Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

APM Business Unit

TINA and Real-time Systems

Ian Macmillan

(

Abstract

This document presents a summary of the issues involved in supporting time critical functions (e.g. multi-media services and connection setup), in telecommunications information networks and in particular how these relate to the choice of both operating system and middleware for distribution.

The aim of the document is to provide an introduction to various ANSA technical reports which contain a considerable amount of additional detail. To this end, part of the document is organised as a commentary with suitable references.

The ANSA technical reports cover the characteristics of distributed real-time systems, the types of mechanism that must be provided by middleware to support such systems and the description of a practical implementation of such extensions in ANSAware.

.

APM.1518.01 **Approved** 19th July 1995
Architecture Report

Distribution:
Supersedes:
Superseded by:

TINA and Real-time Systems

TINA and Real-time Systems

Ian Macmillan

APM.1518.01

19th July 1995

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(01223) 515010
INTERNATIONAL	+44 1223 515010
FAX	+44 1223 359779
E-MAIL	apm@ansa.co.uk

Contents

1	**1**	**Introduction**
1	1.1	Overview
1	1.2	Audience
1	1.3	Structure of document
2	**2**	**Operating Systems**
2	2.1	Overview
2	2.2	Monolithic operating systems
4	2.3	Micro-kernel operating systems
5	2.4	Real-time oriented operating systems
7	**3**	**Middleware**
7	3.1	Introduction
7	3.2	Middleware over a monolithic operating system
8	3.3	Middleware over a microkernel
8	3.4	Middleware over a real-time operating system
9	3.5	Conclusion
10	**4**	**Real-time facilities and Middleware**
10	4.1	Introduction
10	4.2	Open Distributed Systems and Middleware
11	4.3	Development of a Performance Framework
11	4.4	The ANSA Binding Model
12	4.5	Some Engineering Aspects of Real-Time
12	4.6	ANSAware/RT
14	**5**	**Summary**

1 Introduction

1.1 Overview

Time critical functions (e.g. multi-media services and connection setup) place a number of unique requirements on the operating systems that support them; requirements which often conflict with those of the more traditional non real-time variety. Since the majority of operating systems in general use today were designed with the needs of the latter in mind, it is hardly surprising that these systems are unable to satisfy the real-time requirements of the time critical services that are beginning to emerge.

This document presents a summary of the issues involved in supporting time critical functions and discusses how well these are met by the various kinds of infrastructure:

- Traditional monolithic operating systems;
- Micro-kernel operating systems;
- Operating system plus Middleware.

1.2 Audience

This document is aimed at the technical reader wishing to assess the suitability of the various infrastructure options for supporting time critical functions in Telecommunications Information Networks, e.g. multi-media services and connection setup.

1.3 Structure of document

This document is divided into two four chapters. The first considers the various types of operating system and discusses their relative merits with regard to hosting time critical functions.

The second chapter discusses the combined characteristics of a distributed infrastructure formed by layering middleware over the types of operating system previously described. It includes an overview of the direction in which distributed infrastructure is evolving.

The third chapter provides an introduction to recent research into the type of facilities that will need to be provided by future middleware products to support time critical functions. An extended version of ANSAware is described which incorporates these ideas. The chapter is in the style of a commentary referencing various ANSA technical reports which provide considerable additional detail.

The final chapter contains a summary of the main points.

2 Operating Systems

2.1 Overview

Traditional operating systems are largely monolithic in construction and tend to offer a single application programming interface (API) as the means to access operating system functions. Typical examples include: most existing implementations of Unix, DEC's VMS and the majority of mainframe operating systems.

Modern operating systems are becoming more modular and often provide various API's, or 'personalities', as layers on top of an underlying micro-kernel. Examples of such systems include Chorus/MIX and Microsoft Windows NT.

Recent trends towards client server applications and open distributed systems in general, are resulting in an increasing number of middleware products. These facilitate the construction and deployment of distributed servics that will be used in an environment consisting of multiple heterogeneous systems. Middleware is typically layered over existing operating systems. Examples include DCE and CORBA.

Finally, communications and real-time oriented operating systems are emerging which will allow application control of system resources and hence enable applications to offer real-time guarantees.

2.2 Monolithic operating systems

Traditional, monolithic operating systems are typically structured along the lines of Figure 2.1. In these systems, any module[1] can call any other and data is often global. In general, all the operating system code runs in privileged mode with no protected boundaries. Operating system code is normally protected from application code by running the latter in a processor mode conferring less privilege.

An alternative structure is shown in Figure 2.2. In this model, a semblance of order is imposed by allowing each layer of code to access only interfaces provided by lower layers. In this way, some measure of modularity is provided although it is seldom enforced by hardware[2]. However, general experience suggests that layering frequently has a negative impact on performance.

Monolithic operating systems tend to be large, complex and inflexible. Other potential problems lie in the difficulty of assuring reliability and security due to the large volume of privileged code.

1. In some monolithic operating systems, a module may be as small as a single procedure.

2. Some layered operating systems utilise multiple levels of hardware privilege to enforce the software layering on platforms where this is available.

Figure 2.1: Monolithic operating system

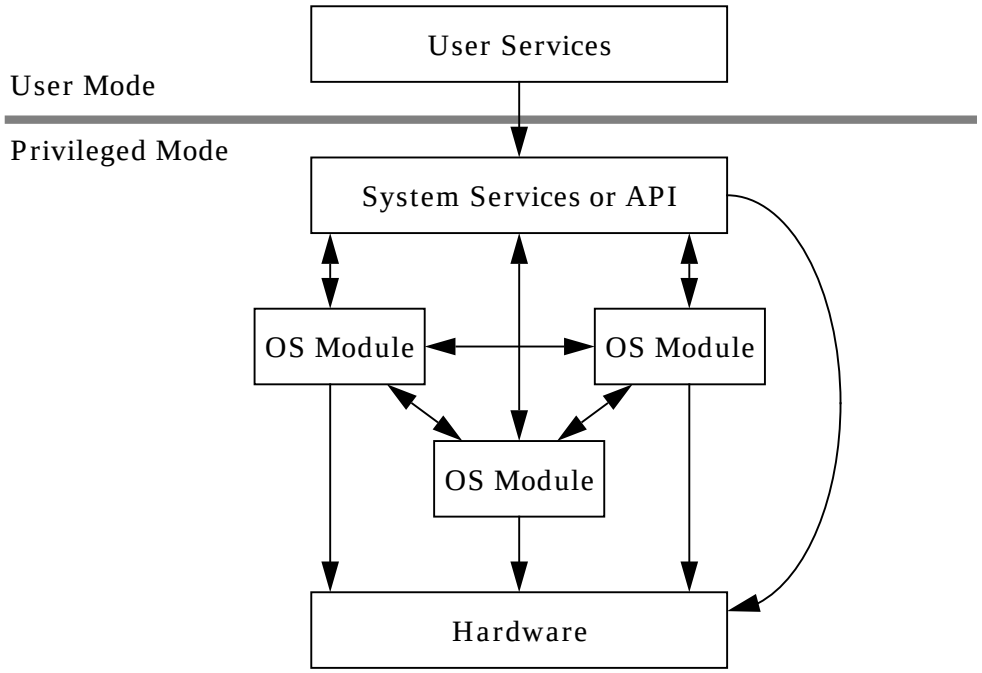
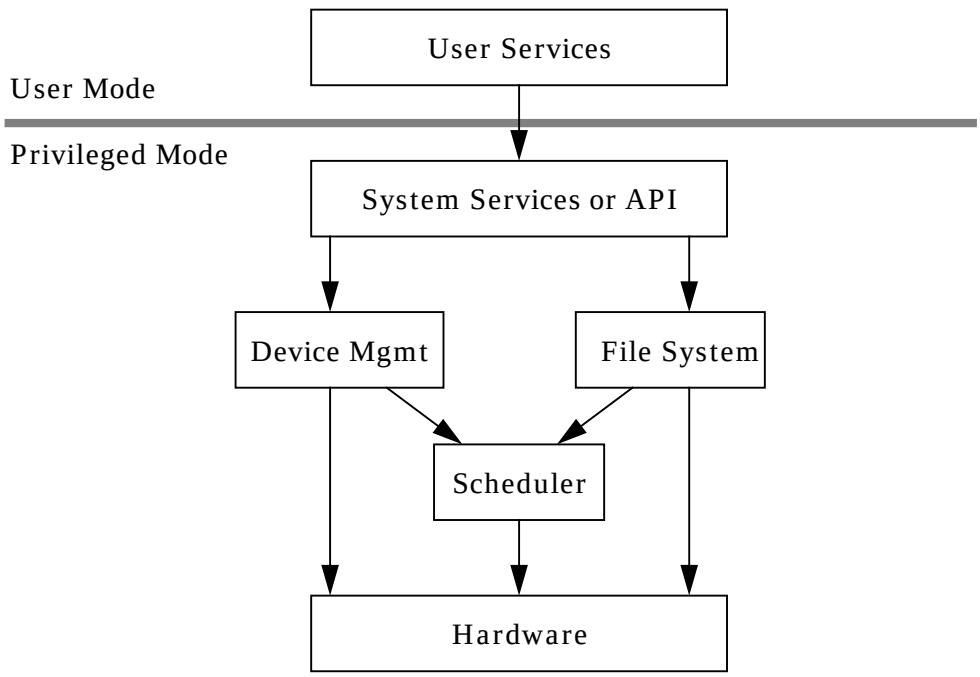


Figure 2.2: Layered Operating System



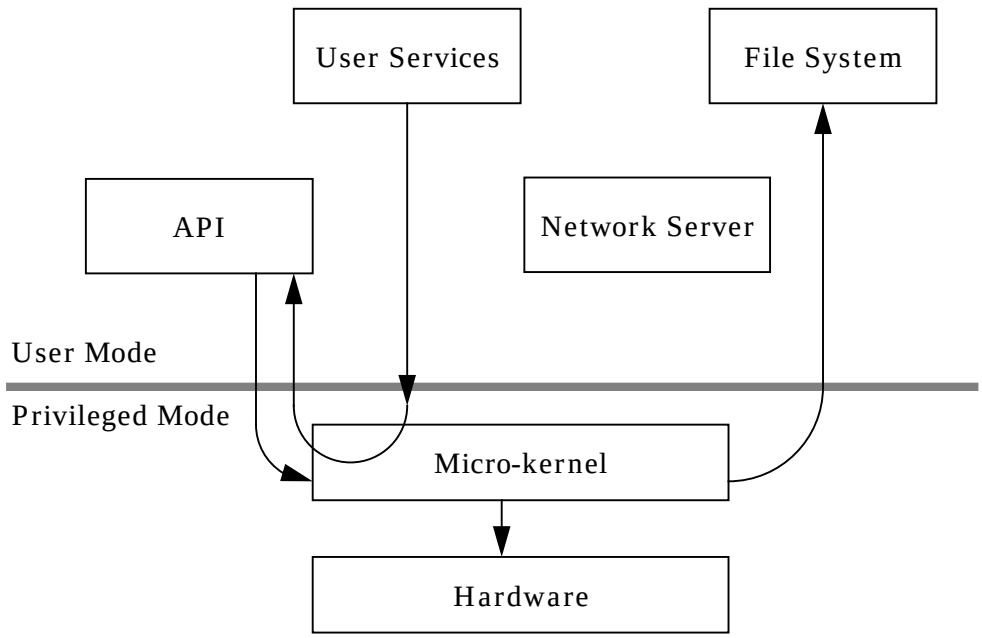
A major objective of such systems is to offer reasonable responsiveness to interactive functions whilst maintaining high overall system throughput with a broad mix of services. In order to achieve this, it is necessary for the system to control the allocation of scarce resources, e.g. CPU and memory, so that all services have an equal chance to execute. Since an executing function may be pre-empted by the system and have its resources reallocated to another, it is

unable to offer any guarantees with respect to time. As a consequence, this type of system is inherently a poor match to the needs of time critical functions.

2.3 Micro-kernel operating systems

Most modern operating systems are based on the concept of a micro-kernel in which only the minimum amount of code must run in privileged mode. The micro-kernel, or privileged portion of the system, typically provides only the most basic functions such as: message passing, scheduling, virtual memory and interrupt handling. System services, file systems and networking run in user mode as separate encapsulated processes, just like other user services. Processes communicate with one another through the use of messages, delivered by the micro-kernel. An example micro-kernel based system is illustrated in Figure 2.3.

Figure 2.3: Micro-kernel Operating System



The principal advantage of these systems is greater reliability and resilience to software faults. These advantages are brought about by hardware enforced encapsulation boundaries between services running in user mode. Failure of one service will not directly affect another and the possibility of graceful degradation after the occurrence of a fault condition is greatly enhanced.

Flexibility and scalability are also greatly improved over monolithic operating systems, in that many services may be treated as optional components. It is therefore possible, for example, to construct a system with file systems appropriate to the specific hardware configuration that the system is to run on. This has advantages for embedded systems in that the overheads of redundant components may be avoided. In addition, multiple application programming interfaces (API's) may be configured to provide multiple 'personalities', e.g. A Unix API and a DOS API to support services originally coded for different platforms.

Security can be more easily assured since the principal mechanism on which it depends (encapsulation) can be made small and hence may be adequately inspected. Provided the basic encapsulation mechanism is in place, individual services are then in a position to independently implement their own security policies.

Microkernel operating systems typically provide multi-threaded user level processes, simplifying the construction of services which must handle multiple asynchronous events.

Performance is more easily achieved since the emphasis on minimal generic low level services, enables a microkernel to be executed in an efficient and policy-free way. This allows nearly the full, device-level performance of the underlying hardware to be attained.

Cooperative scheduling between user and kernel services enables application specific scheduling through the use of process specific policies supplied by the application. This allows user level tasks to pre-empt operating system work done on behalf of other, lower priority, user tasks. Examples of this sort of facility are: Mach/CHORUS external pagers and CHORUS [Rozier 88] process linked device drivers.

All these facilities allow much greater application control of resources and improved flexibility through the ability to strip down and customize the system.

2.4 Real-time oriented operating systems

Real-time oriented operating systems are specialist systems designed to address the needs of real-time applications. Real-time requirements include such things as: predictability, timeliness and performance.

Traditional real-time systems are often highly optimised, closed systems that are not easy to modify or adapt for different applications. Often these systems are embedded and used to drive special purpose hardware.

Recent research has looked at providing generic real-time facilities using modified microkernel operating systems. For example, Real-time Mach [Tokuda 90] was developed to provide the following features:

- Real-time Thread Model. The real-time thread model supports a predictable real-time scheduler and provides a uniform interface to both real-time and non real-time threads. Apart from priority, timing attributes may be associated with a real-time thread, which include deadline, worst case execution time and periodic properties.
- Integrated Time-Driven Scheduler. The scheduler is divided into two layers: policy and mechanism. A scheduling policy is a self-contained object and can be associated with a processor or a processor set. Apart from providing traditional real-time scheduling policies, the scheduler also provides rate monotonic policies.
- Real-time Thread Synchronisation.
- Memory Resident Object Manager. This mechanism eliminates the unpredictable page fault handling delay associated with more traditional demand paged systems.

The key facilities that must be provided by any real-time operating system are predictable scheduling and suitable communications. The latter includes

aspects such as: bandwidth, jitter, multiplexing and latency. Collectively these are referred to as Quality of Service (QoS). Real-time services must be able to request suitable values for these parameters when establishing and using a communications channel. This is called QoS negotiation.

3 Middleware

3.1 Introduction

Large scale open distributed systems are generally built on top of a set of distribution mechanisms packaged into a layer known as middleware. Middleware isolates the application from platform specific differences, both hardware and software, and provides facilities to hide the undesirable aspects of distribution. These are often referred to as distribution transparency mechanisms and can be classified as follows:

- location transparency - masking the physical location of services;
- access transparency - masking differences in representation and operation invocation mechanism;
- concurrency transparency - masking overlapped execution;
- replication transparency - masking redundancy;
- failure transparency - masking recovery of services after a failure;
- resource transparency - masking changes in the representation of a service and resources used to support it;
- migration transparency - masking movement of a service from one application to another;
- federation transparency - masking administrative and technology boundaries.

Remote Procedure Call (RPC) and client-server interactions are widely accepted as middleware technical apparatus. Examples of middleware include DCE, CORBA and ANSAware.

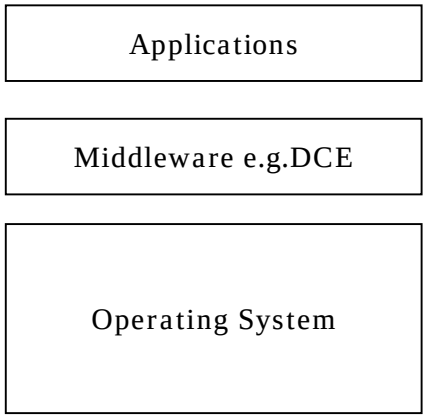
Current implementations of middleware are layered on top of existing operating systems. This chapter considers the implications of this layering for the various types of operating system already described.

3.2 Middleware over a monolithic operating system

One approach to implementing middleware is shown in Figure 3.1. Here the middleware is layered over a monolithic operating system and can therefore offer only the facilities provided by the underlying system. These facilities will seldom include real-time facilities as already mentioned in §2.2 *Monolithic operating systems*.

The implication is that an open distributed system, based on this type of operating system, is unlikely to be able to adequately support time constrained functions[1]. This suggests that current approaches to distribution, based on the structure depicted in Figure 3.1, are inappropriate for time critical functions.

Figure 3.1: Distribution using a traditional OS



It should be noted that DCE is shown only as an example of a middleware product, various alternatives could be substituted such as OMG's CORBA. One possibility, which is definitely not recommended for real-time applications, is to employ DCE as the mechanism for inter-ORB communication. Such a solution exacerbates the difficulty of providing end to end time guarantees at the application, due to the additional overhead and the need to negotiate for resources between the extra layers.

3.3 Middleware over a microkernel

Middleware may be layered on top of a microkernel in several different ways. The simplest structure, which has the advantage that it is also currently available, is illustrated in Figure 3.2. This is an improvement on the monolithic approach in that it takes advantage of the microkernel system's greater robustness. However the problems of inadequate real-time facilities and difficulty in negotiating end to end guarantees remain.

Other, more promising alternatives exist although they are not yet generally available. Research into these has already been carried out by, amongst others, OSF with respect to its OSF/1 microkernel, Mach [Black 91] and by Sun with SPRING. The options include:

- Build the middleware into the API personality layer;
- Build the middleware directly on top of the microkernel;
- Incorporate the middleware directly into the microkernel.

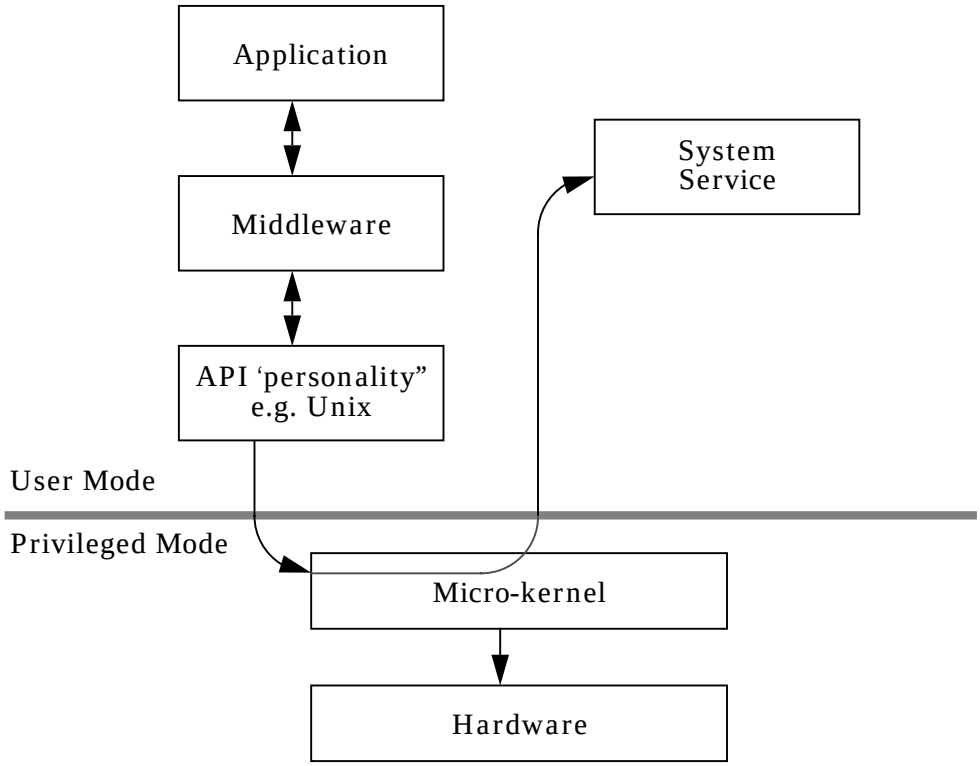
Of the possible options, a mixture of the last two would be the most natural.

3.4 Middleware over a real-time operating system

In practice, time constrained functions will require the sort of real-time facilities found in communications oriented systems. It follows that

1. In practice, real-time guarantees can only be offered under exceptional circumstances such as when the system load is completely predictable. This imposes severe constraints on the design of such a system.

Figure 3.2: Middleware over a microkernel personality



distributed time critical services will need to be built using middleware layered over a real-time operating system. However, as already mentioned, current communications oriented systems are neither sufficiently general nor extensible.

3.5 Conclusion

The logical conclusion is that distributed interactive multi-media requires:

- An operating system with real-time facilities;
- Middleware that allows access to the underlying real-time facilities[1].

Furthermore, for reliability, security and performance, the operating system should be based on a microkernel. A system in which the middleware is properly integrated as discussed in §3.3 *Middleware over a microkernel* will provide finer grain control of resources.

1. This is discussed in §4 *Real-time facilities and Middleware.*

4 Real-time facilities and Middleware

4.1 Introduction

This chapter examines the suitability of middleware for supporting time critical functions, both in terms of currently available product and that emerging from recent research. It is organised as a commentary with references to various ANSA technical reports which provide considerable additional detail.

4.2 Open Distributed Systems and Middleware

Open distributed processing is concerned with the use of commodity technology to build integrating applications or services that link together existing applications, databases, control systems and users. This technology is often referred to by the name middleware since it is usually implemented as the middle layer between applications and the operating system.

Such systems will typically consist of various heterogeneous platforms and the services themselves are likely to change rapidly in response to business needs. These factors lead to pressure for these services to be implemented above standard interfaces which can be expected to persist over technological change. Middleware provides this standard interface.

Distributed services must be designed to meet business needs and there will inevitably be trade-offs between precision of results, timeliness, scale of access and so on. In addition, some particular kinds of service, such as time critical functions, will require very specific performance targets to be met if the service is to be viable. To achieve these goals, both middleware and the underlying operating system together must offer suitable mechanisms to enable service end to end guarantees. In addition there must be support for continuous information flows or streams.

Current reference models for open distributed processing, including ISO RM-ODP [ISO/IEC 95], OSI Management and OMG Object Management Architecture (OMA) [OMG 92], say little about real-time issues. Current associated standards, such as the Open Software Foundation's Distributed Computing Environment (DCE) and the Object Management Groups Combined Object Request Broker Architecture (CORBA), do not address real-time requirements.

The following sections introduce reports describing the results of recent ANSA research into ways of identifying the requirements and identifying the types of mechanisms required to satisfy them.

4.3 Development of a Performance Framework

A general framework for performance management in distributed systems is given by [Wai 93] which provides an in-depth study of the parameters, scope, and interaction and interference between timeliness, performance and other open, distributed system control functions. This report covers:

- The role of Quality of Service (QoS) in the performance control process. Negotiations of QoS is an essential concept in a federated, heterogeneous environment;
- An overview of the fundamental concepts associated with timeliness. This includes a discussion of reactive systems and synchronous programming;
- Development of a classification of real-time systems including some hints on design principles;
- Design guidelines based upon the following observations:
  - Open distributed systems do not have specific performance characteristics
  - Interactions between real-time and non real-time systems must be allowed
  - Predictable systems are by design
  - Scheduling guarantees of predictable systems must be protected
  - Timing failures cannot be avoided
  - Resources and latent mechanisms for predictable systems must be predictable
  - There is no global clock, but there are synchronised clocks
  - Uncertainties in exclusion mechanisms must be scoped
- A study of constraints between timeliness and system functions such as fault tolerance, federation, monitoring and management. The aim is to identify areas where optimization, efficiency and meeting guarantees may or may not be extracted;
- The report concludes with a proposal for extending the ANSA computational model to cater for real-time needs. The roles of QoS contracts, streams and timed path expressions are put into context and the work is related to CORBA IDL and C++.

4.4 The ANSA Binding Model

A binding may loosely be described as the association between a computational name and a specific instance of an interface[1]. The process of establishing a binding (engineering the allocation of all the resources needed to make an invocation) is particularly relevant to time critical functions requiring particular performance guarantees. For example, a real-time service may wish to request a certain minimum bandwidth or amount of permissible jitter when communicating with another component.

The subject of binding in the ANSA architecture is covered by [Otway 95]. This report describes implicit binding as adopted in most existing implementations

1. A more rigorous definition of binding may be found in [OMG 92].

of middleware and explains why this is insufficient for multi-media streams and predictable (time critical) functions. It goes on to describe an explicit binding mechanism that:

- Provides precise control over the timing and duration of binding;
- Generates domain specific QoS specifications;
- Caters for multi-channel and multi-party bindings;
- Handles all binding management in applications.

4.5 Some Engineering Aspects of Real-Time

Explicit binding is essentially concerned with the communications aspects of real-time, things like: bandwidth, latency, jitter and so on. However, this is not the complete story; a time critical function wishing to offer performance guarantees must also be able to exercise control over the allocation of other scarce system resources, such as the CPU for example.

In the context of ANSA, [Li 95] proposes an integrated architecture for distributed real-time systems. It covers some engineering designs of middleware, as extensions of ANSA, for real-time services. The report describes how:

- real-time services may control their scheduling through the concept of a **scheduling entry**;
- the processing of requests arriving at an interface may be prioritised by being associated with a specific **scheduling entries** which in turn may be allocated a number of system processing tasks;
- a task may dynamically change its priority by performing a **rendezvous** with a specified **scheduling entry**;
- a time critical function may control the amount of communications multiplexing through QoS statements.

4.6 ANSAware/RT

The ideas described in [Li 95] have been incorporated in ANSAware 4.1, producing a new real-time variant known as ANSAware/RT. A description of the new programming and systems facilities can be found in [Li 95a]. In brief, ANSAware/RT provides:

- An extended ANSAware that runs over a standard real-time environment (POSIX real-time threads) and maintains the properties of the environment;
- Selective communication multiplexing through QoS specification and explicit binding operations.

ANSAware/RT 1.0 is currently available for DEC Alpha running OSF/1. A version for HP/RT and LynxOS also exists and a version for Windows NT is in preparation.

The structure of [Li 95a] is as follows:

- A description of the ANSAware 4.1 enhanced application programming interface. This includes the use of the new scheduling **entry**, creation and

assignment of tasks, task scheduling policies, QoS objects, binding and the entry **rendezvous** mechanism.

- The engineering of real-time tasks over POSIX threads [POSIX], including the notion of stackable tasks and the use of synchronous I/O.
- The structure of the modified communications system and the engineering support for QoS. A new real-time RPC protocol, TREX is also described.
- The report concludes with some measurements of ANSAware/RT 1.0 performance using the distributed Hartstone benchmark [Wai 93].

5 Summary

In summary, time critical functions require support from their infrastructure in order to offer end to end guarantees. This implies:

- that the system be predictable;
- resource control must be possible from the user service;
- support for continuous information flows (streams);
- explicit binding with QoS negotiation.

The case has been made for using an integrated real-time middleware and operating system environment to host time critical functions. It is suggested that a microkernel operating system is best suited and that the middleware be as closely integrated as possible. A standard open middleware solution is recommended such as OMG's CORBA.

Given that current commercially available products have not yet reached the required stage of development, it is recommended that pressure be applied to encourage vendors to produce:

- CORBA platforms that incorporate resource control and QoS negotiation;
- CORBA implementations that are better integrated with microkernel operating systems;
- Networks that allow for QoS negotiation.

Research and advanced development of combined operating system and CORBA technology is already under way (e.g. the ReTINA project) and therefore suitable products can be expected to exist in the TINA deployment timeframe.

Furthermore, ANSAware/RT demonstrates that with existing microkernel and real time operating systems, prototyping is possible.

References

[Black 91]

D L Black et al., *Microkernel Operating System Architecture and Mach;* **Journal of Information Processing**, 1991.

[ISO/IEC 95]

ISO/IEC 10746-3, *ITU-TS Recommendation X.903: Reference Model of Open Distributed Processing: Architecture*, January 1995.

[Li 95]

Guangxing Li, *Some Engineering Aspects of Real-Time*; **APM.1222.02**, APM Ltd., Cambridge U.K., April 1995.

[Li 95a]

Guangxing Li, *ANSAware/RT 1.0: Programming and Systems Overview*; **APM.1460.01**, APM Ltd., Cambridge U.K., April 1995.

[OMG 92]

OMG , *Object Management Architecture Guide*; **OMG TC Document 92.11.1**, 1992.

[Otway 95]

D Otway, *The ANSA Binding Model*; **APM.1392.01**, APM Ltd., Cambridge U.K., January 1995.

[OSF 91]

Open Software Foundation , *Introduction to OSF DCE*, December 1991.

[POSIX]

POSIX, *IEEE POSIX Std 10003.4a (Draft 13)*, September 1992.

[Rozier 88]

M Rozier et al., *CHORUS Distributed Operating Systems;* **Computing Systems Journal**, December 1988.

[Tokuda 90]

H Tokuda, T Nakajima, P Rao, *Real-Time Mach: Towards a predictable real-time system*; **USENIX 1990 Mach Workshop**, 1990.

[Wai 93]

F Wai, D Otway, N Howarth and A Herbert, *A Performance Framework*; **APM.1137.01**, APM Ltd., Cambridge U.K., March 1994.

[Weiderman 89]

N Weiderman, *Hartsone: Synthetic Benchmark Requirements for Hard Real-Time Applications*; **Technical Report CMU/SEI-89-TR-23,** Software Engineering Institue, Carnegie Mellon Univeristy, June 1989.