Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (0223) 323010**
INTERNATIONAL: **+44 223 323010**
FAX: **+44 223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

An Open Architecture for Real-Time Processing

Guangxing Li and Dave Otway

APM, Poseidon House, Castle Park, Cambridge, UK CB3 0RD

Abstract

This paper describes the ANSA work-in-progress on an open distributed real-time systems architecture. It examines the problem space and technology bases of open distributed real-time processing. An integrated system architecture is suggested and the benefits of the architecture are presented. It then goes on to outline the important progress of the ANSA phase 3 project to extend the ANSA architecture for real-time and multimedia processing. Work is currently under way to build an ANSA real-time platform.

APM.1270.01 **Approved** 12 August 1994

External Paper

Distribution:
Supersedes:
Superseded by:

An Open Architecture for Real-Time Processing

1 Motivation and Benefits

Open distributed processing is concerned with the use of commodity technology to build integrating applications that link together existing applications, databases, control systems and users.

Real-time processing is concerned with the timeliness of computing activities.

The need for an integrated open system architecture for real-time processing is driven by two technology trends:

- *general purpose distributed computing environments are evolving towards real-time systems.* For example, the advances in digital communication networks and in personal computer workstations are beginning to allow the generation, communication and presentation of real-time voice and video media simultaneously. Many non-real-time systems have been disembowelled to extend their use to real-time processing [Leung 90]. This trend requires distribution and real-time control functionality to be intrinsic elements of the system. *There is a great demand to provide real-time functionality as normal system services, rather than as special add on features.*
- *real-time applications are evolving towards large distributed systems.* One-million-line real-time software systems in telecomms, manufacturing, transportation and other application areas are becoming common today [Gopinath 93]. Such systems are large by any standard and inevitably distributed. Therefore, in addition to the problems associated with real-time operation, such applications are subject to all of the problems of any large software system, such as maintainability and distribution. Furthermore, in many real-time applications, tight real-time constraints may apply to only part of the whole system. For example, it is estimated that only 10 to 30 percent of a typical vehicle control software system is directly related to actual real-time control of the vehicle. *There is an increasing need to adopt an open and architectural approach so that*

real-time software engineering can be augmented with other techniques to address evolution, scale, distribution and other issues.

An integrated system architecture provides the capability to treat all forms of real-time objects as *first class citizens* in a system environment. That is, operations and mechanisms provided for existing non-real-time components can be applied to, and used by, real-time components. The provision of a uniform system environment facilitates increased productivity, especially for applications which offer combinations of distributed and real-time functionality: e.g. multimedia conference, distributed control. Increased integration allows existing distributed system mechanisms (such as trading, security, monitoring, replication, location, migration and federation) to be applied to real-time components. An integrated system architecture also allows evolution of systems from the development of individual real-time systems, to groups of real-time systems and then to *enterprise-wide* command and control real-time systems.

Current reference models for open distributed processing, including ISO RM-ODP [ISO 93], OSI Management, OMG Object Management Architecture (OMA) [OMG 92] and ANSA [Herbert 94], make no mention of performance or real-time issues.

Current standards for open distributed processing, such as the Open Software Foundation's Distributed Computing Environment (DCE) and the Object Management Group's Common Object Request Broker Architecture (CORBA) do not address real-time or performance management requirements. As relatively new technologies, attention has focused entirely on functionality; real-time issues are not addressed.

Therefore the scope of the ANSA Phase 3 work on performance and real-time is to define a framework for distributed real-time computing and to investigate candidate technology for practical standardization.

This work will offer several benefits:

- service providers will be able to specify and implement services that give required performance and real-time guarantees
- except in the most demanding of situations, real-time and performance management will cease to be special cases, decreasing the costs and time to deployment for such systems
- open distributed processing technology will be able to support performance and real-time guarantees, increasing the range of

applications and services it can support, and increasing the market for the technology.

2 Desirable Features for Real-Time Systems

Instead of defining what is a real-time system (for which there is hardly a widely accepted definition), we list the desirable features of real-time systems.

2.1 Predictability

Predictability is the capability of a system to perform a set of operations in a well-defined, or determined fashion, such that each operation's timing requirements are satisfied.

A fully predictable system performs operations in the same amount of time, every time, independent of surrounding conditions such as system load.

Predictability applies to every level of the components of a real-time distributed system environment. Such an environment must provide a certain degree of predictability, even though it is not always possible to be fully predictable, if it is to support any useful performance guarantees.

2.2 Programmer Control

Many real-time applications are embedded systems (which often have static loads), therefore it is possible to control the system's behaviour. On the other hand, real-time applications have immense behaviour diversity, therefore fixed system behaviour cannot cater for many real-time application requirements. Consequently, it is required that a programmer has ultimate control of the behaviour of the system.

The simplest method of programmer control of system behaviour is probably the choice of priorities for real-time activities. By indicating the relative priorities of activities, a programmer can affect throughput and/or responsiveness goals for the system on a much finer granularity than by a *do the best you can* approach. Programmers may also be allowed to select the scheduling policy, pre-allocation of system and application resources to critical services and so on.

2.3 Timeliness

The correctness of a real-time system depends not only on the functional behaviour of the system, but also depends on its temporal behaviour. A real-time system environment must provide mechanisms which take these time

related issues into account and must help application programs to meet these timing constraints. A simple example is to allow an application to associate deadlines with real-time activities, and to have the system employ a deadline based scheduling policy to ensure deadlines are met or to cancel obsolete operations when missed deadlines are identified. Other required functions include the description and enforcement of temporal relations among related computational activities.

2.4 Mission Orientation

A mission oriented distributed computer system is one whose functions are dedicated to the fulfilment of a specific goal through the cooperative execution of one or more application programs distributed across its nodes. In the real-time sense, mission orientation also means *mission critical* -- the degree of mission success is strongly correlated with the extent to which the overall system can achieve the maximum dependability regarding real-time constraints. In its simplest form, mission orientation requires that a priority or deadline associated with a mission has global meaning when it spans the network. More generally, global importance and urgency characteristics are propagated through the system for use in resolving contention over system resources according to application defined policies.

2.5 Performance

Real-time applications often have stringent raw performance requirements. Consequently, the optimized integration of application software and its supporting environment is necessary. This is in contrast with the popular layered design approach for non real-time applications. Also, real-time applications often require to trade off modularity, flexibility and functionality in order to maximise performance.

3 Technologies

This section is structured as follows:

- a description of the fundamental contributory technologies
- a review of functions in an open distributed system environment
- a brief description of the current state of art of distributed real-time system environment research and engineering, and the additional functions required in such an open, real-time, distributed architecture.

3.1 Contributory Technologies

The fundamental contributory technologies are illustrated in Figure 1. It represents the integration of real-time systems, open systems and object oriented systems.

Note: Figure 1 comes here.

Real-time system technology provides the functionality of resource management for guaranteeing the stringent time-constrained computing activities.

Open system technology provides the functionality for distribution, evolution, heterogeneity, federation and scale.

Object oriented technology provides the functionality for software reuse and maintenance.

3.2 Distributed System Environments

A distributed system environment is a run-time system that provides a set of abstractions and tools to support the writing of distributed programs. The principal benefit of using a distributed system environment is that applications are automatically supported by a run-time environment which incorporates a set of *distribution transparency* mechanisms. These transparencies shield application designers and users from the technological complexities underpinning distributed application programs. Remote Procedure Call (RPC) and client-server interactions are widely accepted as distributed system environment technical apparatus.

It is now recognised [Herbert 91] that distribution transparency can be broken down into a number of individual transparency issues:

- location transparency --- masking off the physical location of services
- access transparency --- masking any differences in representation and operation invocation mechanism
- concurrency transparency --- masking overlapped execution
- replication transparency --- masking redundancy
- failure transparency --- masking recovery of services after failures
- resource transparency --- masking changes in the representation of a service and the resources used to support it
- migration transparency --- masking movement of a service from one location to another

- federation transparency --- masking administrative and technology boundaries.

3.3 Real-Time Distributed System Environments

Despite the relative maturity of distributed system environment research, real-time distributed system environment remains a neglected, if not unaddressed, topic. Consequently, even if base technologies (such as microkernel, ATM networks etc.) can provide real-time services, a distributed system environment provides no corresponding abstractions to use these services. Even worse, a distributed system environment often masks off the real-time features of base technologies. Therefore, one of the main aims of this work is to extend the real-time features of base technologies to the distributed system environment level.

Note: Figure 2 comes here.

One common misconception is that a distributed system environment is not suitable for real-time applications because RPC (one of the main technologies in a distributed system environment) is often criticized for providing poor performance or for not being fast enough. This is a misconception because the objective of real-time computing is to meet the timing requirements of an application, rather than to be just fast. The most important property of a real-time system is *predictability*. Moreover, fast is a relative term. As technology progresses, there will be faster and faster RPC systems. For example, there are already reports of systems that can provide hundreds of microseconds level RPC calls [Biagioni 93] [Johnson 93]. Fast computing is helpful in meeting stringent timing constraints, but fast computing alone does not bring real-time properties.

A real-time system must be able to handle time-constrained processing of requests. A real-time distributed system environment adds another dimension to the problem of distributed system environment, since the concern is now not only with the functional correctness, but also with the timeliness of the results produced. In Figure 2, a graphical illustration of the real-time distributed system environment functionality is given. The curve in the figure illustrates that the real-time distributed system environment functionality are the trade-off of the real-time functionality and distributed system environment functionality. This reflects the fact that real-time functionality and distributed system environment functionality are often conflicting goals. For example, most distribution transparencies (such as RPC protocols) are based on *time*

redundancy technologies. Such technologies need to be revised for real-time applications.

4 Target

Real-time systems span a wide variety of field of applications, including military, industry, commerce, medicine and so on. This indicates a wide spectrum of possible problems.

The scope of this research for real-time applications is *supervisory control* [Northcutt 88] as opposed to low-level, synchronous sampled data loop functions like sensor/actuator feedback control, signal processing, priority interrupt processing and so on.

Note: Figure 3 comes here.

Supervisory control is a middle-level function (see Figure 3), above the local control and data acquisition functions and below the human interface management functions. This type of system does not do much direct polling of sensors and manipulation of actuators, nor does it provide extensive man machine interfaces; rather, it deals with subsystems which provide these functions. The real-time response requirements of a supervisory control system are closer to the millisecond than either the microsecond or second ranges.

5 Designing for Real-Time and Multimedia Processing

The core of ANSA architecture is its Computational Model [Rees 93] and Engineering Model [ISO 93]. This section outlines some important progress of the ANSA Phase 3 project in extending the two models for real-time and multimedia processing. The focus here is *why* the extensions are needed rather than *what* are the achieved technical results. It is difficult to deal with such a wide-ranging subject in a short article. Readers who would like to read more can find details in the references.

This section first briefly describes the ANSA object model, and then discusses streams, signals, synchronous computing, explicit binding, Quality of Service (QoS) framework, and real-time programming model issues.

5.1 ANSA Objects

The ANSA computational model uses *objects* as units of distribution for management and replacement. An object has one or more *interfaces* that are the points of provision and use of services. Interfaces are first class entities in their own right and references to them may be freely passed around the system.

An interface contains a set of named *operations* (i.e. procedures or methods) which defines its type. Interfaces have the usual remote procedure call style of interaction: operations are invoked with a set of arguments and a response is returned. Arguments and results to invocations consist of references to other interfaces. The effect of an interaction is that the client and server share access to the argument and result interfaces. This model makes each interface an abstract data type.

5.2 Streams

The traditional ANSA object interaction model is asymmetric: any client holding an interface reference can invoke a server object. The server does not have the control over which clients may access it.

In terms of communication paradigm, the asymmetric interaction model is a many-to-one model. The server can only reply to a request, and cannot initiate any communication. This is not sufficient for multimedia and real-time processing where end-to-end (one-to-one) communication is essential.

The support of communication endpoints in real-time ANSA objects is provided by ***stream interfaces*** [Otway 94]. Streams consist of uni-directional data flows. Flows consist of application dependent frames. The source of a data flow must be bound to its sink explicitly before data can pass along it. Streams can be bidirectional: a stream endpoint can be both a source and a sink for different flows. Computational constructs are provided to receive and send frames at stream endpoints. This models a symmetric interaction paradigm in which the traditional client and server distinction does not exist, and the communication happens between peers.

Frames can be transmitted, received, monitored and controlled by synchronous expressions as discussed later.

5.3 Signals

The traditional ANSA object invocations are typical RPC style, which enforces the normal call/reply ordering. For real-time and multimedia applications, the notification of events and signals cannot normally be structured as call/reply style of messages. A more freely ordered message passing scheme is necessary.

The support of freely ordered messages is provided by ***signal interfaces*** [Otway 94] in real-time ANSA objects. A signal models a call or a response message for a client, and a request or a reply message for a server (in other words, signals are the elements necessary to construct RPC style invocations, but not restricted for the purpose). Each signal has a direction, and has a source and a sink. Multiple signals can be combined together as a signal interface.

Like frames, signals can also be transmitted, received, monitored and controlled by synchronous expressions as discussed in the next section.

5.4 Synchronous Computing

The traditional ANSA object model is asynchronous, which is necessary for programming large, federated, concurrent, and non-deterministic distributed systems. In contrast, real-time systems require predictable access to shared resources and programs, the behaviours of which are deterministic.

In real-time ANSA objects, the reactive model of synchronous systems used by such languages as ESTEREL [Boussinot 91] is adopted [Otway 94] because it produces deterministic parallel programs whose behaviours are reproducible and whose execution times are predictable.

In a synchronous system, time is divided into a series of logical instants. At each instant, a number of expressions are executed in response to the input signals received since the previous instant. These logical instants are ordered but have zero duration. A synchronous expression waits for a certain (combination of) signals and is executed in the same instant that the signals are received. Typical synchronous expressions include sending a signal, waiting for a signal, testing the presence of signals, watchdogs, parallel execution, sequential execution etc.

The synchronous expressions allow the construction of synchronous islands within an asynchronous world.

5.5 Explicit Binding

Binding is the process by which an activity in one object establishes the ability to invoke operations at an interface to some other object. A binding establishes and controls the communication sessions involving multiple objects so that their interactions are possible.

The traditional ANSA object model is supported by an implicit binding model which is designed to have good scaling characteristics, to optimise the usage of resources and to be optimized for RPC style interactions. It uses maximum multiplexing for efficient resource management and provides only a single Quality of Service.

Real-time objects (including the service provided by stream and signal interfaces) require predictable (and different) methods of resource allocation, resource scheduling and a wide variety of Quality of Services. Explicit binding operations [Otway 94b] are introduced to

- associate QoS with bindings
- cope with different styles of object interactions
- control the time of binding
- manage/control a binding.

5.6 QoS Framework

The traditional ANSA object model covers only the functional requirements for distributed processing. A QoS framework [Li 94] is introduced to cover the non-functional requirements such as security, dependability, and especially performance.

QoS statement is a generic mechanism which can express

- the performance requirements for a client
- the performance a server provides
- the performance constraint of the infrastructure between them.

QoS statement can also be used to conduct the negotiation of the required performance and monitor the provided performance.

QoS requirements are categorised: for a particular class of application area, a particular QoS domain is required. A universal QoS domain for many applications is unlikely to be practical.

As QoS requirements are categorised and there are many different QoS domains, it is unrealistic to use the same mechanism for all of the domains. On the other hand, since the ANSA project is interested in a common architecture that can apply to many applications, it is logical to work out a framework so that QoS domains can co-exist and relate to each other. A QoS framework provides a common conceptual model for the definition, organization, co-relation, management and engineering of different domains of QoS.

The real-time ANSA architecture uses a language-based approach to define QoS domains, QoS expressions, and QoS domain supported explicit binding operations. It allows the association of QoS statements to interface definitions, binding operations, trading operations and object invocations.

5.7 Real-Time Programming Model

Real-time processing concerns not only how computational activities are carried out, but also how shared resources are used (i.e. the manner in which contention for system resources is resolved taking into account timing constraints of real-time activities). The essence of a real-time programming model is to provide the basic abstractions so that stringent timing constraints of real-time activities are respected (guaranteed, ideally). Traditional real-time systems provide concepts such as priority and deadline to achieve the predictable execution of computational activities.

The original ANSA object execution model is non-deterministic in the sense that object execution is completely dependent on the system's resource management policy and the infrastructure provides no possibility of interacting with this management; object execution entirely depends on the system workload.

The real-time ANSA architecture defines a real-time programming model [Li 94b] to provide a predictable object execution model and to extend the traditional real-time computing concepts to distributed processing. It extends the traditional ANSA engineering model to include various priority-based and deadline-based object execution models. The programming model incorporates tasks and communication channels (the two most important resources in real-time distributed computing) as its basic programming components. It synthesises aspects of resource requirements, resource allocation and resource scheduling into an object-based programming paradigm.

6 Real-Time ANSAware

ANSAware is an implementation of the ANSA computational model and an example of the ANSA engineering model. Traditional ANSAware was designed to scale, to cover diversity, to support federation and to be efficient in resource usage. Real-time and multimedia processing introduces new requirements such as predictable resource access, separation of resource allocation, and making use of existing real-time technologies etc.

A real-time ANSAware is under development, the first version (named RAW 1.0) [Li 94a] is already in operation under the DEC Alpha OSF/1 and the HP HP/RT real-time environments. RAW 1.0 has achieved the following results:

- compatible with ANSAware 4.1
- running over a de-facto industry standard: real-time POSIX threads
- full p-thread real-time scheduling and threading capabilities
- selective communication multiplexing by QoS specification and explicit binding operations
- application controlled resource allocation
- supporting the real-time programming model given in [Li 94b]
- comparable performance to other distributed real-time system environments.

RAW 1.0 extends ANSAware 4.1 in the following aspects in terms of functional requirements:

- extended tasking system
  - — entry: a new abstraction which represents a scheduling point. Separate scheduling points can be used for the separation of concerns of different scheduling requirements
  - — real-time scheduling: preemptive priority-based scheduling
  - — multiple scheduling policies: the co-existing of real-time and non-real-time scheduling support
  - — real-time tasks: full real-time p-thread functions.
  - — stack-based rescheduling system: a thread may use its task (stack) resource to execute another thread to facilitate dynamic real-time scheduling
  - — real-time threads: a thread may be associated with a priority and/or deadline

- multiple thread scheduling policies based on policy/mechanism separation.

- extended communication system
  - multiple execution protocols: using separate transportation protocols for real-time and non-real-time communications
  - timed execution protocol: this is a new RPC protocol which understands priority, deadline and deadline types. It can also handle various deadline-expire exceptions
  - connection-oriented execution protocols and message passing protocols: this allows the preallocation of separate communication endpoints for different interfaces
  - selective multiplex of communication channels
- extended application programming interface
  - abstractions for accessing tasking resources
  - QoS objects, two kinds of QoS objects are introduced: one for the description of a communication end point, another for the description of in-band QoS requirement for an invocation
  - explicit binding operations: to bind an interface with a communication channel of a specific QoS
  - invocations with QoS: the attachment of in-band QoS based on invocations.
- extended ANSAware PREPC and IDL language to include QoS statements

7 Summary

This article examined the problem space and technology bases of real-time open distributed processing. An integrated system architecture was suggested and the benefits of the architecture were presented. The practical need and importance of the architecture was discussed along with the current technology trends in both distributed processing and real-time applications. It was also suggested that the architecture might target (not exclusively) *supervisory control* as its applications.

The article also presented some of important progress of the ANSA Phase 3 project to extend the ANSA object model for real-time and multimedia processing to include:

- a symmetric object interaction model (stream interface) to supplement the traditional asymmetric client/server interaction model, so that applications may access communication endpoints for multimedia streams (the provision of peer-to-peer communication)
- a symmetric object invocation model (signal interface) to supplement the traditional asymmetric RPC style invocation model, so that multimedia frames and real-time signals can be treated as normal object invocations
- a synchronous computation model to provide predictable computing
- a QoS driven explicit binding model for performance guarantee and resource management
- a generic QoS framework for addressing non-functional requirements and handling the complexity of different QoS domains
- a real-time programming model for extending the traditional real-time computing concepts to distributed processing
- a real-time ANSAware for supporting the above computational and engineering extensions.

8 Related Work

Current research at CNET [Hazard 93], Lancaster University [Coulson 92] and University of Kent are all converging on a common architecture for distributed multimedia and real-time processing relevant to our real-time ANSA architecture.

9 Acknowledgements

The authors of this article would like to acknowledge the contribution of their colleagues in the ANSA core team, particularly Andrew Herbert, Yigal Hoffner, Owen Rees, Gomer Thomas and John Warne for their valuable comments.

10 References

[Biagioni 93] BIAGIONI, E., COPPER, E. and SANSOM, R. Designing a Practical ATM LAN. IEEE Network. March 1993.

[Boussinot 91] BOUSSINOT, F. and SIMMONE R. The ESTEREL Language, Proc. of the IEEE, Vol. 79, No. 9, September 1991.

[Coulson 92] COULSON G. et al. Extensions to ANSA for Multimedia Computing, Computer Networks and ISDN Systems, 25, pp 305 - 323, 1992.

[Gopinath 93] GOPINATH P. and BIHARI T. Concepts and Examples of Object-Oriented Real-Time Systems, In Readings in Real-Time systems, Y H Lee and C M Krishna ed., 123-136, IEEE CS Press, June 1993

[Hazard 93] HAZARD L. et al. Towards the Integration of Real Time and QoS Handling in ANSA Architecture, ANSA Phase 3 Project Report CNET/RC.ARCADE.01, June 1993.

[Herbert 94] HERBERT, A.J. An ANSA Overview. IEEE Network, pp18-23, January 1994.

[Herbert 91] HERBERT A. J. The Challenge of ODP, Document APM 1033, Architecture Projects Management Ltd., Cambridge U.K., February 1993, Also Appeared as an Invited Paper for the Berlin ODP Conference, October 1991.

[Johnson 93] JOHNSON, D.B. and ZWAENEPOEL, W. The Peregine High-Performance RPC System. Software Practice and Experience, 23(2). 1993.

[Leung 90] LEUNG W. H. et. al. A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks, IEEE JSAC, 8(3):380-390, April 1990.

[Li 94] Li G. A Model of Real-time QoS, Document APM.1151, Architecture Projects Management Ltd., Cambridge U.K., March 1994.

[Li 94a] Li G. Real-time ANSAware (RAW) Version 1.0: Programming and System Overview, Document APM.1207, Architecture Projects Management Ltd., Cambridge U.K., May 1994.

[Li 94b] Li G. Some Engineering Aspects of Real-Time, Document APM.1222, Architecture Projects Management Ltd., Cambridge U.K., May 1994.

[ISO 93] ISO/IEC 10746-3, ITU-TS Recommendation X.903: Basic Reference Model of Open Distributed Processing: Prescriptive Model, (2nd CD draft) June 1993.

[Northcutt 87] NORTHCUTT, J.D. Mechanisms for Reliable Distributed Real-Time Operating Systems: The Alpha Kernel. Academic Press. 1987.

[OMG 92] OMG. Object Management Architecture Guide. OMG TC Document 92.11.1.

1992.

[Otway 94] OTWAY D. Streams and Signals, Document APM.1108, Architecture Projects Management Ltd., Cambridge U.K., May 1994.

[Otway 94a] OTWAY D. Explicit Binding, Document APM.1239, Architecture Projects Management Ltd., Cambridge U.K., June 1994.

[Rees 93] REES, O. The ANSA Computational Model. Document APM.1001,Architecture Projects Management Ltd., Cambridge U.K., February 1993.

11 Biographies

Guangxing Li

Guangxing Li obtained his Honours degree in Computer Science from the University of Electronic Science and Technology of China in 1983, ChengDu, P. R. China. He went on to gain an MSc in Computer Software from the same university in 1986, and then a PhD in Computer Science from the University of Cambridge in 1993. The PhD topic was “Supporting Distributed Real-Time Computing”. He then joined APM as a member of technical staff and has been working on the development of the real-time ANSA architecture.

Dave Otway

Dave Otway is deputy chief architect of the ANSA project. He has worked on ANSA since its inception in 1985. Before that he headed the information technology division at GEC’s Hirst Research Centre. He has 25 years experience in the computing industry and a BSc in computer science from Manchester University.

12 Epilogue

The results of the ANSA Phase III programme are available to sponsors and vetted bodies.

The ANSA project has a World Wide Web server (URL is http://www.ansa.co.uk). Documents describing the results can be obtained through an FTP server (ftp.ansa.co.uk). The FTP server is not publicly accessible. Each sponsor has its own password. The contact person within ICL is John Brenner@BRA05.

General e-mail enquires to the ANSA project can be made to apm@ansa.co.uk.

13 Figures

Figure 1: Contributory Technologies

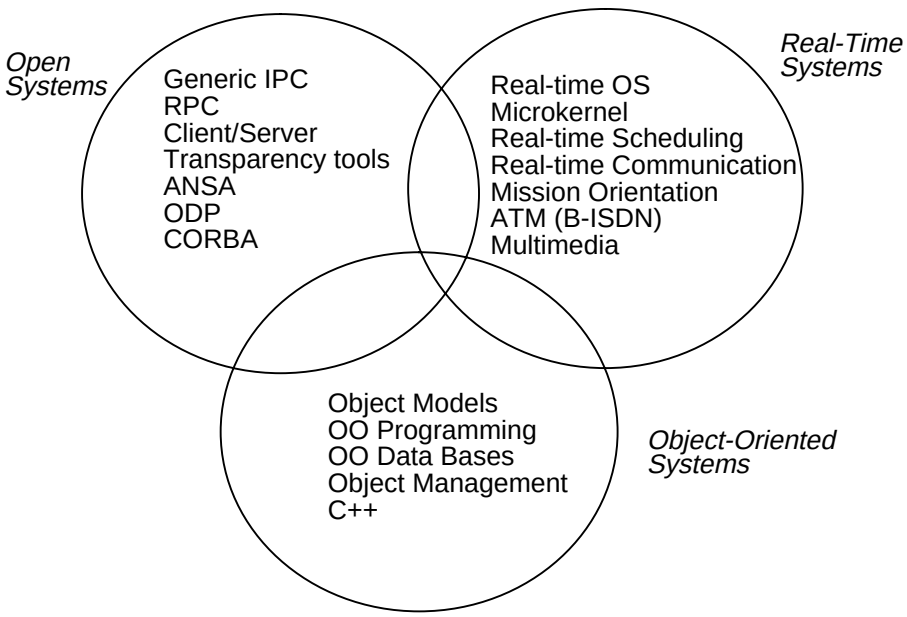

Figure 2: Real-Time ODP Functionality

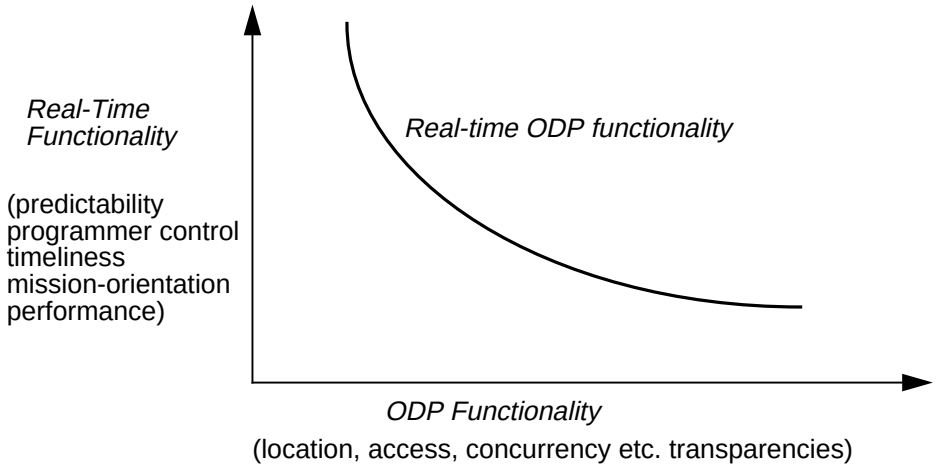

Figure 3: Supervisory Control