain extent, compensate for the latency on-line. The latency can be viewed as a temporal disturbance acting on the feedback loop, or as an uncertainty. By applying control-based methods such as disturbance feed-forward or gain-scheduling the variations can be compensated for. However, it is normally only the part of the latency that lies between the sensor node and the controller node that can be handled in this way. In networked control loops a prerequisite is also that time stamping information is associated with the measurement data.

The analysis of how control loop timing parameters such as sampling interval, input-output latency, and the jitter in these, affect control performance is very complicated. For linear system and where all delays are independent from sample to sample it is possible to apply jump-linear theory and Markov theory to perform the analysis numerically. This theory has been packaged within the Jitterbug toolbox developed by Lund University.

Although it is possible to, to some extent, analyze how imperfect controller timing will affect control performance, this is seldom done in industrial practice. Instead there are too often a clear separation between the control design group and the software implementation group. The control designer simply assumes that the software group will provide an implementation with jitter-free sampling and negligible latency, whereas

the software engineers often have a very vague understanding of how delays and jitter influence control performance.

Simulation is a common activity in all control design. However, normally the simulation performed is restricted to the pure control aspects, disregarding the effects caused by the computing and communication. The normal setup is simulation of feedback loops consisting of continuous time models of the controlled plant and discrete time models of the controller. Issues like computational latency and network latency are at best modelled as constant time delays. However, using the new Simulink-based simulation tool TrueTime, developed at Lund University, it is now also possible to simulate the true timely behaviour of a networked control loop, taking issues like delays caused by preemption and blocking in real-time kernels and delays caused by collisions and resendings in networks into account. Hence, it is possible to perform true co-simulations of the control aspects, computation aspects, and communication aspects of a networked control loop.

The fact that most control loops are relatively robust towards timing variations can also be utilized to obtain more flexibility. This is achieved if the control performance is treated as a quality-of-service parameter (quality-of-control), for which the designer provides desired values together with acceptable ranges. Associated with the different available controller are capability parameters that decide which level of performance the controller can obtain as a function of the sampling interval and the latency of the control loop. During run-time an adaptive scheduler can then adjust sampling intervals and switch between different controller based on feedback from the current resource utilization and control performance. Methods based on contracts and negotiation can be applied. The resulting system has the potential to cope better with uncertain and dynamically changing workloads than current, statically designed, systems. Dynamic system solutions of this type create new demands on representation of timing related information. The timing information must be available at run-time and be accessible from the scheduling system. The scheduling system must also have access to timing information from the network, e.g., worst-case and best-case latencies or latency distributions. It is still an open question how this information best is represented. One possibility would be to represent this information in the IDL. The area of dynamic feedback scheduling-based control systems, although promising and interesting, is, however, still at the research level.

Part 5
Case Studies

This page has been intentionally left blank.

20 Strategic Plant Control

20.1 Introduction

Strategic decision making in complex continuous process plants (chemical, oil, cement, etc.) has been a topic restricted for humans since ever [Sheridan 83]. The reasons for this restriction —lack of automation indeed— are grounded in the unpredictability derived from plant complexity [Åström 00].

Strategic control issues are those related with the top level management of the plant. They are oriented to reach global objectives that, in many cases, are not suitable to be integrated in an automated planner due to their heterogeneity and abstractness. Examples are safety, production, stability or maintainability. In these plants, the responsibility for this type of decisions is always of a human that decides what to do in any problematic situation.

Automation, however, is desirable —in a general sense— in any type of plant and for any type of task if this automation does not mean the sacrifice of any one of those top level objectives. Partial automation is achieved for some of the objectives but no total solution is available in general because these plants are mostly unique (at least after some time of operation).

The flexibility offered by present day information technology helps bridge the gap between a heterogeneous collection of information sources without sacrificing dependability or performance. Automated decision support systems are emerging to help human operators in making reliable, fast, and economically advantageous decisions [Petrov 00].

In this chapter we are going to show an example of how a suitable integration technology like CORBA can lead to a specifically tailored decision support system that can provide an integrated plant view for strategic decision making.

Not surprisingly, the construction process for these tailored systems is extremely complex due to the needs of integrating heterogeneous information sources (new and legacy systems) into a single whole application. Extreme complexity is reached when the system is designed even for integration with future (not yet existing nor specified) systems as is the common need in complex plants.

The PIKMAC system described in this chapter was developed[12] to support human-centered operation of a cement plant during periods when this operator is the only person in the plant (*i.e.* the only person capable of making strategic decisions in real-time). This application exploits the integrational capability of CORBA middleware [OMG 00] to gather heterogeneous information that is fused into simple quality, economy and maintenance views. This application was deployed atop the first version of the ICa integrated control architecture [Sanz 99b] that was specifically built for the control systems domain.

20.2 Strategic Process Control

Control systems in large plants are hierarchically organized to integrate the complex functionality required from them. **Figure 37** shows an overview of the layers of a typical hierarchy. Lower layers are typically available in any process control system and higher layers are typically custom-built to target specific plant needs.

[12] This work was funded by the European Commission through Project IST DIXIT

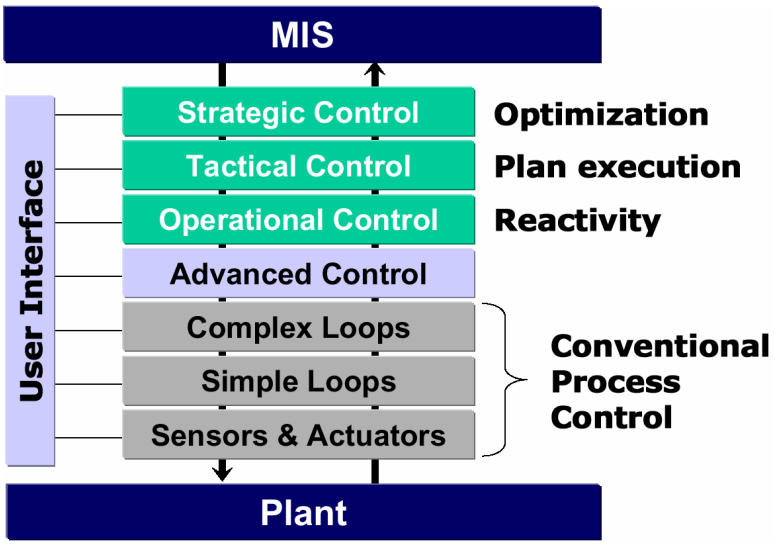


Figure 37: Typical hierarchical organization of a layered control system in complex plants.

Strategic process control is the set of activities regarding top level decision making in a process plant. Strategic control is traditionally considered a management activity and hence studied as part of business processes and practice [Simons 95]. In our domain we consider it as mostly related with global optimization and risk management at the enterprise level. Hence this chapter focus strictly on a technical level, addressing strategic control of production systems from a purely technical perspective.

While strategic decision making is typically considered a human activity, the necessary incorporation of advanced computing mechanisms in the top-level decision making process in large industries, makes this process an mixed human-machine system. In the case of management of purely technical systems, the decision support system becomes critical for the proper and timely understanding and assessment of the situation of the plant.

Decision Support Systems (DSS) are a specific class of interactive computing system that support human decision-making activities. DSS in control applications are interactive computer-based systems intended to help plant operators (decision makers) use control, computing and communications technologies to exploit data, documents, knowledge and/or models to identify and solve problems and make proper decisions in real-time. Five specific DSS types are typically identified:

- Communications-driven DSS,

- Data-driven DSS,
- Document-driven DSS,
- Knowledge-driven DSS, and
- Model-driven DSS.

Expert systems technology has been instrumental in the implementation of knowledge-based decision support systems leading to multiple successes in the enhancement of processes carried by human operators. Good results have been achieved even in the presence of uncertainty by means of mechanisms based on bayesian methods or fuzzy logic.

But in many cases, decision making is done in the presence of excess of uncertainty that forbids automatic decision-making. This is particularly clear during fault and emergency management. Even while experiments have been done in the automatic management of these situations in small systems [Bernard 99], the technologies available so far do not scale up to complex industrial plant emergency management. In these situations decisions are necessarily taken by hybrid decision makers (human + machine).

The work described in this article focus on the implementation of a concrete DSS that uses CORBA technology to integrate heterogeneous sources of knowledge to help the decision making-process.

20.3 Operational objectives for the Contes plant

Cement production is somewhat tricky due to the extreme nature of the process (chemical reactions in high-temperature fused material) and the nature of the input solids (they are usually rocks from a mountain nearby). Chemical composition is critical for the quality of the final product (it gets hard by chemical reaction with water) and the main cost is not raw materials but energy and salaries.

Energy is obtained from the combustion of different products: coal, fuel, oil, waste, etc. (during one of the demonstrations of this application they were burning peach). Each type of fuel has its own qualities (energy per kilogram, cost per kilogram) and side effects (especially in kiln controllability).

Cement plants suffer —as any other industrial plant— the global business objective of reduction of human personnel in all types of tasks. But this

must be done with a minimum sacrifice in the rest of the strategic objectives.

Some of the key factors of success for cement industry are to gain capability to quickly react to customer needs; be able to employ different fuels of poor quality (heavy fuel, recycled oils, waste, etc.) without altering the quality of the final product; and reach the capability to compete at a larger geographic scale by permanently streamlining the production costs, and optimizing the value created by the company [DIXIT 98b].

To advance in the achievement of these capabilities Lafarge Ciments managers decided to develop a new generation of IT applications, taking advantage of existing process automation and supervision applications already installed, to be used by all types of plant personnel.

The ideas behind this technology programme were:

- To extract from the huge amount of data continuously stored in CIM.21 process database and other plant databases, the synthetic information relevant for decision making at any moment;
- To derive through an explicit (mathematical) model or an implicit (neural net) models high level value added information consistent with the overall target objectives of the plant (cost, quantity, quality);
- To provide decision support to control room operator to take better decisions in case of failure, in a way that incorporate commercial data, economic factors and human resources constraints to the pure technical data usually taken into account until now;
- To gather in a single user interface all the information, heterogeneous in nature, to facilitate the global control of the production;
- To offer an innovative presentation paradigm and exploration tool making easier production global performance comparison for different moments (for example now *vs* one month ago) or different process configurations (fuel types, clinker quality, etc).
- To facilitate the real time dissemination of production performance information according to formats that can be shared and understood by all the plant department people (management, maintenance, commercial, production, quality, etc.).

The plant selected for the development of the PIKMAC tools is placed in Contes, France. This plant had another challenge for PIKMAC, because it was operated by only one person during night and week-end shifts (this means that he was the only person in plant during that time).

20.4 The PIKMAC decision support system

PIKMAC stands for *Process Information and Knowledge Modelling for Advanced Control*. This CORBA application was demonstrated in the Lafarge Ciments cement plant in Contes (France).

The purpose of this system is to keep operators informed to perform a better strategic control of the process in terms of *maintenance*, *quality* and *cost*. PIKMAC is based on the fact that a lot of process information is continuously acquired (sensor measurements, control system variables, operator commands, automatic test laboratory, etc.) but remains under-used in most cases.

This information concerns all the parts of the cement production process —raw material mill, kiln, cooler and clinker mill— and covers a wide range of process behaviour characteristics.

While several applications could be designed in order to efficiently support the plant operators and process engineers only three integrated applications were demonstrated in PIKMAC:

- Production Performance Synthetic Indicator (PPSI): provides real time estimations of production performance (quantity and cost) using a Global Production Control concept.
- Quality Deviation Early Detector (QDED): estimates continuously key quality parameters making possible the early detection of non-optimal situations.
- Alarm Management Operator Assistant (AMOA): helps the operator —in particular during night shifts and weekends— to deal with alarm situations to optimize calls to maintenance people.

20.5 Global Application Structure

The global structure of the application is very simple: it is a collection of active and passive agents running over the ICa broker. These agents provide different types of functions and their interaction capabilities are expressed by means of CORBA Interface Definition Language (IDL) [OMG 00].

The system is depicted in Figure 38 showing the collection of agents that composed the application.

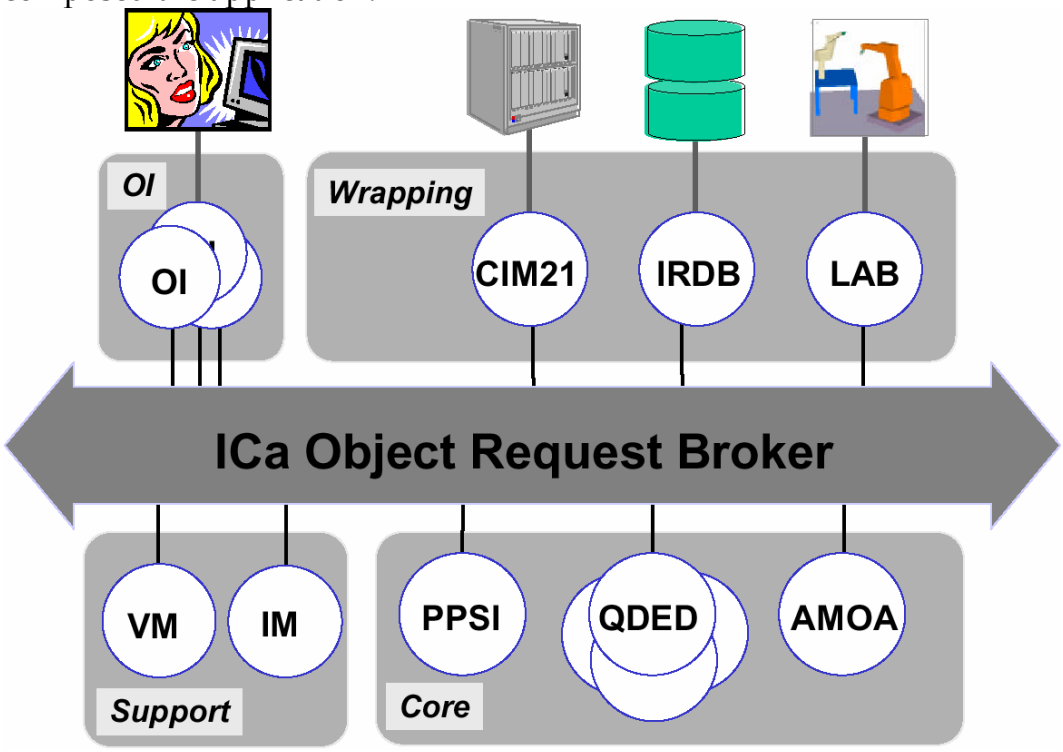


Figure 38: PIKMAC is built as a collection of active CORBA objects that provide specific pieces of functionality: core systems, legacy application wrapping, operator interface and system support.

The domain architecture for PIKMAC was designed by Lafarge Ciments personnel to follow their own ideas on this new set of IT tools for plant management. The people in charge of the DIXIT architecture did only map that conceptual architecture to a specific CORBA implementation based on ICa agents.

PIKMAC agents can be grouped in four categories:

- Core systems: they provide the basic functionality of the PIKMAC demonstrator. They are QDED, PPSI and AMOA.

- Data sources wrappers: they wrap external data sources to be exploited in a CORBA environment. They are CIM.21, LAB and IRDB.
- Operator interface: they are the user interface for the system. There is only one type of agent (OI) but it can be replicated in any number of hosts.
- System Support: they provide hidden functionality for the rest of the agents. They are the ICa Monitor and the VarManager.

20.5.1 Data Sources Wrappers

The three main data sources for PIKMAC are:

- the real-time process database of the CIM.21 control system,
- the incident report database (IRDB) and
- the automated laboratory (Lab). All they are legacy systems that can be accessed using their specific APIs.

They appear in PIKMAC as conventional CORBA objects resulting from wrapping part of the APIs.

20.5.2 PPSI

The PPSI agent implements the core functionality for PPSI service. It performs calculations of process throughput and cost per processed unit. These calculations are done online in a continuous manner and uses sampled process data gathered from the plant.

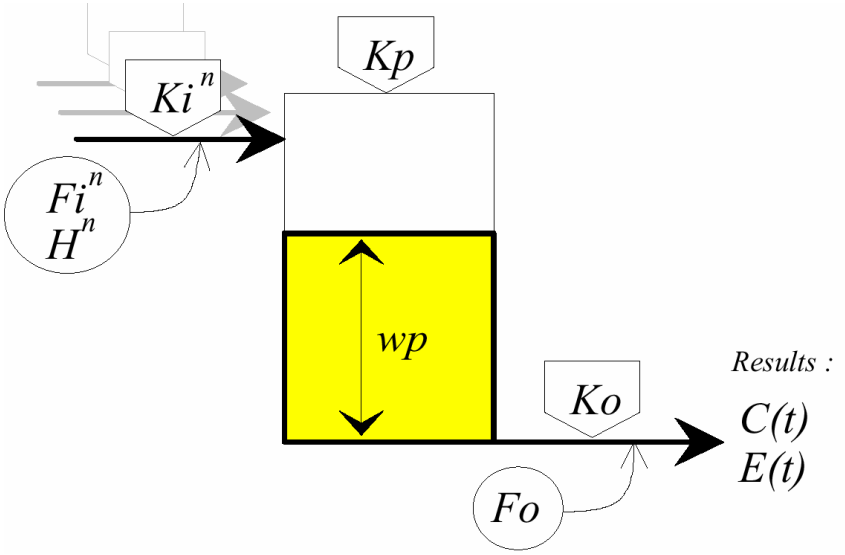


Figure 39: An overall view of PPSI calculation model.

The PPSI implements a sophisticated cost calculation model (Figure 39) which takes into account the long processing time for the raw material in cement production.

20.5.3 AMOA

AMOA is built entirely on G2, an expert systems development tool from Gensym (www.gensym.com). It uses DIXIT's G2-ORB Bridge to connect to the plant data sources and other applications like the operator interface through the ICa ORB.

The main component in AMOA is the *process reasoning module*. It gathers and analyzes the real data coming from the plant, generating reports regarding present and possible future failures. In case of a problem situation, AMOA will generate reports informing the user about the real root causes of the problem, based on the process analysis it does. AMOA will also guide the user in the task of deciding whether a maintenance team is to be called or not, and which is the maintenance team that must be called if it is the case.

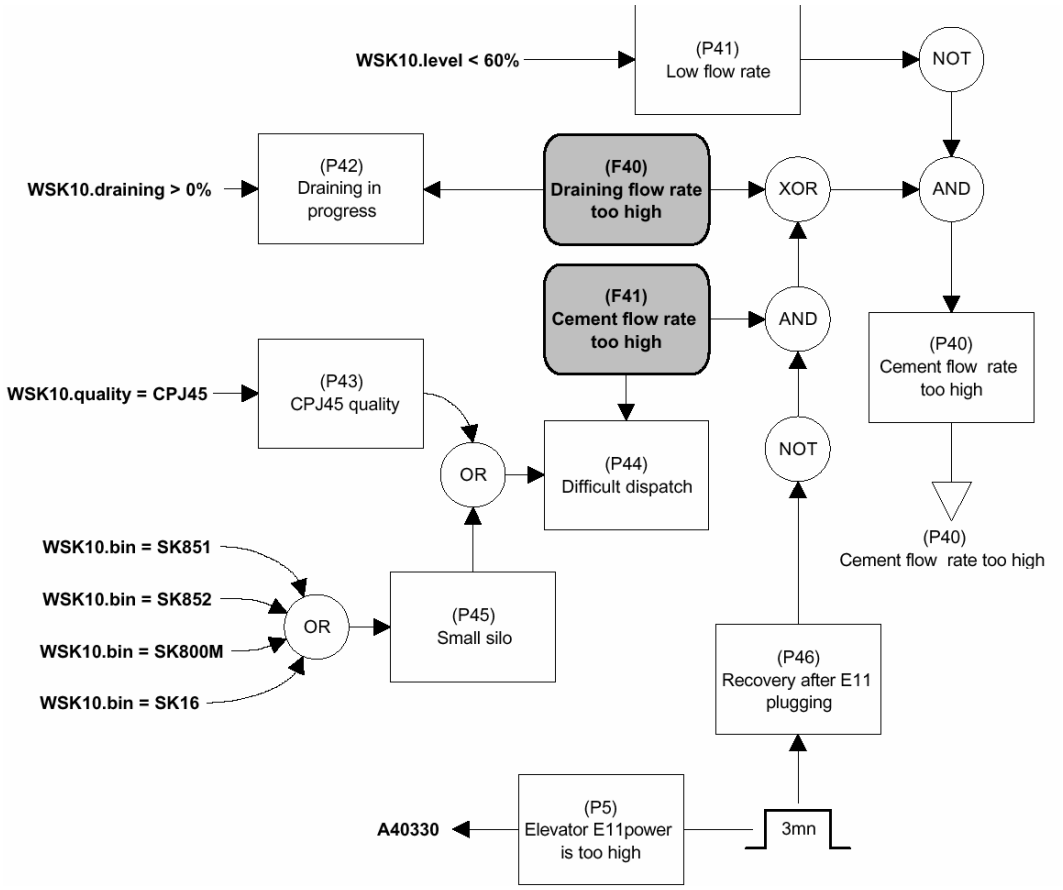


Figure 40: Failure causal flow chart about P30 flow rate used in the knowledge base of AMOA.

The interaction with AMOA can be done using the generic PIKMAC user interface (that provides a simple synthetic view) or the more specialized AMOA native G2 interface.

20.5.4 QDED

The QDED is not an agent but an agent society predicts some critical quality properties of cement (free lime percentage, SO3 ratio, C3S ratio and C3A ratio) as the cement is produced in the factory.

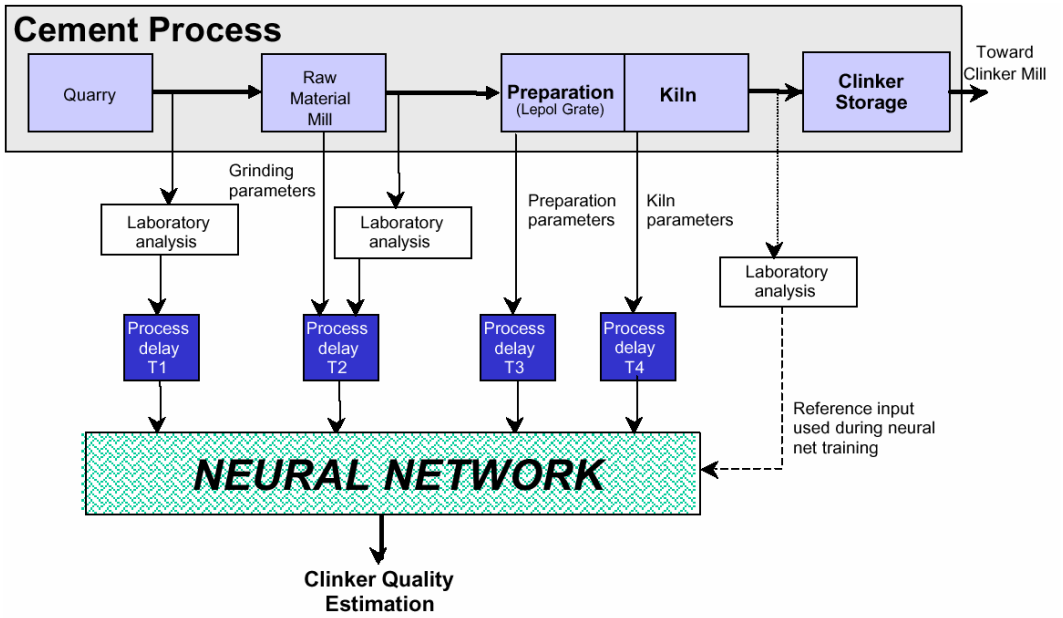


Figure 41: The QDED neural network uses inputs from all the cement process with proper delays to estimate present quality.

QDED agents use neural networks trained on historical data to make online predictions (see Figure 41). The core issue here is to provide reliable estimates of these parameters, avoiding the delay and the cost derived from slow and expensive automatic laboratory analysis.

A single network is trained for each of these tasks, and the final version of QDED thus requires all four agents running simultaneously to provide estimates for all four parameters.

20.5.5 Operator interface

The PIKMAC system is deployed over a heterogeneous collection of computing equipment. In the demonstration application most parts are running on Alpha/UNIX and Intel/NT platforms.

The user interface runs on NT computers providing a synthetic view of the plant state from the three perspectives: cost, quality and maintenance.

Figure 42 shows the main user interface for the quality section. It provides numeric and visual information about the status of quality elements for the cement provided by a remote QDED agent.

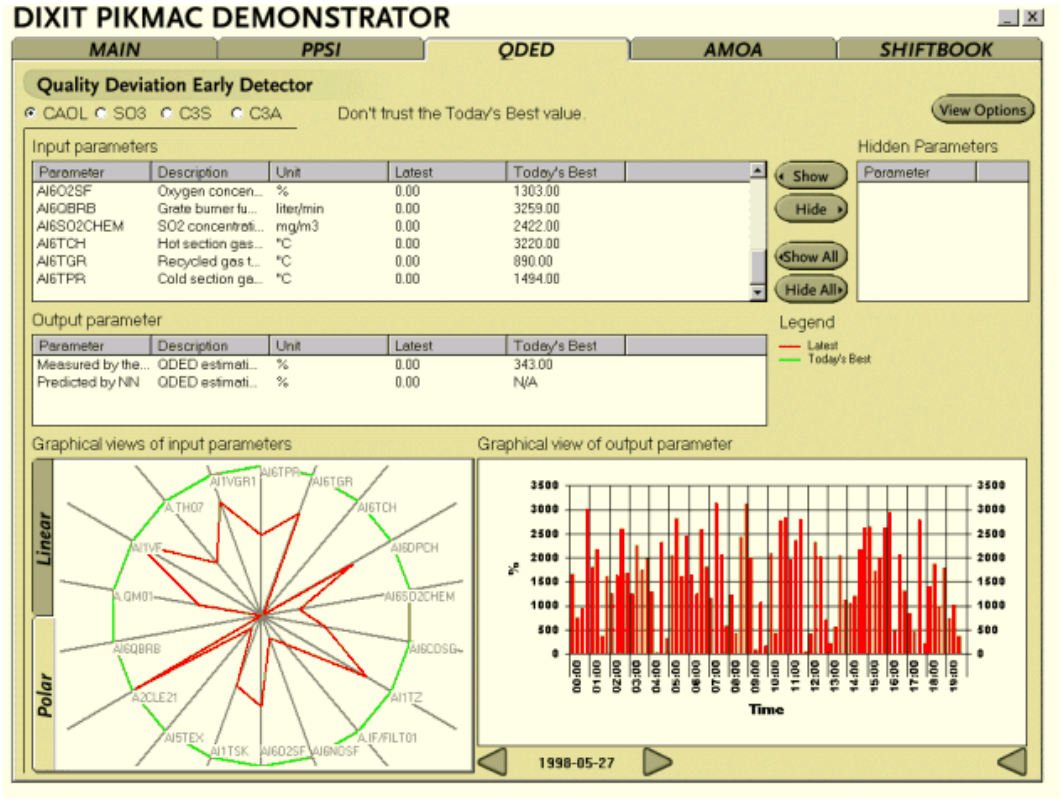


Figure 42: PIKMAC has a user-friendly human-machine interface that runs on Windows NT platforms. This figure shows the part of the operator interface that contains quality information from QDED.

There is no fixed number of operator interfaces than can be run in an installation, thanks to the brokerage mechanisms provided by the CORBA middleware.

This agent is built using native Microsoft technology elements (*i.e.* COM components, OLE Automation and Active-X controls) that are connected to the CORBA world by means of COM-CORBA interoperability mechanisms.

20.5.6 System Support

There are two agents that provide support for the rest of the system. ICa Monitor continuously monitors the state of the systems controlling the particular status of any agent.

PIKMAC VarManager is a real-time database with added features for domain applications: it gathers data from data sources upon schedule, can

process this information and calculate derived data and also supports subscription services for any data it handles.

20.6 Lessons learnt

The PIKMAC system demonstrates how integration architectures and technologies help develop future plant-wide integrated control systems in an easy and modular way. All the systems described here were developed by five development groups in four countries and put to work together in a matter of hours (only for demonstration purposes).

The PIKMAC application can only be considered a demonstration of the technology and not a full fledged application. More work is necessary to make it a dependable decision support system. Extensions to support expert systems justification [Guida 97] or fault tolerance [OMG 99b, Butler 93] are obviously necessary to exploit it in a real context.

CORBA technology is here to stay and offers a clear opportunity for control system developers for leveraging previous developments in an easy way. But some contribution to OMG is needed from the controls

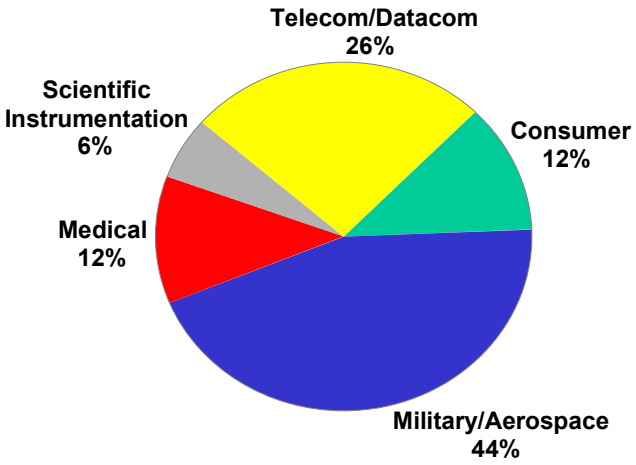


Figure 43: Current use of ORB middleware in embedded and real-time applications (from [Czerny 00]).

community. While CORBA is been widely used in real-time settings (see Figure 43) not many industrial applications are described that pose critical requirements for the ORB infrastructure.

The contribution of control systems engineers is necessary for this technology, and the results that we can obtain from it are extremely high

for the deployment of new control technologies in large, complex and distributed plants. A new working group in Control Systems has been recently chartered by the OMG to foster the suitability of OMG technologies for control systems implementation.

21 Strategic Emergency Management

The system described here was implemented to support human operators dealing with cement plants but the technology behind it was also successfully used to implement other strategic controllers focused on risk reduction and emergency management.

The RiskMan system [Sanz 00d] was implemented as a solution to the problems of emergency management in chemical plants by means of ICa-based agents. The basic subsystems in this application were a *prevention* system, an *emergency* manager and a *work permit* manager to handle human induced risks. All them were implemented using CORBA objects over the ICa integration middleware (see Figure 24).

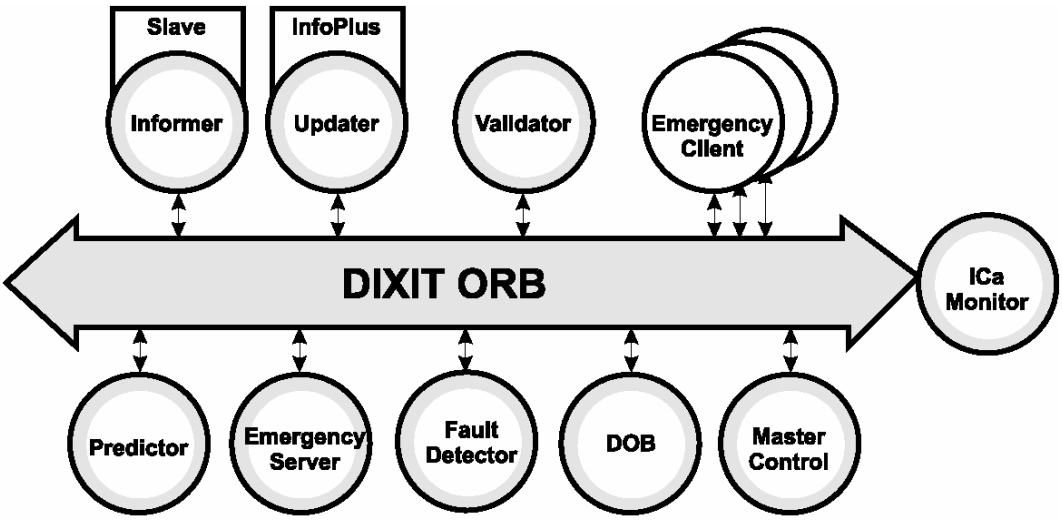


Figure 44: RiskMan Application structure as a collection of interacting CORBA objects. The figure shows the base middleware (ICa) and the objects that implement system functionality.

The *Emergency Manager* deals with the management of emergencies and implementation of the plant safety plan following the already established policies for dealing with emergencies. Safety protocols for this plant are

very complex because they involve safety regulations from the European Union, Spanish laws, Catalonian laws, Tarragona's chemical sector plans and Repsol's own policies. This includes the real-time elaboration of the emergency organization chart, *i.e.* the human organization structure to deal with the emergency, under the constraints posed by the emergency as well as the communication of the actuation procedures to the personnel involved in the emergency.

Once the emergency is declared, the system automatically handles all issues related to the organization chart elaboration and information management.

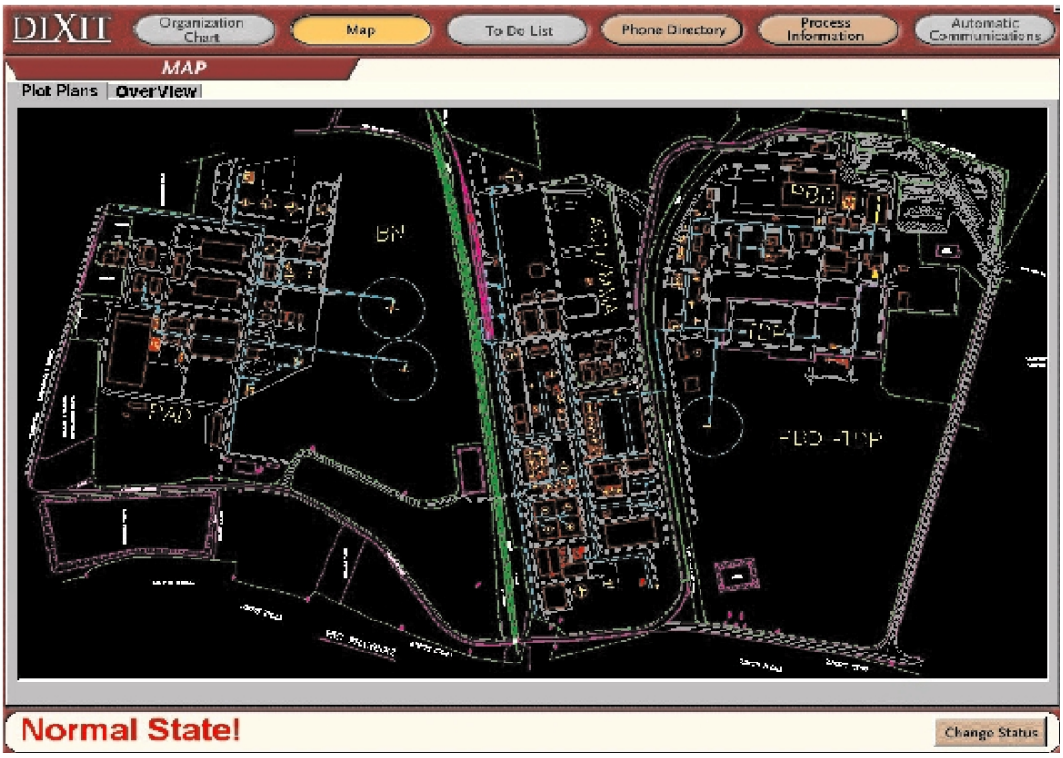


Figure 45: RiskMan Emergency manager user interface. The figure shows the navigation map used to focus on specific areas of the chemical complex.

The *Preventive System* monitors the state of a subsystem detecting abnormal situations before they reach a critical stage. This component is only applied to a set of selected equipment in order to fully test its suitability and correctness. A complete implementation, i.e. covering the whole complex, was out of the scope of the project. A rule-based approach is utilized in this software module.

The acceptance or not of certain human-performed maintenance activities depends on the result of a risk evaluation. It was estimated that automating these protocols, at least partially, could save a lot of time and reduce the risk of accident in the maintenance operations. This leads to the definition and implementation of a *Workpermit Manager*, an application that helps Repsol personnel in the management of the protocols for the authorization and control of risk-inducing maintenance operations. In order to do so, the application automates many of the procedures that are currently done by hand with the subsequent loss of time and increase of risk. The application helps the user by considering relevant on-line process information that should be taken into account for the authorization and execution of such maintenance operations.

22 The HRTC Process Control Testbed

22.1 Introduction

The Process Control Testbed is an experimental platform to evaluate distributed systems software in the implementation of integrated process control systems.

The purpose of the PCT is:

"The main objective of the distributed process control testbed is to identify (mainly hard real time) requirements for distributed control systems and perform experiments in conditions of systems heterogeneity and legacy integration. Experiments will be done using conventional IIOP and the new real-time protocol. "

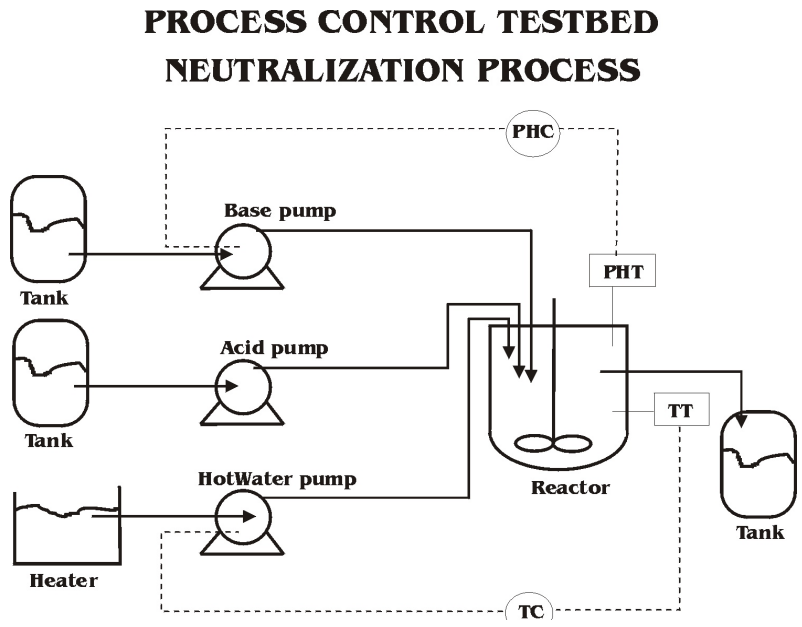
22.2 Process description

The physical process (the plant) is the neutralization of acetic acid (0.1M) with sodium hydroxide (0.1M). It has two control loops: one controls the pH and another one controls the temperature.

The process has two feeds, the first one is the acid which is the one to be neutralized. This is set to a fixed flow and concentration and any variation is a disturbance to the process. The second one is the base feed, this feed is set by the pH control loop. This loop has the pHmeter a controller (PI) and the base pump as the actuator.

The PCT uses a small CST reactor for the neutralization process and its output stream goes to a product tank through a weir.

There is an additional control loop for temperature control. This loop has no special relevance for this particular process but it is needed for the experiments to be taken. This loop has a temperature transmitter (pt100) a controller (PI) and pump as actuator. This pump is fed by hot water



coming from a heater.

Figure 47: Process control set up

22.3 Computing Components

22.3.1 Ethernet network

A 100BASE-TX Ethernet with (redundant) connection to 2 switches. A maximum of 8 nodes are used in any of the experiments.

22.3.2 Instruments

Sensors

Sensors measure physical values of the process variables. There are different types in a process plant: temperature sensors, pressure sensors, flowmeters, etc.

Sensors are usually connected to conventional (4-20 mA) or 'smart' (digital bus) transmitters that transport the measurement to the control system. In the commercial DCS they enter through the I/O cards.

For connecting the sensors to the Ethernet network in the PCT it is necessary to have a wrapper node that, ideally, could be embedded in the instrument. In the PCT we use dedicated computers to perform this function.

Two kind of sensors are used:

1. Actual (physical) instruments with a transmitter and an input card in the DCS (analog signal or serial interface) or the wrapper node (serial interface).
2. Simulated sensors instantiated in any node. They allow testing the effect of a large number (a more realistic scenario at a reasonable cost) of sensor on the system performance.

Actuators

Actuators are the final elements of a control loop, modifying the process conditions as the result of the controller command. They include control valves, frequency variators, etc.

As it happened in the case of the sensors, a wrapper node (or the DCS) with I/O cards is necessary to connect them to the network (or the HPM controller, see TPS subsection below). Also two kinds of actuators are used in the experiments:

1. Actual actuators
2. Simulated actuators

Controllers

The controller receives the signal of the sensors and as a function of their setpoints and control algorithms calculate the output signal to be sent to the actuator. There are two controller types:

1. Controllers that are integrated in the DCS (HPM) that receives and sends signals (initially) internally without entering the Ethernet network.

2. Controller nodes specifically built for this testbed as CORBA objects that implement the control algorithms, and that communicate with the sensors and actuators through the Ethernet or TT networks.

22.3.3 Human-Machine Interface

The Human-Machine Interface in most plant control systems is usually a graphical interface, with or without windows. The HMI allows the monitoring function carried by human operators, as well as their interaction with the process by means or control actions, such as starting up/stopping units, changing setpoints, etc.

In the PCT, graphical HMI nodes are used in order to access and interact with the data and agents on the network.

22.3.4 Database

Historical databases record selected data from the control system configuration and/or operation. Also, they usually contain the system software files. Operators can typically access them through HMIs.

22.3.5 Commercial DCS (TPS)

An already available commercial DCS, the Honeywell TPS (TDC 3000), is used. The system is composed by:

1. A High-Performance Process Manager (HPM) controller
2. A Global User Station (GUS)
3. A History Module (HM)
4. A Network Interface Module (NIM)
5. A redundant Local Control Network (LCN)
6. A redundant Universal Control Network (UCN)
7. Several I/O cards:
    a. Analog Input (AI)
    b. Analog Output (AO)
    c. Digital Input (DI)
    d. Digital Output (DO)
    e. Serial (Modbus) Interface (SI)

With the available hardware, to integrate the TPS in the Ethernet network the system could be wrapped (with a PC) via the serial bus or via the GUS.

The serial bus has the advantage of directly accessing the controller (HPM) like sensors or actuators do.

A temperature sensor and transmitter enter the system through the AI card. The heating module is controlled by an AO output signal.

22.3.6 Simulation

An increasing number of control and monitoring functions utilize models in on-line and off-line applications as:

1. Hardware in the loop
2. Operator training

In such context, the availability of pluggable simulation nodes accessible by the other components in a transparent way will constitute and advance from the current state. The software AbACUSS is used in this testbed by means of a CORBA object wrapper.

22.4 Functionality

The PCT is able to comprehend the functionality of both present and future process plant control systems. The original idea that motivated the construction of the testbed is to try to build such a control system using CORBA components and check whether it was possible to:

1. Perform the tasks that current systems usually do.
2. Accomplish the tasks that future systems are expected to achieve.

The results of the experiments (mainly the negative ones) will identify the features needed in distributed software technology to be used in control systems.

Some of the experiments performed are:

1. Single control loop
2. Legacy system integration
3. Simulation components integration
4. Traffic capacity test
5. Concurrent access

22.5 Hardware Setup

Figure 48 shows the basic process hardware (& equipment) setup. This topology is used to perform the experiments although in some cases a node can change its functionality as in the case that the sensor (H007.1) becomes the simulation node to perform Operators training with the HMI. In other cases an additional nodes can be connected to the network, as in the intensive traffic experiment.

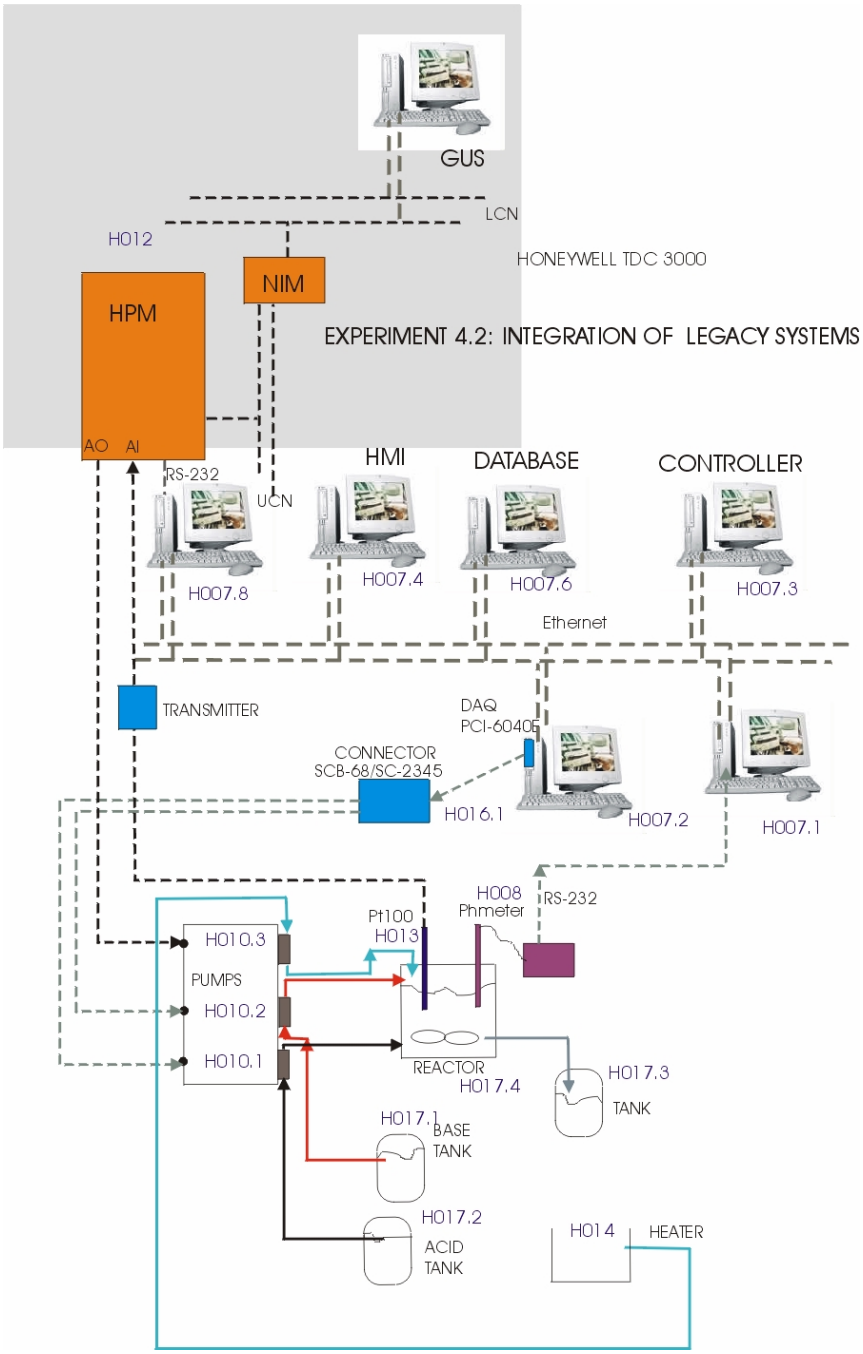


Figure 48: PCT Hardware.

22.6 Software Setup

The software of the PCT is based on CORBA objects that, while based in the same pattern, offer different functionality.

Figure 49 shows the implementation for the main control loop. Node functionality is as follows: Database is a server to all the other nodes, Sensor and Actuator are clients to the database and servers to the regulator. Regulator is a client to the Sensor, Actuator and Database nodes and finally the HMI is a client of all the other nodes. Each line represents a CORBA thread for client-server communication. Communication between the sensor wrapper node and the actual pH sensor are through a serial port. Communication between the actuator wrapper node and the actual actuator (pump) is made through a PCI data acquisition card delivering a 0-5volt signal to the pump.

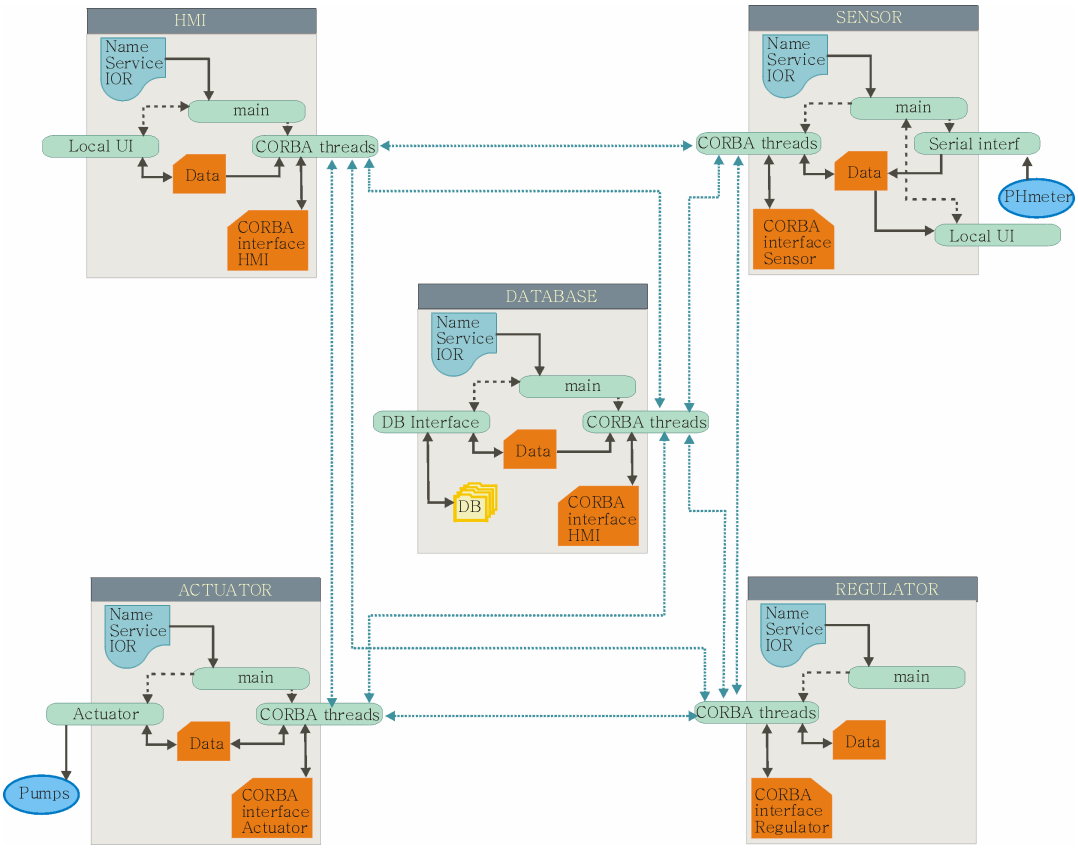


Figure 49: PCT Software organisation for the simple control loop experiment.

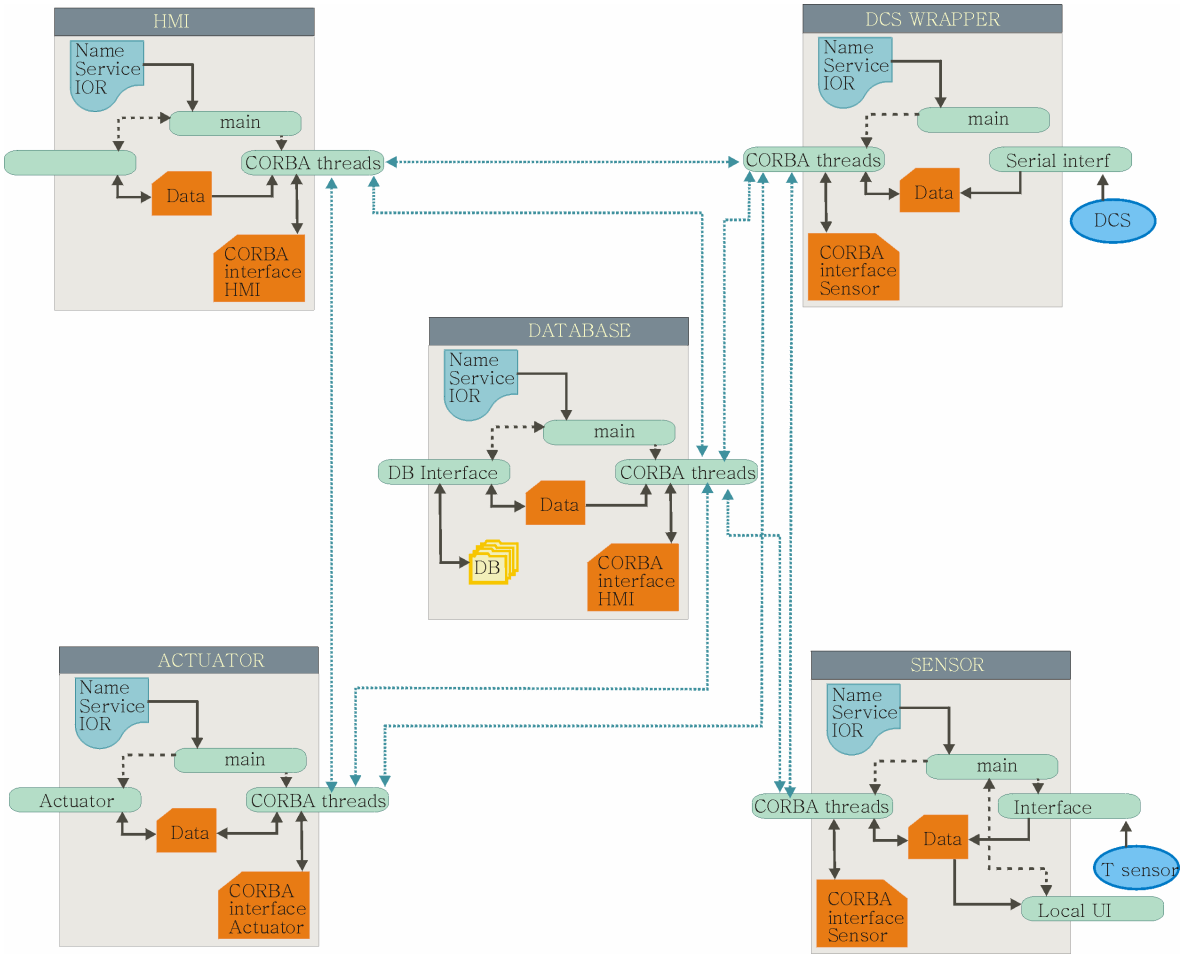


Figure 50: PCT Software organisation for the legacy integration experiment.

Figure 50 shows the software implementation for the legacy systems (Honeywell TPS) experiment. Communication between the DCS wrapper node and the commercial DCS is made through a serial port using MODBUS.

23 The Integrated Control Architecture

The PIKMAC and RiskMan applications described befor (see Chapters 20 and 21) were deployed using an implementation of CORBA middleware specially suited for control purposes. This was the first release of the ICa Broker and it is placed at the cornerstone of the Integrated Control Architecture project.

This is an ongoing, long-term project at the Universidad Politécnica de Madrid, with a basic objective: simplify the construction, deployment and maintenance of software intensive, distributed controllers.

The Integrated Control Architecture follows the specifications developed by OMG for distributed object systems. On top of these specifications, it uses control design patterns [Sanz 03] and class libraries to support these patterns.

The election of CORBA as a basis for the architecture is grounded in its extensibility and the capabilities it offers for real-time and embedded systems [OMG 00, OMG 02]. It is not easy to find these capabilities in other enterprise integration architectures like Microsoft .NET, Java EJB or XML/SOAP.

As an example, the Figure 51 shows how CORBA subsystems running atop heterogeneous protocols (for example TCP/IP and IEC 60870) can be seamless interoperated [Sanz 02].

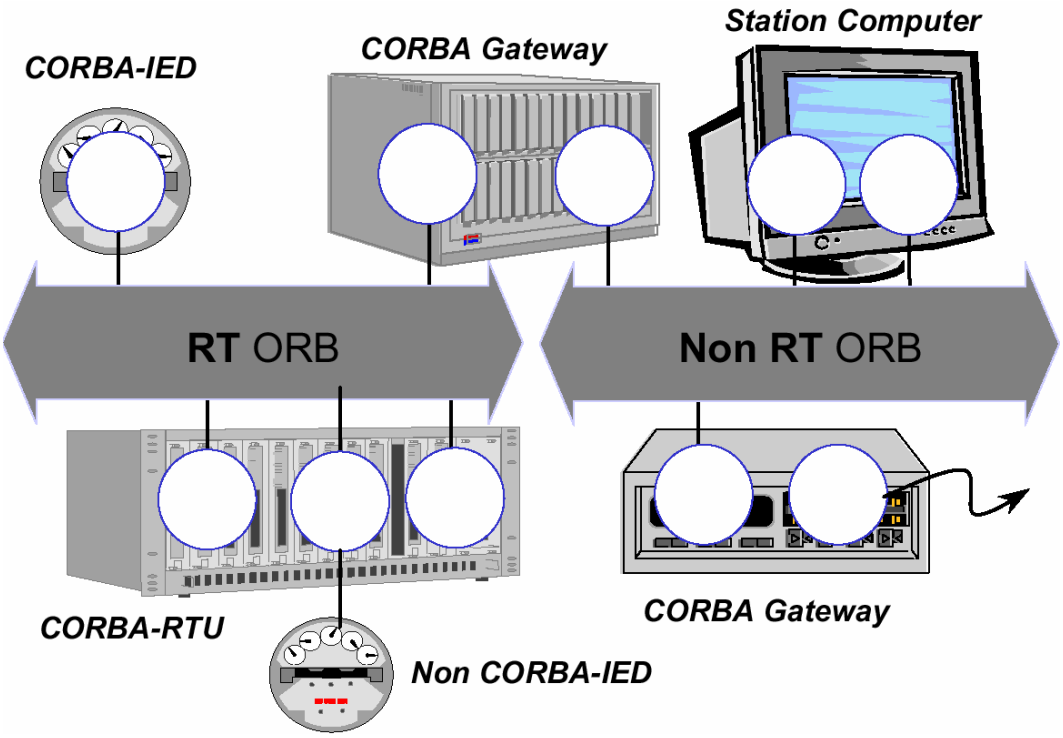


Figure 51: A CORBA-based domain architecture provides the required functionality to deal with the special requirements of the distributed, real-time and embedded domain.

The ICa methodology is strongly based on the use of design patterns. Software pattern technology [Gamma 95, Buschmann 96] is a methodology used for the capture, transfer and exploitation of design knowledge. It has been deeply used in the object-oriented programming community but it has also proved useful in other communities related with the design and implementation of complex real-time systems [Sanz 03].

The development model is based on the use of object frameworks that are specialized to narrower domains to construct complex control product lines [Sanz 99]. The CORBA IDL technology helps in this task because it provides mechanisms of multiple inheritance that simplify the mixing of functionality that is typically required when using multiple design patterns.

Part 6
Additional Materials

This page has been intentionally left blank.

24 Common pitfalls

24.1 Don't start from the beginning

What is most useful of all CORBA technology is the core conceptual model that it provides and that matches so well the distributed controls domain. However, not many people do grasp this strength in its full capacity and offer/perform entries into the CORBA world based just on the fashionable automatic distributed code generation from IDL.

What is important is the object, not the interface. This is clear in the OMA and this should be clear for CORBA developers. Try start reading about the core model and left the tools for the after-understanding phases. Object autonomy should be the central analysis and design target of CORBA solutions.

The Object Management Architecture Guide (OMAG) describes OMG's technical objectives and terminology and provides the conceptual infra-structure upon which supporting specifications are based. The guide includes the OMG Object Model, which defines common semantics for specifying the externally visible characteristics of objects in a standard implementation-independent way, and the OMA Reference Model.

24.2 Overselling of CORBA solutions

CORBA is a good technology for many applications. This has the negative consequence of inappropriate fitness. While the CORBA core object model suits perfectly the modular design of distributed controllers, many engineers fail to understand where CORBA may usefully be applied.

CORBA gets its best in the implementation of control systems when factors concur: distribution, openness, transparency, consistency, heterogeneity, load sharing, etc.

CORBA is a complex technology with educational and implementations barriers for common use. A thorough evaluation should be done before deciding to use it in a concrete application, while, in or experience, the

range of applicability grows when domain engineering is considered a target of the development process.

24.3 Being religious or dogmatic about CORBA

The ideas behind CORBA are common in other technology bases. Some of the available technology in CORBA is obsolete. It is not a revealed truth.

Being religious about the technology only means attaching oneself to suboptimal solutions that do not meet the necessary requierements in control systems. OMG process is a good process for evolution and this is a best focus for dogmatism: make specifications evolve as control domain knowledge is gained.

24.4 Don't know why we want CORBA

Excessive hype has the effect of convincing without arguments. To some extent CORBA has been on the safe side of this problem because there are not good selling practices in this business. However, other related technologies, like agents or Java, do have this effect and in some situations this can influence also CORBA.

CORBA is difficult to understand and master, even while it is a relatively simple thing.

24.5 Being generic for one-of-a-a-kind problems

CORBA is a killer technology in the sense that it can be used for implementing everything (except extreme applications). As was said in section 24.3, CORBA is good when domain engineering is involved but sometimes this is taken too far.

There are situations, i.e. applications, which will not have a future evolution and are not complex enough to require the use of this type of technology. While CORBA objects can be small and targeted, achieving it usually requires a big amount of knowledge and expertise. CORBA objects, by nature, are universally interoperable in the GIOP world, which means that they are usually too generic to be efficient, in particular for one-off applications.

In these systems it is usually much more effective the use of simpler, less universal frameworks, that will require a little platform works that will be compensated by the less effort put into analysis and design phases.

It should not be forgot, however, that when a developer master the technology, the affectivity in building not very constrained applications is maximal if using this technology.

24.6 Belief in silver bullets

There are no silver bullets. CORBA is not a silver bullet (even when it may look like that if we pay attention to what has been written in this document and others).

24.7 Forget that the focus is developing software controllers

It is usually the case that the amount of software that really implements the very control code is less that 1% of the total code. If using CORBA this proportion may reach 0.1 % or less with ease.

Any distributed control system project has its budget and its deadline. The use of CORBA technology may refocus the effort on the CORBA thing instead of the control thing. This is specially critical when the people developing the controller are not CORBA experts but control experts. The CORBA learning barrier is so high that might vampirize other parts of the effort.

Besides the requested focus on core control code, car must be taken to approach the problem of controller construction using the best software practices available. Having good platforms like CORBA do not alleviate the load of doing our work properly.

Also, the flexibility of the technology is so high that people get trapped in some for of digital art, adding distributed features to applications just for the sake of funny added functionality without any traceability to requirements.

24.8 Forget about true physical concurrency

Problems will happen if we forget the fact that we're implementing true multithreaded software. From the inherently multithreasded nature of truly distributed applications to the inner workings of a POA, understanding concurrency and being careful about it is mandatory.

Problems will appear if we are not conscious about the facts that distributed systems are inherently concurrent. Problems will also appear —or to be more precise, will not disappear— if we are unable to do a design that does exploit this concurrency.

24.9 NIH Architectural Syndrome

There are many CORBA-based frameworks for reactive, autonomous, agent-based applications. Each CORBA-based control developer ends building its own **active agent** class. Why?

The NIH[13] Syndrome is well known and is always lurking inside the apparent special requirements from our applications. Beware of innovation in a field so full of working brains.

24.10 Insufficient Intelligence in CORBA Agents

Many things can happen to a CORBA control object. From intrinsic failures of the control code itself to physical actuator faults or global networking or host failures. CORBA objects in control systems cannot (well, should not) be stupid pieces of code.

Fault-tolerant CORBA helps a little handling these problems, but that's not enough. There are many things that should be taken into account when performing actions that are necessary or could be catastrophic. There are many cases of controller that are not safe-critical, but in any case, good engineering can increase the level of intelligence of CORBA objects to reach an adequate trade-off between complexity or run-time impact and the capabilities for autonomous behaviour.

24.11 Excessive Intelligence in CORBA Agents

While intelligence is necessary we should be aware of the risk of excessive intelligence. Using the words of J. Doyne Farmer we can say that CORBA control components should be "smarter than Ping-Pong balls, but not smarter that they need to be."

We can use a complete BDI[14] architecture to implement the CORBA objects that constitute our distributed PID control. That's an obvious

[13] Not Invented Here.
[14] Beliefs, Desires, Intentions.

overkill. Simple objects for simple applications don not need the capability of speaking Ontolingua.

Object-oriented CORBA technology by means of multiple interface and implementation inheritance, somewhat helps solving this problem. The cost we must pay for this feature is making harder the understanding and mastering of the frameworks.

24.12 Seeing Objects/Agents Everywhere

There are cases where objectifying does not help at all. Conventional data-centric applications are good solutions for many applications or particular subsystems. Object wrapping helps in the integration of such heterogeneous pieces of code into cohesive CORBA applications.

24.13 Monolithic Agencies

Beware the big agent. Beware of control systems as single-man agencies. This is another manifestation of the old fat class problem of object orientation. Fat objects are candidates (or better, opportunities) for refactoring.

24.14 All time working in the infrastructure

Project after project we find ourselves repeating, rebuilding some parts of the application (typically integration code). This phenomenon takes engineers into the conviction that it is necessary to work-out the infrastructure once for ever.

Illuminati tend to do work into the *real* issues of middleware, protocols, schedulers and relatives, while the real control application is never thought enough[15]. In some sense this is another manifestation of the NIH Syndrome.

24.15 Insufficient Freedom for Agents

Distributed control systems are distributed. That means that components are inherently independent. Trying to totally tight masters and slaves in such an schema often leads to brain trash paralysis. The central brain is continuously paying attention to details that could be handled in an autonomic way efficiently.

[15] This is not the case of the HRTC Project, which had a specific focus on these issues.

Control policies in distributed control applications are necessarily distributed (unless the platform completely hides the fact that the application is distributed).

This is strongly related with the proper level of intelligence and autonomy that was discussed before.

24.16 Excessive Freedom for Agents

In the same sense, control applications control plants with a single, typically economical, objective. Coordination to achieve the central objective is necessary and object autonomy should necessarily be bounded autonomy.

Part 7
Appendices

This page has been intentionally left blank.

25 References

[Aarsten 96] Amund Aarsten, Davide Brugali and Giuseppe Menga. Designing Concurrent and Distributed Control Systems. Communications of the ACM, October 1996.

[Aarsten 97] Amund Aarsten and Davide Brugali. From Object Orientation to Agent Orientation: Common issues in the development of System Architectures. *ICRA'97*, Albuquerque, USA; 1997.

[Adler 95] Richard M. Adler, Emerging Standards for Component Software. *IEEE Computer*, March 1995.

[Agha 86] Gul Agha. Actors, A Model of Concurrent Computation in Distributed Systems. MIT Press, 1986.

[Agre 95] Philip E. Agre and Stanley S. Rosenschein. Computational Theories of Interaction and Agency. MIT Press, 1995.

[Alarcón 94] Alarcón, M.I., Rodríguez, P., Almeida, L.B., Sanz, R., Fontaine, L., Gómez, P., Alamán, X., Nordin, P., Bejder, H. and de Pablo, E.; Heterogeneous Integration Architecture for Intelligent Control. *Intelligent Systems Engineering*, Autumn 1994.

[Albus] James S. Albus. RCS: A Reference Model Architecture for Intelligent Systems. National Institute of Standards and Technology.

[Allen 96] Realtime CORBA. A White Paper - Issue 1.0. December 5, 1996.

[Astrom 97] Aström, Karl Johan and Björn Wittenmark (1997). Computer Controlled Systems. third ed. Prentice-Hall. New York, NJ.

[Awad 96] Maher Awad, Juha Kuusela and Jurgen Ziegler. *Object Oriented Technology for Real-Time Systems*. Prentice-Hall, 1996.

[Bass 97] Len Bass, Paul Clements, Sholom Cohen, Linda Northrop and James Withey. *Product Line Practice Workshop Report*. CMU/SEI-97-TR-003. June 1997.

[Blanke 01] Blanke, Mogens, Christian Frei, Franta Kraus, Ron J. Patton and Marcel Staroswiecki (2000). Fault tolerant control systems. In: Control of Complex Systems (Aström et al. Eds.). Springer. 2001.

[Boullart 92] Boullart, L., Krijsman, A. and Vingerhoeds, R.A. (Eds) *Application of Artificial Intelligence in Process Control*. Pergamon, 1992.

[Bresciani 2001] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Modeling early requirements in Tropos: a transformation based approach. AOSE workshop at Conf. Agents 2001

[Brownsword 96] Lisa Brownsword and Paul Clements. A Case Study in Successful Product Line Development. CMU/SEI-96-TR-016 October 1996.

[Brugali 98] David Brugali. *From Objects to Agents: Software Reuse for Distributed Systems*. PhD Thesis. Politecnico di Torino, 1998.

[Buschman 96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad and Michael Stal. *Pattern Oriented Software Architecture. A System of Patterns*. Wiley, 1996.

[Davidson 1994] John B. Davidson and David K. Schmidt. Extended Cooperative Control Synthesis. NASA Technical Memorandum 4561. 1994.

[Fischer 94] Fischer, G. Domain-Oriented Design Environments. *Automated Software Engineering, Vol. 1 No. 2*, pp. 177-203, 1994.

[Francez 96] Nissim Francez and Ira R. Forman. Interacting Processes. A Multiparty Approach to Coordinated Distributed Programming. ACM Press-Addison Wesley, 1996.

[Franklin 96] Franklin, S. and Graesser, A. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. Springer-Verlag, 1996

[Fuxman 2001] A. Fuxman, M. Pistore, J. Mylopoulos, P. Traverso. Model Checking Early Requirements Specifications in Tropos. IEEE Int. Symp. on Requirements Engineering, 2001

[Gamma 94] Erich Gamma, Richard Helm, Ralph Jhonson and John Vlissides. *Design Patterns. Elements of Reusable Object Oriented Software*. Addison-Wesley, 1994.

[Garlan 94] Garlan, D., Allen, R. and Ockerbloom, J. Exploiting Style in Architectural Design Environments. *ACM SIGSOFT'94*, USA, 1994.

[Garlan 95a] David Garlan, Robert Allen and John Ockerbloom. Architectural Mismatch: Why Reuse is so Hard. *IEEE Software*, November 1995.

[Giorgini 2001] P. Giorgini, A. Perini, J. Mylopoulos, F. Giunchiglia, P. Bresciani. Agent-Oriented Software Development: A Case Study. Proc. Int. Conf. on Software Engineering & Knowledge Engineering (SEKE01), 2001

[Gupta 96] Gupta, M.M. and N.K. Singh (1996). Intelligent Control Systems. IEEE Press. Piscataway, NJ.

[Gupta 96] M.M. Gupta and N.K. Sinha. *Intelligent Control Systems*. IEEE Press, 1996.

[HINT 94] *HINT Manual for System Developers*. HINT Consortium, 1994.

[ISO 97] ISO/IEC & ITU-T, Open Distributed Processing - Reference Model - Part 1: Overview" ITU-T Rec. X.901 | ISO/IEC IS 10746-1: 1997

[ISO 96] ISO/IEC & ITU-T, Open Distributed Processing - Reference Model - Part 2: Foundations" ITU-T Rec. X.902 | ISO/IEC IS 10746-2: 1996

[ISO 96a] ISO/IEC & ITU-T, Open Distributed Processing - Reference Model - Part 3: Architecture" ITU-T Rec. X.903 | ISO/IEC IS 10746-3: 1996

[ISO 98] ISO/IEC Information technology -- Guideline for the design of programming languages. TR 10176:1998

[Jagannathan 89] Jagannathan, V., Dodhiwala, R., Baum, L.S.; *Blackboard Architectures and Applications*. Academic Press, New York..

[Jalote 94] Pankaj Jalote. *Fault Tolerance in Distributed Systems*. Prentice-Hall, 1994.

[Jennings 94] N.R. Jennings. Cooperation in Industrial Multi-Agent Systems. World Scientific, 1994.

[Kinny 97] David Kinny and Michael Georgeff. Modelling and Design of Multi-Agent Systems. In [Muller 97].

[Klingler] Carol Diane Klingler James Solderitsch.Dagar: A Process For Domain Architecture Definition and Asset Implementation.

[Kolp 2001] M. Kolp, P. Giorgini, J. Mylopoulos. A Goal-Based Organizational Perspective on Multi-Agent Architectures. Proc. Int. Workshop ATAL-2001

[Krutchen 95] Philippe Krutchen. The 4+1 View of Software Architecture. *IEEE Software*, November 1995.

[Lejter 96] Moises Lejter and Thomas Dean. A Framework for the Development of Multiagent Architectures. *IEEE Expert*, December 1996.

[Linden 95] Frank J. Van der Linden and Jürgen K. Muller. Creating Architectures with Building Blocks. *IEEE Software*, November 1995.

[Luyben 90] William H. Luyben. Process Modeling, Simulation and Control for Chemical Engineers. McGraw-Hill. 1990.

[Maffeis 97] S. Maffeis and D.C. Schmidt. Constructing Reliable Distributed Communication Systems with CORBA. *IEEE Comunications Magazine, Vol. 14, No.2*. 1997.

[Medvidovic 97] N. Medvidovic. A Classification and Comparison Framework for Software Architecture Description Languages. Technical Report, UCI-ICS-97-02, University of California, Irvine, January 1997.

[Monroe 97] Robert T. Monroe, Andrew Kompanek, Ralph Melton and David Garlan. Architectural Styles, Design Patterns and Objects. *IEEE Software* Jan/Feb 1997.

[Mowbray 97] Thomas J. Mowbray and Raphael C Malveau. *CORBA Design Patterns*. Wiley, 1997.

[Musliner 96] David J. Musliner & Christopher A. Miller. Agent and Task Modeling at Honeywell. Working Notes of *the AAAI Workshop on Agent Modeling*. Portland, Oregon, August 1996, pp. 112-118.

[Ogata 90] Katsuhiko Ogata. *Modern Control Engineering*. Prentice Hall, 1990.

[OMG 96] *An Overview of the OMA*. Object management Group.

[OMG 96] Realtime CORBA. A White Paper. OMG Realtime SIG. Object Management Group, 1996.

[OMG 97] *UML Notation Guide*, version 1.1 (1 September 1997), The Object Management Group, doc. no. ad/97-08-05.

[OMG 98] Common Object Request Broker Architecture and Specification, Ver. 2.2. Object Management Group, 1998.

[OMG 98] Real-Time CORBA. Joint Revised Submission Document Number orbos/1998-12-10, Object Management Group, Needham, MA, U.S.A., December 1998. Available at http://doc.omg.org/orbos/1998-12-10.

[OMG 01a] CORBA 3.0 New Components Chapters. Updated CCM specification Document Number ptc/2001-11-03, Object Management Group, Needham, MA, U.S.A., November 30, 2001. Available at http://doc.omg.org/ptc/2001-11-03.

[OMG01b] Real-Time CORBA 2.0: Dynamic Scheduling Specification. Final Adopted specification Document Number ptc/2001-08-34, Object Management Group, Needham, MA, U.S.A., August 2001. Available at http://doc.omg.org/ptc/2001-08-34.

[OMG 02] Common Object Request Broker Architecture and Specification. Release 3.0. Object Management Group. Falls Church, USA.

[OMG02a] Enhanced View of Time V1.1. Available Specification Document Number formal/2002-05-07, Object Management Group, Needham, MA, U.S.A., May 2002. Available at http://doc.omg.org/formal/2002-05-07.

[OMG 02b] Fault Tolerant CORBA. Available Specification Document Number formal/2002-06-59, Object Management Group, Needham, MA, U.S.A., May 2002. Available at http://doc.omg.org/formal/2002-06-59.

[OMG 03a] Extensible Transport Framework. Revised Submission Document Number mars/2003-02-01, Object Management Group, Needham, MA, U.S.A., March 3, 2003. Available at http://doc.omg.org/mars/2003-02-01.

[OMG 03b] Data Distribution Service submission. Joint Submission Document Number mars/2003-03-16, Object Management Group, Needham, MA, U.S.A., March, 2003. Available at http://doc.omg.org/mars/2003-03-16.

[OMG 03c] Smart Transducers Interface V1.0. Available Specification Document Number formal/2003-01-01, Object Management Group, Needham, MA, U.S.A., January 2003. Available at http://doc.omg.org/formal/2003-01-01.

[OMG 03d] Unified Modeling Language V1.5. Available Specification Document Number formal/2003-03-01, Object Management Group, Needham, MA, U.S.A., March 2003. Available at http://doc.omg.org/formal/2003-03-01.

[Otte 96] Randy Otte, Paul Patrick and Mark Roy. *Understanding CORBA*. Prentice Hall PTR. 1996.

[Parunak 03] Parunak, H. Van Dyke, Sven Brueckner, Mitch Fleischer and James Odell. A Preliminary Taxonomy of Multi-Agent Interactions. Second International Conference on Autonomous Agents and Multi-Agent Systems, AAMAS'03. 2003.

[Perini 2001] A. Perini, P. Bresciani, F. Giunchiglia, P. Giorgini and J. Mylopoulos, A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. Proc. Int. Conf. Agents 2001

[Peterson 94] A. Spencer Peterson and Jay L. Stanley Jr. Mapping a Domain Model and Architecture to a Generic Design. Technical Report CMU/SEI-94-TR-8. May 1994.

[POSIX.21 95] Interface Requirements for Real-Time Distributed Systems Communication. IEEE POSIX.21 Working Group, 1995.

[Regev 02] G. Regev, A. Wegmann, UML for Early Requirements Elicitation: A Regulation based Approach 1/15 EPFL-IC Technical report no. IC/2002/013.

[Rodriguez 99] Rodríguez, Manuel and Ricardo Sanz (1999). HOMME: A modeling environment to andel complexity. In: IASTED Modeling and Simulation Conference.

[Rushby 99] Rushby, John (1999). Partitioning in avionics architectures: Requirements, mechanisms, and assurance. Technical Report NASA/CR-1999-209347. NASA.

[Samad 00] Samad, Tariq and Weyrauch, John, Eds. (2000). Automation, Control, and Complexity: New Developments and Directions. John Wiley and Sons. Chichester, UK.

[Samad 98] Samad, Tariq (1998). Complexity management: Multidisciplinary perspectives on automation and control. Technical Report CON-R98-001. Honeywell Technology Center. Minneapolis, USA.

[Sanz 00] Sanz, Ricardo (2000). Agents for complex control systems. Chap. 10, pp. 171–190. In: Samad and Weyrauch (2000).

[Sanz 91] Sanz, R., A.Jiménez, R.Galán, F.Matía and E.A.Puente. Intelligent Process Control: The CONEX Architecture. In *Engineering Systems with Intelligence.* S. Tzafestas (Ed.). Kluwer Academic Publishers, 1991.

[Sanz 94] Sanz, R., R.Galán, A.Jiménez, F.Matía, J.Velasco and G.Martínez. Computational Intelligence in Process Control. *ICNN'94, IEEE International Conference in Neural Networks*. Orlando, USA, 1994.

[Sanz 96] Sanz, R., F.Matía, R.Galán and A. Jiménez. Integration of Fuzzy Technology in Complex Process Control Systems. *FLAMOC'96*. Sydney, Australia, 1996.

[Sanz 98] Sanz, Ricardo et al. Progressive Domain Focalization in Intelligent Control Systems. *IFAC 5th Workshop on Algorithms and Architectures for Real Time Control.* Cancún, Mexico, 1998.

[Sanz 99a] 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.

[Sanz 99b] Sanz, Ricardo, Miguel J. Segarra, Angel de Antonio and José A. Clavijo (1999b). ICa: Mid-dleware for intelligent process control. In: IEEE International Symposium on Intelligent Control, ISIC'1999. Cambridge, USA.

[Sanz 99c] Sanz, Ricardo, Miguel J. Segarra, Angel de Antonio, Fernando Matía, Agustín Jiménez and Ramón Galán (1999c). Patterns in intelligent control systems. In: Proceedings of IFAC 14th World Congress. Beijing, China.

[Sanz 03] Sanz, Ricardo and Janusz Zalewsky. Control Engineering using Design Patterns. IEEE Control Systems Magazine, June 2003.

[Schmidt 96] Douglas C. Schmidt, Ralph E. Johnson and Mohamed Fayad. Software Patterns. *Communications of the ACM, Vol. 39, No. 10*, October 1996.

[Selic 94] Bran Selic, Garth Gullekson and Paul T. Ward. *Real-Time Object Oriented Modelling*. Wiley, 1994.

[Selic 96] Bran Selic & Garth Gullekson. Design Patterns for Real-Time Software. *Embedded Systems Conference West '96*. 1996.

[Selic 98] Bran Selic and Jim Rumbaugh. Using UML for Modeling Complex Real-Time Systems. March 11, 1998

[Sha 93] Sha, L. and Sathaye, S.S. A Systematic Approach to Designing Distributed Real-Time Systems. *IEEE Computer*, Vol.26, No.9, 1993.

[Sha 95] Sha, Lui; Rajkumar, Ragunathan; Gagliardi, Michael. A Software Architecture for Dependable and Evolvable Industrial Computing Systems. Technical report CMU/SEI-95-TR-005. 1995.

[Shaw 96] Shaw, Mary and David Garlan (1996). Software Architecture. An Emerging Discipline. Pren-tice- Hall. Upper Saddle River, NJ.

[Shokri 00] Shokri, Eltefaat and Phillip Sheu (2000). Real-time distributed object computing: An emerging field. IEEE Computer pp. 45–46.

[Siegel 00] Siegel, Jon (2000). CORBA 3: Fundamentals and Programming. 2nd ed. OMG Press/Wiley. New York.

[Siegel 96] Jon Siegel. *CORBA Fundamentals and Programming*. Wiley 1996.

[Siljak 91] Dragoslav Siljak. *Decentralized Control of Complex Systems*. Academic Press, 1991.

[Svoboda] Svoboda, Frank. The Three "R's" of Mature System Development: Reuse, Reengineering, and Architecture [online]. Available WWW <http://source.asset.com/stars/darpa/Papers/ArchPapers.html> (1996).

[van der Linden 95] Frank van der Linden and Jürgen Müller. Creating Architectures with Building Blocks. *IEEE Software* 1995.

[White 92] White, D.A., and Sofge, D.A. (Eds) *Handbook of Intelligent Control*. Van Nostrand-Reinhold, 1992.

[Wickman] Grant R. Wickman, Dr James Solderitsch and Mark Simos.A Systematic Software Reuse Program Based on an Architecture-Centric Domain.

[Wielinga 94] Bob Wielinga, Guus Schreiber, Wouter Jansweijer, Anjo Anjewierden and Frank van Harmelen, Framework and Formalism for Expressing Ontologies. University of Amsterdam. KACTUS DO1b.1. 1994.

[Wittig 92] Wittig, T. (Ed). *ARCHON, an Architecture for Multi-Agent Systems*. Ellis Horwood, 1992.

[Wooldridge 95] M.J. Wooldridge and N.R.Jennings. *Intelligent Agents*. Springer Verlag 1995.

[Wooldridge 96] Michael Wooldridge, Jörg P. Müller and Milind Tambe. *Intelligent Agents II*. Springer Verlag 1996.

[Wooldridge 00] Michael Wooldridge, Nicholas R. Jennings and David Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems, 3, 285-312, 2000.

26 OMG Specification Catalog

This is the list of OMG specifications as of the date of closing of this document. An updated list can be obtained from:

http://www.omg.org/technology/documents/spec_summary.htm

Specification category	Domain	Current	Document #	Past
OMG Modeling Specifications				
Common Warehouse Metamodel (CWM™)	data warehousing, modeling	1.1	formal/2003-03-02 (volume 1)	1.0
Common Warehouse Metamodel (CWM™) Metadata Interchange Patterns (MIPS)	data warehousing, modeling	1.0 finalization	ptc/2002-12-01	n/a
Meta-Object Facility (MOF™)	modeling	1.4	formal/2002-04-03	1.3
Software Process Engineering Metamodel (SPEM)	modeling	1.0	formal/2002-11-14	n/a
Unified Modeling Language™ (UML™)	modeling	1.5	formal/2003-03-01	1.4 and Action Semantics
UML Human-Usable Textual Notation (HUTN)	modeling	1.0 finalization	ptc/2003-06-05	n/a
UML™ Profile for CORBA®	modeling	1.0	formal/2002-04-01	n/a
UML™ Profile for Enterprise Application Integration (EAI)	modeling	1.0 finalization	ptc/2003-02-01	n/a
UML™ Profile for Enterprise Distributed Object Computing (EDOC)	modeling	1.0 finalization	ptc/2002-02-05	n/a
UML™ Profile for Schedulability, Performance and Time	modeling	1.0 finalization	ptc/2003-03-02	n/a

UML™ Testing Profile	modeling	1.0 finalization	ptc/2003-07-01	n/a
XML Metadata Interchage (XMI®)	modeling	1.2	formal/2002-01-01	1.1
	modeling	2.0	formal/2003-05-02	1.1
CORBA/IIOP Specifications				
Common Object Request Broker Architecture (CORBA/IIOP)	middleware	3.0.2	formal/2002-12-02	3.0.1
Common Secure Interoperability (CSIv2)	security, middleware	3.0.2	Chapter 24 of CORBA/IIOP 3.0.2	Chapter 24 of CORBA/IIOP 2.6
CORBA Component Model	middleware, components	3.0	formal/2002-06-65	n/a
CORBA-FTAM/FTP Interworking	middleware	1.0	formal/2002-03-13	n/a
CORBA / TC Interworking and SCCP-Inter ORB Protocol	middleware	1.0	formal/2001-01-01	n/a
CORBA-WSDL/SOAP Interworking	middleware	1.0 finalization	ptc/2003-05-15	n/a
Deployment and Configuration of Component-based Distributed Applications	middleware	1.0 finalization	ptc/2003-07-02	n/a
Fault Tolerance	middleware	3.0.2	Chapter 23 of CORBA/IIOP 3.0.2	Chapter 23 of CORBA/IIOP 3.0.1
Firewall Traversal	middleware	1.0 finalization	ptc/2003-01-13	n/a
GIOP SCTP	middleware, telecommunications	1.0 finalization	ptc/2003-08-20	n/a
Interworking between CORBA and TMN Systems	middleware, telecommunications	1.0	formal/2000-08-01	n/a
Online Upgrades	middleware	1.0 finalization	ptc/2003-03-03	n/a
Wireless Access & Terminal Mobility in CORBA (Telecom Wireless)	middleware, telecommunications	1.0	formal/2003-03-64	n/a
WSDL/SOAP-CORBA Interworking	middleware	1.0 finalization	ptc/2003-07-04	n/a
CORBA Security Specifications				
Authorization Token Layer Acquisition Service (ATLAS)	security, middleware	1.0	formal/2002-10-01	n/a
Common Secure	(see			

Interoperability (CSIv2)	CORBA/IIOP Specifications)			
Security Service	(see CORBAservices Specifications)			
Resource Access Decision Facility	(see OMG Domain Specifications)			
IDL / Language Mapping Specifications				
Ada	software development	1.2	formal/2001-10-42	1.1
C	software development	1.0	formal/99-07-35	n/a
C++	software development	1.0	formal/99-07-41	n/a
COBOL	software development	1.0	formal/99-07-47	n/a
CORBA Scripting Language	software development	1.1	formal/2003-02-01	1.0
IDL to Java	software development	1.2	formal/2002-08-05	1.1
Java to IDL	software development	1.2	formal/2002-08-06	1.1
Lisp	software development	1.0	formal/2000-06-02	n/a
PL/1	software development	1.0	formal/2002-09-05	n/a
Python	software development	1.2	formal/2002-11-05	1.1
Smalltalk	software development	1.0	formal/99-07-65	n/a
XML	software development	1.1	formal/2003-04-01	1.0
Specialized CORBA Specifications				
Data Distribution	real-time, middleware	1.0 finalization	ptc/2003-07-07	n/a
Data Parallel Processing	real-time, middleware	1.0 finalization	ptc/2003-03-05	n/a
Dynamic Scheduling	real-time, middleware	1.0 finalization	ptc/2002-09-14	n/a
Lightweight Logging Service	real-time, middleware, telecommunications	1.0 finalization	ptc/2003-05-22	n/a
Minimum CORBA	real-time,	1.0	formal/2002-08-01	n/a

	middleware			
Online Upgrades	(see CORBA/IIOP Specifications)			
Real-Time CORBA Architecture	real-time, middleware	1.1	formal/2002-08-02	Chapter 24 of CORBA/IIOP 2.5
Unreliable Multicast	real-time, middleware	1.0 finalization	ptc/2003-01-11	n/a
CORBA Embedded Intelligence Specifications				
Smart Transducers	real-time, embedded systems	1.0	formal/2003-01-01	n/a
CORBAservices Specifications				
Additional Structuring Mechanisms for the OTS	transaction mgmnt, middleware	1.0	formal/2002-09-03	n/a
Collection Service	collection mgmnt, middleware	1.0.1	formal/2002-08-03	1.0
Concurrency Service	object consistency, middleware	1.0	formal/2000-06-14	n/a
Enhanced View of Time	time mgmnt, middleware	1.1	formal/2002-05-07	1.0
Event Service	event mgmnt, middleware	1.1	formal/2001-03-01	1.0
Externalization Service	object state mgmnt, middleware	1.0	formal/2000-06-16	n/a
Licensing Service	software licensing, middleware	1.0	formal/2000-06-17	n/a
Life Cycle Service	object life cycle mgmnt, middleware	1.2	formal/2002-09-01	1.1
Management of Event Domains	event mgmnt, middleware	1.0	formal/2001-06-03	n/a
Naming Service	object location mgmnt, middleware	1.2	formal/2002-09-02	1.1
Notification Service	event mgmnt, middleware	1.0.1	formal/2002-08-04	1.0
Notification / JMS Interworking	event mgmnt, middleware	1.0 finalization	dtc/2003-06-01	n/a

Persistent State Service	object persistence, middleware	2.0	formal/2002-09-06	replaces Persistent Object Service
Property Service	object properties, middleware	1.0	formal/2000-06-22	n/a
Query Service	collection mgmnt, middleware	1.0	formal/2000-06-23	n/a
Relationship Service	object relationships, middleware	1.0	formal/2000-06-24	n/a
Security Service	security, middleware	1.8	formal/2002-03-11	1.7
Telecoms Log Service	(see OMG Domain Specifications)			
Time Service	time mgmnt, middleware	1.1	formal/2002-05-06	1.0
Trading Object Service	object location mgmnt, middleware	1.0	formal/2000-06-27	n/a
Transaction Service	transaction mgmnt, middleware	1.3	formal/2002-08-07	1.2.1
CORBAfacilities Specifications				
Internationalization and Time	software development	1.0	formal/2000-01-01	n/a
Mobile Agent Facility	software development	1.0	formal/2000-01-02	n/a
OMG Domain Specifications				
Air Traffic Control	transportation	1.0	formal/2000-05-01	n/a
Audio / Visual Streams	telecommunications	1.0	formal/2000-01-03	n/a
Bibliographic Query Service	life sciences research	1.0	formal/2002-05-03	n/a
Biomolecular Sequence Analysis (BSA)	life sciences research	1.0	formal/2001-06-08	n/a
Clinical Observations Access Service (COAS)	healthcare	1.0	formal/2001-04-06	n/a
Computer Aided Design (CAD) Services	manufacturing & utilities	1.1	formal/2003-03-63	1.0
CORBA-FTAM/FTP Interworking	(see CORBA/IIOP Specifications)			
CORBA / TC Interworking and SCCP-Inter ORB	(see CORBA/IIOP			

Protocol	Specifications)			
Currency	finance	1.0	formal/2000-06-29	n/a
Data Acquisition from Industrial Systems (DAIS)	manufacturing & utilities	1.0	formal/2002-11-07	n/a
Distributed Simulation Systems	simulation	2.0	formal/2002-11-11	1.1
Federated Charging	telecommunications	1.0 finalization	dtc/2003-01-01	n/a
General Ledger	finance	1.0	formal/2001-02-67	n/a
Gene Expression	life sciences research	1.1	formal/2003-10-01	1.0
Genomic Maps	life sciences research	1.0	formal/2002-02-01	n/a
GIOP Tunneling over Bluetooth	telecommunications	1.0 finalization	dtc/2003-05-06	n/a
Historical Data Acquisition from Industrial Systems (HDAIS)	manufacturing & utilities	1.0 finalization	dtc/2003-02-01	n/a
Interworking between CORBA and TMN Systems	(see CORBA/IIOP Specifications)			
Laboratory Equipment Control Interface Specification (LECIS)	life sciences research	1.0	formal/2003-03-19	n/a
Lexicon Query Service	healthcare	1.0	formal/2000-06-31	n/a
Lightweight Logging Service	(see Specialized CORBA Specifications)			
Macromolecular Structure	life sciences research	1.0	formal/2002-05-01	n/a
Management of Event Domains	telecommunications	1.0	formal/2001-06-03	n/a
Negotiation Facility	electronic commerce	1.0	formal/2002-03-14	n/a
Notification / JMS Interworking	(see CORBAservices Specifications)			
Organizational Structure (OSF)	cross-domain	1.0 finalization	dtc/2001-09-04	n/a
Party Management Facility	finance	1.0	formal/2001-02-68	n/a
Person Identification Service (PIDS)	healthcare	1.1	formal/2001-04-04	1.0
PIM and PSM for SDO	cross-domain	1.0 finalization	dtc/2003-04-02	n/a
Product Data Management (PDM) Enablers	manufacturing & utilities	1.3	formal/2000-11-11	1.2
Public Key Infrastructure (PKI)	electronic commerce, security	1.0	formal/2002-09-04	n/a
Resource Access Decision (RAD)	healthcare, security	1.0	formal/2001-04-01	n/a

Surveillance User Interface (Surveillance Manager)	transportation	1.0	formal/2003-03-62	n/a
Task and Session	cross-domain	1.0	formal/2000-05-03	n/a
Telecoms Log Service	telecommunications	1.1.2	formal/2003-06-01	1.1.1
Telecom Service & Access Subscription (TSAS)	telecommunications	1.0	formal/2002-12-01	n/a
Telemetry and Telecommand Data (XTCE)	space	1.0 finalization	dtc/2003-05-07	n/a
Utility Management Systems (UMS) Data Access Facility	utility management	2.0	formal/2002-11-08	1.0
Wireless Access & Terminal Mobility in CORBA (Telecom Wireless)	(see CORBA/IIOP Specifications)			
Workflow Management Facility	cross-domain	1.2	formal/2002-05-02	n/a

27 Final Comments

27.1 Some final thoughts

CORBA technology is impressive for people writing controller code. But perhaps it is too impressive for normal control systems developers.

In some sense it suffers what has been called a *second system* effect, trying to address any possible functionality or requirement. As control engineers, we must clearly 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 *Why do we need integration?*. Beyond many obvious answers (to build TotalPlants, to achieve total safety, to be the first in the market, to spend less money, etc.) we 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, and this will eventually lead to reach enormous complexity levels as those found in human minds. For sure CORBA will not be the integration technology for future C3POs, but it will open the way. If you remember the movie 2010, HAL 9000 goes back to life (or conscience) when Dr. Chandra reconnects the modules that encapsulate high-level mental functions using an integrational backbone.

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 unnecessarily collapse design dimensions of the controller design space [Shaw 96]. This is, for example, the case of some fieldbus technologies that support several slaves but only *one* master. While design restrictions simplify development (by means of prerequisite design decisions) 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, design; but the underlying field bus mechanism should allow for alternative, multi-master designs. This can be done by means of the development of agent libraries that provide predefined partial designs in the form of design patterns [Sanz 03], and a transparent object-oriented real-time middleware. CORBA is a less-commitment approach, so to say.

This approach will let complex control system developers construct their own agencies to support their own designs because it is impossible to fight the *not-invented-here* syndrome. Let the people do what they think that they need. Do not define *ultimate* solutions. Provide reusable assets that can be adapted to any problem in a progressively focused domain [Sanz 99].

27.2 A lot of work to be done

This handbook is a first attempt at providing a compendium of necessary knowledge to effectively apply CORBA technology in the domain of industrial control systems.

As any first attempt it is full of problems, errors and missing items. Much work needs to be done in future releases to make it a valuable tool for the complex distributed control engineer.

Sample topics to be addresses in future releases:

- UML Profile for Schedulability, Performance and Time
- UML CORBA Profile
- Quality assurance practices
- CORBA control system validation
- Agent UML
- DAIS and HDAIS
- Lightweigth CORBA Component Model
- CORBAsec and MLS
- Relations with other standardization bodies (ISO, IEEE, ITU)