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

A Model of Real-Time QoS

Guangxing Li

Abstract

This document discusses ANSA real-time QoS extensions.It explores the requirements for QoS specification and design constrains for QoS expressions. A QoS model is defined and an example of the application of the QoS model in the design of real-time applications is provided.

APM.1151.00.04 **Draft** 16 March 1994

Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

Copyright © 1994 Architecture Projects Management Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.

A Model of Real-Time QoS

A Model of Real-Time QoS

Guangxing Li

APM.1151.00.04

16 March 1994

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

Copyright © 1994 Architecture Projects Management Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.

Architecture Projects Management Limited takes no responsibility for the consequences of errors or omissions in this Report, nor for any damages resulting from the application of the ideas expressed herein.

Contents

1	**1**	**Introduction**
1	1.1	Objective
1	1.2	Benefits
1	1.3	Audience and context
2	1.4	Outline of the document
3	**2**	**Quality of Service**
3	2.1	QoS characteristics
3	2.2	Towards a QoS framework
4	2.2.1	QoS domain
4	2.2.2	QoS management
5	2.2.3	QoS basic mechanisms
5	2.2.4	QoS common expression
5	2.2.5	QoS transparency mechanisms
5	2.3	QoS expression
5	2.3.1	Requirements
6	2.3.2	Language level QoS expression
6	2.4	QoS and computational/engineering entities
7	2.5	Designing a QoS language
7	2.5.1	Expressing QoS
7	2.5.2	QoS items
7	2.5.3	Combination
7	2.5.4	Conformance check
8	2.5.5	Construction
8	2.5.6	Management/control interfaces
8	2.5.7	Domain
9	2.6	A simple QoS language
11	2.7	QoS and interface
12	2.8	QoS matching
13	2.9	Viewpoints
14	2.10	Summary
15	**3**	**A model of real-time programming**
15	3.1	Towards a real-time programming model
15	3.2	A model of real-time tasking
15	3.2.1	Real-time objects
17	3.2.2	Real-time object invocation
17	3.2.3	Scheduling
19	3.3	A model of real-time communication
19	3.3.1	Towards a parallel protocol stack
19	3.3.2	Towards a timed RPC protocol
19	3.3.3	Towards a decomposable protocol stack
20	3.4	Summary

21 4 A model of real-time QoS

24 5 Related work
24 5.1 IMAC
24 5.2 Lancaster work
24 5.3 OSI QoS framework
25 5.4 CNET work

26 6 Summary
26 6.1 Acknowledgment

1 Introduction

1.1 Objective

It is the stringent timeliness and performance nature of real-time applications that are the primary source of problems. These applications introduce new problems of behaviour and resource requirement specification and management in additional to the basic ANSA/ODP distribution problems.

It is commonly accepted that behaviour and resource requirement constraints can be addressed by Quality of Service (QoS) statements, while the implementation of QoS constraints can be addressed by binding facilities in ODP context [ISO/ODP 93].

Both QoS specification and binding model in current ANSA/ODP architecture are in early-development stage. This document provides a refined model for extending the current ANSA/ODP work in QoS aspect.

The QoS model provides a structured concepts, entities and their structures which enable the design and implementation of ANSA/ODP compliant distributed real-time programming systems.

1.2 Benefits

The benefits of this work are two-fold:

- explains how the ANSA (binding and QoS) concepts are applied to real-time systems
- explains how real-time technologies are integrated with ANSA systems.

1.3 Audience and context

The audience of this document are expected to be the designers of distributed real-time environments. The audience are assumed to be familiar with the ANSA Computational Model [TR.01 93] and the Engineering Model [RC.282 91].

This document should be read in conjunction with “Engineering Aspects for Real-Time” [RC.1072 93] and “Towards an ANSA Binding Model (slides)” [RC.1150 94]. The documents are related to each other in the following fashion:

- RC.1072 explains and develops a model of real-time programming and engineering solutions for some general real-time problems.
- this document develops a set of requirements for QoS specification and explains a QoS model that would accommodate the real-time programming model developed in RC.1072.

- RC.1150 presents an explicit binding framework and discusses the relations between binding and QoS.

1.4 Outline of the document

Chapter 2 explains the requirements for QoS specification and presents a QoS model.

Chapter 3 summarises a real-time programming model.

Chapter 4 shows how the real-time programming model fits into the more general QoS framework.

Chapter5 discusses related research.

Chapter 6 gives the conclusions.

2 Quality of Service

2.1 QoS characteristics

QoS[1] is a generic mechanism which can express performance requirements for a user, the performance a server provides and the performance constraint of the infrastructure between them. It can also be used to conduct the negotiation of the required performance and the provided performance.

The main purposes of QoS are to express:

- the provided performance,
- the required performance,
- the required resources for the provision of a service,
- the required resources for the access of a service.

In this work, we are interested in QoS with the following characteristics.

- 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 not a goal of this research.
- QoS requirements are behaviour and/or resource constraints: QoS are used to model the behaviour or resource requirement of a system. This can be either declarative or imperative.
- QoS requirements are quantifiable: QoS can be itemized to individual quantified parameters.
- QoS requirements are combinative: QoS items can be combined in various ways for the purpose of specification.
- QoS requirements are directorial: QoS are hints in the selection of mechanisms and policies to meet the various requirements and the interaction between them.

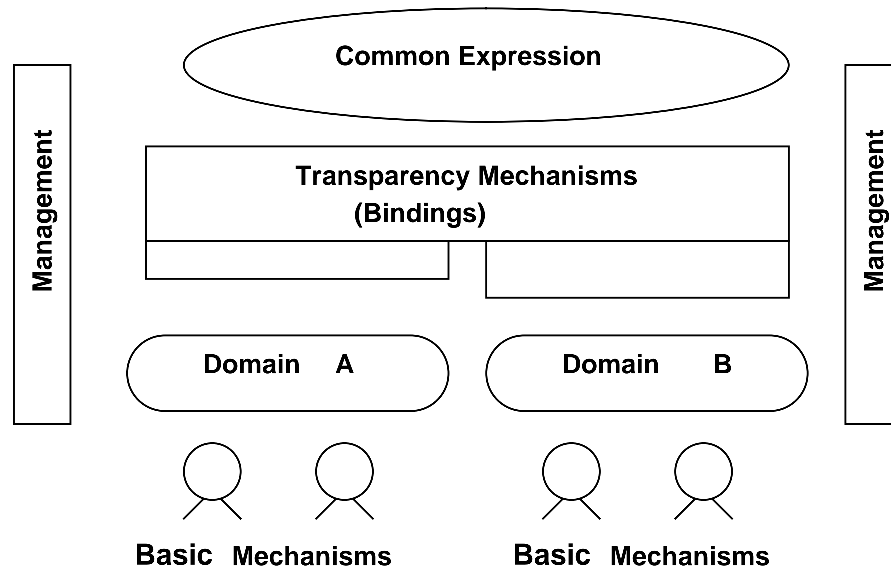
2.2 Towards a QoS framework

As QoS 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 we are interested in a common architecture that can afford many QoS domains, 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.

1. This document uses the term QoS in a narrow sense, i.e. QoS means performance QoS. Generally, QoS can mean any non-functional requirements, such as security, dependability, performance etc. The discussion of a QoS framework for the general QoS is beyond this work.

We propose a framework consisting of five major entities (shown in Figure 3.1): QoS domain, QoS mechanisms, QoS management, QoS expression and QoS transparency.

Figure 2.1: A QoS Framework



2.2.1 QoS domain

A QoS domain is associated with a specific area of application or service. There can be many QoS domains, such as, transportation QoS, real-time QoS, multimedia QoS and OSI QoS.

A QoS domain determines a specific set of QoS parameters or attributes and their combinations. A QoS domain can be constructed out of other QoS domains. For example, a multi-media QoS domain may consist of a transportation QoS domain and a real-time QoS domain.

2.2.2 QoS management

For each QoS domain, there are generic and/or specific QoS management tasks. The QoS management functions include:

- provision,
- initiation,
- change,
- enquiry,
- termination,
- monitoring,
- notification,
- negotiation.

2.2.3 QoS basic mechanisms

For each QoS domain, there exists a set of basic mechanisms for the provision and management of the QoS items in the domain. This is the engineering support for the operation of QoS. For different QoS domains, this layer of interfaces can be substantially different. There is no need for a common interface for all QoS domains. For example, in a normal real-time operating environment, the corresponding basic mechanisms may be a set of POSIX real-time programming interfaces; while in an ATM environment, the corresponding basic mechanisms may be a set of transportation interfaces.

2.2.4 QoS common expression

QoS are required to be associated with computational entities in the ODP architecture. As these computational entities are intended to be expressed in a generic language form, it implies the requirement for a common QoS expression.

Note: Editorial: a common QoS expression does not mean a common syntax, as demonstrated by DPL, it rather mean a common semantics, leaving the transparency tools to handle the difference in syntax, infrastructure etc.

There are two possible approaches for the specification of QoS:

- Application Programming Interface (API) approach --- for each QoS domain there is a set of QoS programming interfaces, this is the common approach in operating system and computer network practice.
- language-based approach --- a common QoS language is used to describe QoS items and their combination.

It is believed the API approach is useful but not adequate, and a language-based approach is preferred because of its adaptability and portability as discussed in Section 2.3.

2.2.5 QoS transparency mechanisms

Given the semantic gaps between the common QoS expression and the various QoS engineering mechanisms, a QoS transparency layer is required to provide the necessary adaptation of the language level QoS to the basic QoS mechanisms.

The QoS transparency mechanism is another set of ODP engineering transparency tools, addressing non-functional requirements.

2.3 QoS expression

2.3.1 Requirements

The common QoS expression needs to meet the following requirements:

- hiding engineering details which are irrelevant to application programmers
- hiding arbitrary syntactic differences between technologies
- keeping concepts as independent and orthogonal as possible
- imposing the simplest possible conformance rules
- providing the maximum negotiation flexibility

- providing the maximum configuration flexibility

2.3.2 Language level QoS expression

Using a language to express QoS constraints can provide a required level of abstraction by the definition of

- a standard set of forms for QoS primitives,
- a standard set of combinators for QoS expression combination,
- a standard set of rules for QoS conformance test,
- a standard set of rules for QoS negotiation,
- a standard set of QoS domains for common applications,
- a standard set of rules for the construction of new QoS domains out of others.

Language level QoS expressions allow the application of automation tools to

- compile application programs onto any suitable API,
- optimize application programs for any particular configuration,
- transform declarative requirements into imperative statements,
- check that the application program will execute correctly,
- apply the conformance rules,
- apply the negotiation rules.

Based on these observations, it seems a language based approach can meet the requirements given in the foregoing discussion, while it is difficulty for an API based approach to achieve the same goals. It is therefore believed that a language based approach to QoS is both important and necessary in ODP QoS specification.

2.4 QoS and computational/engineering entities

In relation with computational components, QoS[1] can be associated with:

- an interface type,
- an interface (an instance of an interface type),
- an object,
- an invocation or an activity.

In relation with engineering components, QoS can be associated with:

- a binding.

Of these, the QoS associated with bindings are the most complicated. Bindings can be organized as local end-system bindings, end-to-end bindings and group bindings etc.; QoS associated with these bindings are therefore representing local end-system QoS, end-to-end QoS and group QoS etc. The detailed discussion of binding and QoS is given in another related document [RC.1150 94].

1. QoS is associated with computational entities does not mean QoS is part of the computational view of a system. QoS relates to enterprise or information requirements, and is realised through engineering mechanisms.

QoS associated with other computational entities (i.e. non bindings) can be seen as QoS templates, they are not activated (in effect) until the relevant bindings are set up at run time.

2.5 Designing a QoS language

2.5.1 Expressing QoS

Syntactic and/or semantic rules may be needed to:

- identify relevant QoS domain,
- identify relevant attributes in a QoS domain,
- express quantitative measures for the identified attributes,
- describe the combination of individual attributes,
- allow conformance check,
- allow negotiation,
- allow the construction of new QoS domains out of existing domains,
- describe domain dependent management/control interfaces.

2.5.2 QoS items

The individual QoS items (or characteristics) depend on the individual platform or domain. For example, on a normal commercial real-time platform, a QoS item can be a priority, a deadline or the allocated tasks etc. On a communication system, QoS items can be the required transportation QoS e.g. jitter, error-rate etc.

A QoS item can be expressed simply as a <name, value> pair, such as <Priority, 10>, or more manfully as a <name, relation, value> triple, such as <Priority, =, 10> and <Jitter, <=, 100>.

2.5.3 Combination

It requires at least three forms of combinators to build QoS expressions:

- logical combinators: *and, or, not* etc. For example: “<priority, 10> *and* <deadline, 100>”
- ordering combinators: expressing the ordering of perceived satisfaction. For example, *prefer* QoS_1 *to* QoS_2, “*prefer* <protocol, TCP> *to* <protocol, IPC>”
- range combinators: expressing a QoS negotiation. For example, “<Rate, >=, 10> *and* <Delay, <=. 15>”.

2.5.4 Conformance check

Conformance check is a sort of *QoS match* procedure. Conformance check can be both *syntactic* or *semantic*. In the first case, it can be realised by a syntax analyser, by a type checker or by some match criteria. In the second case, it can be realised as parts of QoS negotiation procedure, involving some interactions between a QoS manager and some resource managers.

2.5.5 Construction

This is the syntactic component for the construction of new QoS value domains out of existing ones. It may be equivalent to some type constructors. For example, a *record*, a *union*, or a *set* constructor.

2.5.6 Management/control interfaces

A QoS domain may need to provide a *Bind* operation for the creation of bindings in the domain.

A QoS domain may have a few management interfaces which provide operations for QoS related operations. For example, a QoS domain at least has a binding interface which is the result of a *Bind* operation. A binding interface has at least an *Unbind* operation. Other operations of a management interface may be *ChangeQoS*, *Rebind*, *AddNewMember* etc. Other management interfaces include notification interface, monitoring interface etc.

2.5.7 Domain

A QoS domain contains all or some of the components defined in section 2.5.2 to section 2.5.6.

Figure 2.2: Trader Constraint Language

```
<constraint> := <empty>
              | <expr>
              | <expr> -> <superlative>
              | -> <superlative>

<expr> :=       <expr> or <expr>
              | <expr> and <expr>
              | not <expr>
              | ( <expr> )
              | <nexpr> in <nexpr>
              | <nexpr> == <nexpr>
              | <nexpr> != <nexpr>
              | <nexpr> < <nexpr>
              | <nexpr> <= <nexpr>
              | <nexpr> > <nexpr>
              | <nexpr> >= <nexpr>

<superlative> := min [ <nexpr> ]
              | max [ <nexpr> ]

<nexpr> :=      <term>
              | <nexpr> + <term>
              | <nexpr> - <term>

<term> :=       <factor>
              | <term> * <factor>
              | <term> / <factor>

<factor> :=     <identifier>
              | <constant>
              | ( <nexpr> )
              | - <factor>

<empty> :=
```

2.6 A simple QoS language

ANSAware system has already got one simple Trader constraint language for specifying QoS offers and constraints, and for matching QoS constraints to offer. The language was used in IMAC [TR.028 93] for simple multimedia QoS representation and negotiation. This language is therefore chosen to start with.

The short BNF for the constraint language is given in Figure 2.2. The constraint language has the following items to build QoS constraint expressions:

- superlative functions: min, max
- comparative functions: ==, !=, >, >=, <, <=, in
- constructors: and, or, not, -> (is restricted by)
- property names; any identifies
- numeric and string constants
- mathematical operators: +, -, *, /
- grouping operators: (,), [,]

In comparison with the requirements shown in the last section, the main deficiencies of the language are its inability to identify a QoS domain, to specify attributes within a QoS domain, the construction of new QoS domains and the specification of management/control interfaces. This is not a problem for the Trader in which the constraint expressions are used in a purely symbolic or syntactic sense. The extension for the definition of a QoS domain is necessary because our QoS expressions are required to be associated with bindings, i.e. each QoS expression has to be tied to a binding domain, or a binder. A QoS domain constrains the identifiers used in a QoS expression to certain keywords. Figure 2.3 shows our QoS domain definition language, which is an extended IDL language.

A QoS domain defines:

- attributes or KEYWORD and their types,
- the Bind operation,
- the management interfaces,
- QoS constraint macros.

The QoS constraint expressions (the Trader constraint language) are also extended accordingly to allow the specification of *structured keyword* by using a dotted notation, such as the REX.protocol etc.

The QoS constraint language and the domain definition language constitute our QoS language.

To make these ideas more concrete, we will present some examples shown in Figure 2.4. and Figure 2.5.

Figure 2.4 defines two QoS domains: IPC and CHANNEL. The IPC QoS domain defines four keywords: EndType, Protocol, Rate and Address. They can be used as identifiers for constituting QoS expressions in the domain. An IPC QoS expression is show in Figure 2.5. IPC has one management interface, i.e. the binding interface, which provides two control operations: Unbind and Rebind. The IPC Bind operation takes any interface reference as argument and creates an IPC binding (interface) as result. IPC models a typical inter-

Figure 2.3: QoS domain specification language

```
<spec> :=           <header> <needs> <body>

<header> :=         <identifier> : QoSDOMAIN =

<needs> :=          <empty>
                    | <needs> <need>

<need> :=           NEEDS <identifier> ;

<body> :=           BEGIN <declarations> END.

<declarations> :=   <empty>
                    | <declarations> <declaration>

<declaration> :=    <identifier> : TYPE = <type>;
                    | <identifier> : KEYWORD = <type>;
                    | <macro_header> : MACRO "<constraint>";
                    | <identifier> : <proc>;
                    | <identifier> : <interface>;

<proc> :=           OPERATION [ <arguments> ]
                    RETURNS [ <results> ]

<interface> :=      INTERFACE <if_body>

<if_body> :=        BEGIN <if_specs> END.

<if_specs> :=       <empty>
                    | <if_specs> <if_spec>

<if_spec> :=        <identifier> : TYPE = <type>;
                    | <identifier> : <proc>;
```

process communication scenario where communication has two parties: one socket and one plug. IPC is a local end QoS domain. The Bind operation sets up a socket and a plug before communication can happen.

The CHANNEL domain defines six keywords. Of these, Socket and Plug are of the type IPC.K, i.e. they are structured keywords. It is conventional to name the type of the record of the keywords in a QoS domain D as D.K; thus IPC.K is the record type of the keywords in IPC. This models an end-end domain: CHANNEL is built on top of IPC, and has a Socket and a Plug as two ends. CHANNEL defines two management interfaces: one as the binding interface, the other for event notification. The Bind operation takes one type value: a server interface type, and three interface references as arguments: a server binding management interface, a client binding management interface and a notification interface. The operation returns two result arguments: one is the binding control interface, and the other server interface. A server/client binding management interface creates local bindings (by using of the IPC domain, for example) at the server/client side for supporting a required end-to-end CHANNEL QoS. The notification interface can be used to *call-back* the binding creator when there are relevant events to be reported.

Figure 2.5 defines a MEDIA QoS domain, which is built on top of the CHANNEL. It also illustrates examples of QoS macros and QoS expressions for each of the domains.

Figure 2.4: QoS domain and constraint examples

```
IPC: QoSDOMAIN =
BEGIN
   EndType: KEYWORD = {socket, plug};
   Protocol: KEYWORD = {TCP, UDP, MSNL};
   Rate: KEYWORD = INTEGER;
   Address: KEYWORD = STRING;

   IPC_binding: INTERFACE =
   BEGIN
     Result: TYPE = {ok, fail, retry}
     Unbind: OPERATION [] RETURNS [Result];
     Rebind: OPERATION [] RETURNS [Result];
   END;
  Bind: OPERATION [IfRef: InterfaceRef] RETURNS [IPC_binding];
END.
CHANNEL: QoSDOMAIN =
NEEDS IPC;
BEGIN
    Type: KEYWORD = {shared, non_share};
    Rate: KEYWORD = INTEGER;
    Address: KEYWORD = STRING;
    Protocol: KEYWORD = {one_way, two_way};
    Socket: KEYWORD = IPC.K;
    Plug: KEYWORD = IPC.K;
    CH_binding: INTERFACE =
    BEGIN
      Result: TYPE = {ok, fail, retry};
      Unbind: OPERATION [] RETURNS [Result];
      Rebind: OPERATION [] RETURNS [Result];
     END.
    Notifier: INTERFACE =
    BEGIN
      Event: OPERATION [event: INTEGER] RETURNS [];
    END.

    Bind: OPERATION [  Type: TYPE;
                       SvrMgtIf: InterfaceRef;
                       CltMgtIf: InterfaceRef;
                       NtfIf: NotifierRef ]
          RETURNS [ CH_binding, InterfaceRef];
END.
```

2.7 QoS and interface

QoS constraints can be associated with interfaces as QoS templates for simplifying QoS specification, interface trading and conformance test.

QoS constraints can be associated with an interface by extending interface type definitions, i.e. IDL, with QoS clauses. QoS clauses can be associated with individual operations or the whole interface at interface specification time. Figure 2.6 shows such an example.

QoS constraints specified at an interface type may be automatically inherited by an interface instance when it is created at binding time, where additional QoS constraints can also be imposed.

The QoS templates and the binding time QoS clauses can be organized into an interface reference, which can be used for interface trading and QoS conformance test.

Figure 2.5: QoS domain and constraint examples: cont.

```
MEDIA : QoSDOMAIN =
NEEDS CHANNEL;
BEGIN
 Type: KEYWORD = {Audio, Video};
 Rate: KEYWORD = INTEGER;
 Delay: KEYWOE= INTEGER;
 Enconding: KEYWORD = {pal, mpeg, jpeg};
 Channel: KEYWORD = CHANNEL.K;

 M_binding: INTERFACE =
 BEGIN
  Result: TYPE = {ok, fail, retry};
  Unbind: OPERATION [] RETURNS [Result];
  Rebind: OPERATION [] RETURNS [Result];
 END.
 audio(x,y):MACRO "(Rate>=x and Delay<=y)";
 video(x,y,z):MACRO "(Rate>=x and Delay<=y and Encodeing==z)";
 Bind: OPERATION [ Type: String,
                   EndMgtIf: InterfaceRef;
                   SourceMgtIf: InterfaceRef]
       RETURNS [M_Binding, InterfaceRef];
 END.
 --QoS expression on IPC
 (Type == Socket) and (Protocol == TCP) and (Rate == 100)

 --QoS expression on CHANNEL
 (Type == non_share) and (Protocol == two_way) and
 (Socket.Protocol == UDP) and (Plug.Rate == 1000)

 --QoS expressions on MEDIA
 (type == Video) and video(100, 10, pal)

 (type == Audio) and audio(1000, 6)
```

Figure 2.6: IDL QoS Specification

```
VideoDev: INTERFACE =
NEEDS Media;
IF_QOS (Delay <= 100) and (Encoding == pal)
BEGIN
StreamIn: OPERATION [] (Rate <= 1000) RETURNS [];
StreamOut: OPERATION [] (Rate >= 100) RETURNS [];
END.
```

2.8 QoS matching

The constraint language provides well defined rules for matching a required QoS into offered QoS. For example, if the following QoS offers are made:

- (Protocol == TCP) and (Rate == 100)
- (Protocol == TCP) and (Rate == 1000)
- (protocol == UDP)

The constraint expression (Protocol == TCP) would return:

- (Protocol == TCP) and (Rate == 100)
- (Protocol == TCP) and (Rate == 1000)

Alternatively (Protocol == TCP) and (->max[Rate]) would return:

- (Protocol == TCP) and (Rate == 1000)

These rules can be used by a Trader for interface trading as illustrated by ANSAware.

2.9 Viewpoints

Generally, the binding model [RC.1150 94] is a way of structuring QoS into a manageable framework (an engineering model for QoS). There are four major components in the QoS/binding framework:

- binders: they provide binding operations to generate various QoS domain specific bindings.
- a generic QoS language: it provides a common tool to specify various QoS expressions addressing non-functional requirements in the enterprise or information viewpoints.
- QoS domain specific resource managers: for each QoS domain, there is a set of engineering mechanisms for the management and operation of bindings.
- a generic QoS language mapper/manger: it links the common QoS expressions to domain specific binders and resource managers. This entity may also carry out some common QoS operations such as conformance check etc.

Figure 2.7: Viewpoints

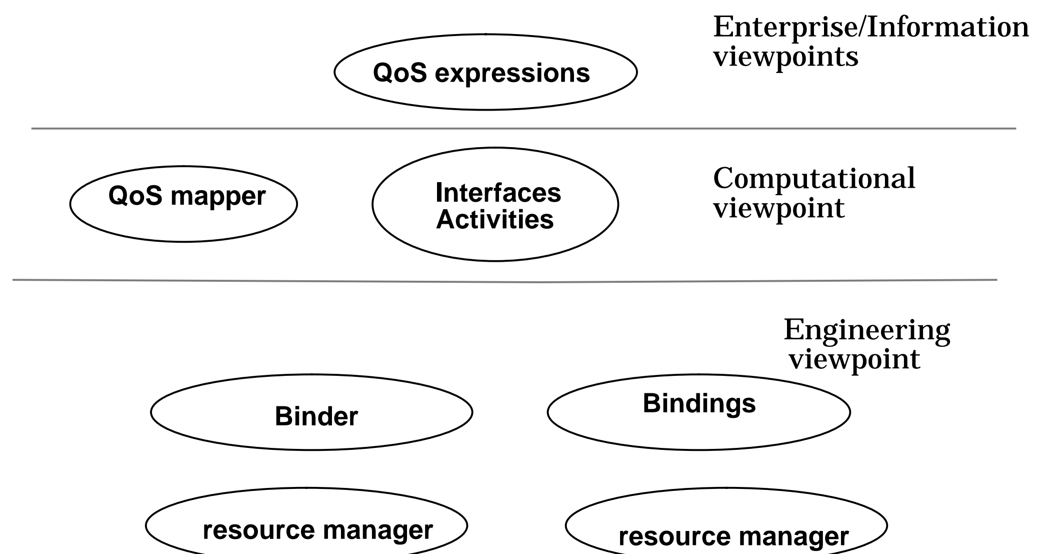


Figure 2.7 gives a graphical illustration of the framework in terms of enterprise, information, computational and engineering views.

2.10 Summary

This chapter presents a generic performance QoS model. A language based approach for QoS specification is advocated because it can provide the necessary level of abstraction. It discusses some basic design requirements for a QoS language, and shows various features of QoS expressions by using of a simple QoS language, which is an extension of the ANSA Trader constraint language and IDL language.

3 A model of real-time programming

This chapter synthesis the earlier work on engineering aspects of a real-time architecture [RC.1072 93] and experiment [Li 93].

This chapter does not attempt to explain in detail the full methodology and reasoning behind [RC.1072 93], but to provide a brief summary of these components necessary to drive a real-time QoS model discussed in the next Chapter.

3.1 Towards a real-time programming model

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 at best).

The real-time programming model developed in [RC.1072 93]] is based on the ANSA computation and engineering models. As in the ANSA system, objects provide the basis for distribution, interfaces of objects provide service access points, and named operations of an interface provide the actual services. Abstractions, mechanisms and policies are developed to allow a programmer to access and control the resource allocation of the supporting environment. Tasks (representing processor resources) and communication channels (representing communication resources) are considered the most important system resources. Both static resource allocation --- the allocation of system resources to interfaces --- and dynamic resource allocation --- the allocation of system resources to invocations are supported. *Predictability, programmer control* and *mission criticality* are the main concerns of the real-time programming model.

The real-time programming model has two parts: a tasking model and a communication model.

3.2 A model of real-time tasking

This section discusses the real-time tasking extensions of ANSA objects.

3.2.1 Real-time objects

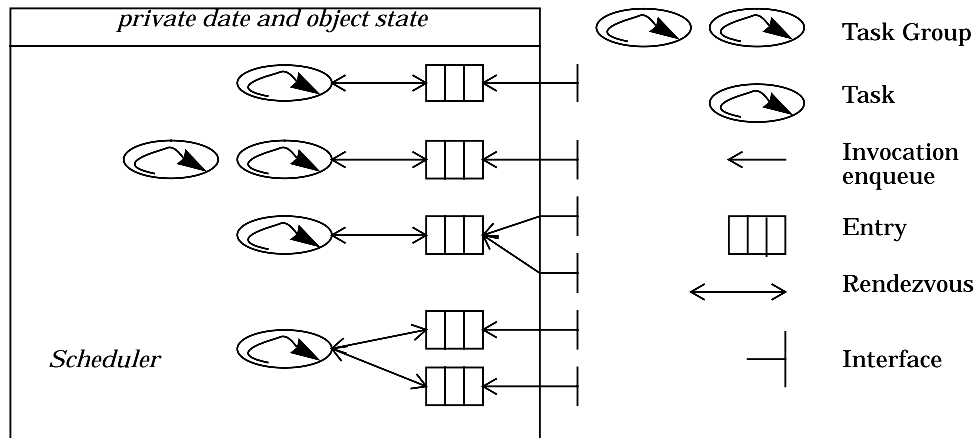
A real-time object model can be obtained by extending the ANSA object execution model with explicit resource allocation and real-time scheduling support.

A real-time object is composed of data, one or more tasks of execution, and a set of interfaces. A new abstraction, ***scheduling entry,*** or shortly ***entry***, is introduced as the basic mechanism for real-time scheduling.

An entry is a thread queue with a record of control data. An entry may be created dynamically, and interfaces of an object may be bound to it. When an

interface is bound to an entry, each operation request on the interface will be transferred (by the infrastructure) to a thread enqueued on the entry. Any thread representing a computational activity is also spawned on an entry. The entry is an engineering concept which is confined within a capsule.

Figure 3.1: Real-time object illustration



In Figure 3.1, a graphical illustration of a real-time object is given.

Flexible tasking is based on the entry abstraction. System tasks may be allocated for each individual entry. The tasks allocated are dedicated to execute the threads on the entry. When executing a thread, a task is also allowed to *rendezvous* with other entries dynamically. A ***rendezvous*** of a task with an entry means that the task waits to accept and execute one thread on the entry. Different control parameters may be selected for each entry to choose a thread enqueue policy, a task/entry rendezvous policy, and to enforce concurrency controls.

In such an object model with data, interface, entry and tasks encapsulated within a capsule, there is a choice of how many entries are allocated, which interface is attached to which entry, how many tasks are allocated to an entry, whether a task can rendezvous with a specific entry, and what kinds of resource scheduling policies are used.

The choice to allocate a new entry for some interfaces reflects the need to separate these interfaces from others for the purpose of resource management.

The number of tasks allocated to an entry not only enforces the real concurrency allowed for the execution of threads on the entry, but also affects the real-time scheduling properties, for example, *preemptivity.*

The flexibility for allowing a task to rendezvous with an entry enables an application to have complete control over its virtual processor(s) based on its knowledge of the system state.

These resource management activities can all be done dynamically, increasing the flexibility and usefulness in an open dynamic environment.

3.2.2 Real-time object invocation

The real-time ANSA allows the association of an optional *priority* (criticality) and/or *deadline* with each invocation. As the invocation crosses the physical boundary and becomes a thread in the called object, this priority and/or deadline is passed and becomes a property of the thread, which may then be used as a scheduling parameter on the server site.

The priority and/or deadline of an invocation is independent of its contents (the invocation parameters) and context (the invocation thread). Allowing explicit invocation priority (and/or deadline) has several benefits:

- it allows extra flexibility in conjunction with the server scheduler, in determining how the invocation is to be processed;
- it allows a low-priority invocation to be sent from a high-priority task without having to enhance the server (thread) task's priority;
- likewise, a low-priority thread may send a high-priority invocation to a server indicating the system has entered an urgent situation.

3.2.3 Scheduling

The main goal of the real-time tasking design is to allow the maximum control of scheduling at the application level. Care has been taken to achieve the balance between flexible and deterministic scheduling. A policy/mechanism separation approach has been taken to address the diversity of real-time programming. Real-Time programming models have been devised for specific applications. Therefore, an ideal general purpose real-time support environment should provide multiple models of real-time programming. This can be supported by the multiple application-selectable scheduling policy modules on top of a shared set of scheduling mechanisms.

The system scheduling behaviour is defined in layers as:

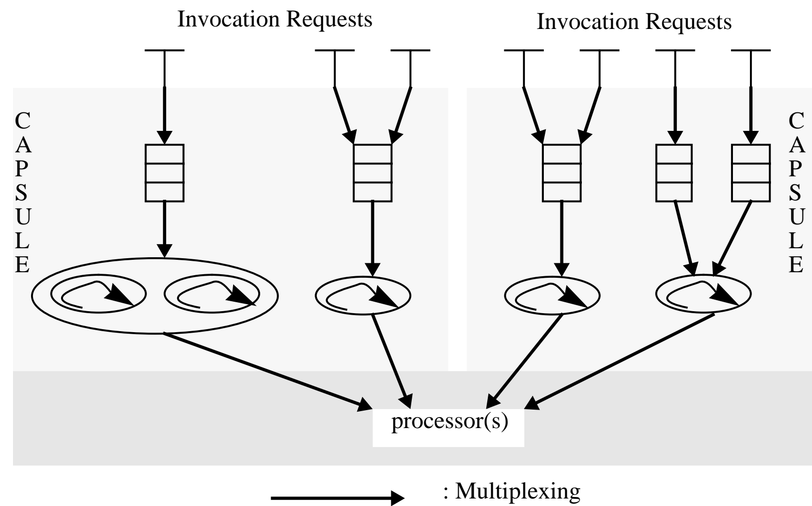
- thread scheduling --- the rendezvous scheduler on each entry.
- task scheduling --- the nucleus scheduler on tasks.

Task scheduling and thread scheduling are two separate, but related scheduling domains. Task scheduling is defined in the nucleus or the underlying operating system kernel. Thread scheduling is defined per entry. Task scheduling manages multiplexing of task executions over processor(s). Thread scheduling manages the multiplexing of requests (thread) over tasks. Figure 3.2 illustrates the structure of this multiplex.

The primary function performed by multiplexing is the sharing of processor resources, which is similar to the multiplexing in communications systems and protocols for sharing communication resources. The use of separate entries to process requests on separate interfaces offers a number of (potential) advantages:

- allows the use of a specific scheduling policy (thread scheduling policy) suitable for each interface or each interface class.
- allows the possibility of using interface specific tasks to serve requests, and thus allows for more efficient resource utilisation.
- separate entries may be processed in parallel, thus increasing performance.
- allows the possibility of end-to-end scheduling and guarantees.

Figure 3.2: Threads, Tasks and Processor(s) Multiplexing



- preserves the modularity and separation of service interfaces.

The nucleus scheduler defines how the real processor(s) is assigned to tasks, i.e. it manages the context switches between tasks. Preemption is used together with task scheduling parameters to order (either partially or completely) the otherwise non-deterministic behaviour of the task execution.

There are two issues in thread scheduling management. One is how a thread is enqueued in an entry (with the assumption that the first thread in the queue is executed first). Such a policy may be a system defined one, like invocation priority based, or an application provided one.

Some typical thread enqueue policies are:

- first come first service
- priority based
- deadline based
- priority and deadline based

Another issue is how a serving task rendezvous with a thread in an entry, i.e. how the thread scheduling parameters (priority and/or deadline) are used/ inherited by the task. This is defined by a task/thread rendezvous policy. Such a policy affects how the serving task competes for processor resources with other tasks.

Some typical task/thread rendezvous policies are:

- null --- the priority/deadline of a thread has no effect on the serving task
- priority inheritance
- transitive priority inheritance
- priority ceiling
- deadline inheritance

Detailed examinations of some typical real-time scheduling schemes, such as priority based scheduling and deadline based scheduling, can be found in [RC 1072 93].

3.3 A model of real-time communication

Real-time applications present more complicated functional requirements to the underlying communication systems. Three extensions aimed at making the ANSAware communication system more suitable for real-time applications are identified:

- a parallel protocol stack
- a timed RPC protocol
- a decomposable protocol stack

Detailed examinations of the three design can be found in [RC 1072 93].

3.3.1 Towards a parallel protocol stack

A parallel communication protocol stack allows the preallocation of communication resources (a separate channel, for example) and the removal of layered multiplexing. The main gain of this design is that it allows the application to explore the communication QoS that the low-level operating environment can provide. For example, in an ATM environment, a channel (or a virtual circuit) is allowed to associate with various transportation QoS, such as jitter, delay, priority etc. Even in an ordinary operating environment, this design allows the choose of communication protocols, such as TCP, UDP, IPC etc.

An application programmer may access the parallel protocol stack by the standard binding operations.

3.3.2 Towards a timed RPC protocol

A timed RPC protocol allows the association of deadlines with invocations. ANSA is an RPC based system. A basic goal of many RPC systems is to make the semantics of a remote call as close as possible to that of a local call. However, distribution cannot be completely ignored: applications will have to deal with the possibilities of concurrent access to shared resources, variable latency in accessing resources and communication failures. The semantics of remote calls are implemented by RPC protocols. Two often referred to semantics are *exactly-once* and *at-most-once* executions. Real-time applications add another dimension to the problem: timeliness --- arbitrary delays associated with synchronous RPC invocations cannot be tolerated.

Our solution to the timed RPC problem is the design of a dependable RPC protocol through which reasonable timing constraints (representing different trade-off between consistency and strictness) of a remote invocation can be specified clearly and enforced. This relieves the additional burden of having to monitor and manage timing constraints by application programmers during remote calls. The timed RPC protocol allows an invocation to be associated with a deadline, a timeout and a deadline type values.

3.3.3 Towards a decomposable protocol stack

A decomposable protocol stack allows the synthesis of the protocol stack to provide different levels of invocation semantics (such as exactly-one, at-most-once), so that an application programmer can customize the system to application-specific requirements of functionality and performance. This work is targeted at new transport protocols with QoS parameters in the operational interface.

3.4 Summary

The real-time programming model provides a framework to facilitate the enforcement of stringent timing constraints found in distributed real-time applications. The 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. The real-time communication system examines various approaches for updating current ANSAware RPC communication system for real-time applications.

4 A model of real-time QoS

This chapter explains how the real-time programming model defined in Chapter 3 fits the general QoS model given in Chapter 2.

The real-time programming model can be summarised by three QoS domains:

- session QoS domain: it allows the association of priority, deadline etc. with object invocations. The binding operation of this domain is done implicitly when an invocation is initiated.
- tasking QoS domain: it allows the specification of task resource requirement and management.
- communication QoS domain: it allows the specification of communication resource requirement and management.

The three domains are defined in Figure 4.1, 4.2 and 4.3 by using of the simple QoS language given in Chapter 2.

Figure 4.1: Session QoS domain

```
Session : QoSDOMAIN =
BEGIN
 DeadlineTye: KEYWORD = {rend_communication, rend_invocation};
 Deadline: KEYWORD = Timer;
 Timeout: KEYWORD = Timer;
 Priority: KEYWORD = INTEGER;
END.
```

Figure 4.1 defines the QoS attributes and their types that can be associated with a session or an invocation. These attributes are the values of a deadline type, a deadline, a timeout and a priority. The session QoS domain has no binding management interfaces or operations because it is managed by the ANSAware implicit binding operations.

Figure 4.2 defines a transportation QoS domain. The domain itself consists of two domains: the message passing (MPS) domain and the Execution (EX) domain. MPS and EX are currently two layers of communication protocols supported by ANSAware. The MPS domain defines four QoS attributes: IPC protocol, address, rate and bandwidth. They can be used by ANSAware to claim network resources. The EX domain has three attributes: RPC protocol, rate, and retries. The RPC protocol can be an exactly-once protocol, an atmost-once protocol, or an timed RPC protocol that understands deadlines. The transportation domain defines another two attributes: end type and channel type, in addition to a MPS and EX attributes. The transportation domain defines a binding management interface to allow the release of a transportation binding which is created by an explicit binding operation on any interfaces (refer the signature of the Bind operation in the domain).

Figure 4.2: Communication QoS

```
MPS: QoSDOMAIN =
BEGIN
    Protocol: KEYWORD = {TCP, UDP, IPC, MSNL};
    Address: KEYWORD = String;
    Rate: KEYWORD = INTEGER;
    Bandwidth: KEYWORD = INTEGER;
END.

EX: QoSDOMAIN =
BEGIN
    Protocol: KEYWORD = {exact-once, atmost-once, timed};
    Rate: KEWORD = INTEGER;
    Retries: KEYWORD = INTEGER;
END.

TRANSPORTATION: QoSDOMAIN =
NEEDS MPS, EX;
BEGIN
    Type: KEYWORD = {socket, plug};
    Channel: KEYWORD = {shared, not-share};
    Mps: KEYWORD = MPS.K;
    Ex: KEYWORD = EX.K;
    T_binding: INTERFACE =
    BEGIN
      Result: TYPE = {ok, fail, retry};
      Unbind: OPERATION [] RETURNS [Result];
    END.
    Bind: OPERATION [if_ref: InterfaceRef]
           RETURNS [T_binding];
END.
```

Figure 4.3 defines the real-time tasking QoS domain, which consists of a TASK domain and an ENTRY domain. The TASK domain defines the attributes of real-time tasks. The ENTRY domain defines the attributes of scheduling entries. The TASKING domain has a Bind operation which allows the association of tasking resources (both tasks and entries) with interfaces. A binding management interface is also defined to allow the cancellation of a binding and the change of tasking QoS.

Figure 4.4 gives some examples of QoS expressions based on the QoS domains defined.

Figure 4.3: TASK QoS domain

```
ENTRY: QoSDOMAIN =
BEGIN
     Id: KEYWORD = INTEGER;
     Type: KEYWORD = {shared, not-share};
     EnqueuePolicy: KEYWORD = {FCFS,PB,DB,PDB,DPB};
     RendezvousPolicy: KEYWORD = {Null,PI,DI,CEILING,PDI};
     Concurrency: KEYWORD = INTEGER;
END.

TASK: QoSDOMAIN =
BEGIN
     Type: KEYWORD = {rt_periodic, rt_sporidic, non-rt};
     Tasks: KEYWORD = INTEGER;
     Period: KEYWORD = INTEGER;
     Priority: KEYWORD = INTEGER;
     Stack: KEYWORD = INTEGER;
END.

TASKING: QoSDOMAIN =
NEEDS ENTRY, TASK;
BEGIN
     Entry: KEYWORD = ENTRY.K;
     Task: KEYWORD = TASK.K;

     T_binding: INTERFACE =
     BEGIN
          Result: TYPE = {ok, fail, retry};
          Unbind: OPERATION [] RETURNS [result];
          Rebind: OPERATION [] RETURNS [Result];
     END.

     Bind: OPERATION [if_ref: InterfaceRef]
           RETURNS [T_binding];

END.
```

Figure 4.4: QoS Expressions

```
-- Communication QoS
          ((Channel == not_share) and
          (Mps.Protocol == TCP) and
          (Ex.Rate == 1000))

-- Tasking QoS
          ((Entry.type == not_share) and
          (EnqueuePolicy == FCFS) and
          (Task.Tasks == 2))
-- Session QoS
          ((priority == 3) and (deadline == 1.10))
-- Session implicit binding

{result} <- if_ref$operation(arguments) (Priority == 3)
```

5 Related work

5.1 IMAC

The Integrated Multimedia Application Communication (IMAC) architecture [TR.028 93] provides a framework which facilitates the construction of multimedia applications.

IMAC is based on the ANSA architecture and has been implemented as an extension to the ANSA Testbench. IMAC provides a mechanism for the specification of communication oriented QoS on a per-invocation basis. Interface operations may specify a set of QoS options with which they are prepared to be invoked. The QoS options are expressed as constraints on the underlying communication system. A method of mapping from application level QoS to communication level QoS is provided.

IMAC QoS extensions are based on the ANSA implicit binding model, i.e. QoS constraints are mainly associated with sessions. It does not address QoS domain issues and explicit binding issues.

5.2 Lancaster work

The Basic Service Platform (BSP) [Coulson 93] developed at Lancaster University provides an ANSA based platform for the design and implementation of distributed multimedia applications. Up to now, the platform has been mainly concentrated on engineering extensions of ANSAware for multimedia transportation and synchronization. Special rate based transportation services, synchronization services, and device management services are developed as a set of special ANSA interfaces. BSP work has resulted in numerous insight on engineering issues.

BSP adopts a typical API approach for QoS: QoS specification and management are provided by a set of standard interfaces. Explicit binding is introduced in an *ad hoc* manner for some special interfaces that provide bounded invocations. BSP does not address the full range of explicit binding and QoS issues.

5.3 OSI QoS framework

The ISO/IEC OSI QoS framework [ISO/OSI/QoS 93] concentrates primarily on QoS for OSI communications. The framework defines terminology and concepts for QoS and provides a model which identifies objects of interest to QoS. The QoS associated with objects and their interactions is described through the definition of a set of QoS characteristics. A model is defined which provides unifying concepts for the management of QoS.

The framework is intended for existing or planned OSI standards and may be applicable to other fields.

The OSI QoS framework is complementary to our work: it defines a detailed example of OSI QoS domain that may fit into our general QoS/binding model. The OSI QoS framework does not address any binding issues.

5.4 CNET work

CNET [Hazard et al. 93] is working on a global architecture for handling problems raised by QoS handling and real time. CNET work adopts a strict approach for separation of computational and engineering concerns: the programmer is provided with means to express QoS in an abstract way, and the engineering infrastructure translates the abstract QoS expressions into precise mechanisms. The main computational elements introduced are an extension to ANSA interface typing and subtyping rules for the specification of QoS constraints and an extension to the notion of binding for the dynamic establishment of QoS constraints.

CNET work proposes a formal specification language QL based on a real time logic model as its QoS language. It also proposes to use synchronous languages to express the control part of real time applications, which has inspired another relevant work at APM.

CNET work has provided many useful insight to our work in the pursue of a generic binding framework and language based approach to QoS. The CNET QoS language QL, however, is not efficient in expressing resource constraints.

6 Summary

This document presents a framework for handling QoS issues raised in the ANSA/ODP architecture. The feasibility of the framework is demonstrated by its capability for the accommodation of a real-time programming model.

The major results of this document are:

- the identification of the requirements for QoS specification
- the identification of the design constraints for QoS
- the development of the required concepts, entities and their structures for a QoS framework
- the application of the QoS framework on real-time.

The main attributes of the QoS model are separation and integration, which provides a realistic approach for the description of resource management activities and allows the evolution of technologies.

Some of the future work are

- a dedicated QoS language
- better understanding of the relations between QoS and binding
- QoS tools and supporting service
- prototyping.

6.1 Acknowledgment

The author wishes to thank Nigel Edwards, Gray Girling, Andrew Herbert, Nicola Howarth, David Iggulden, Dave Otway, Francis Wai for their review and feedback on the document.

References

[Coulson 93]

G Coulson, Multimedia Application support in Open Distributed Systems, Ph.D. Thesis, Lancaster University Computing Department, April, 1993.

[Hazard et al. 93]

L Hazard, F Horn, and J B Stefani, Toward the Integration of Real Time and QoS handling in ANSA Architecture, CNET Report CNET.RC.ARCADE.01, June 1993.

[ISO/ODP 93]

ISO/IEC JTC1/SC21/WG7, Reference Model for Open Distributed Processing: Part 1, Working Document 21/7/N755, August, 1993.

[ISO/OSI/QoS 93]

ISO/IEC JTC1/SC21/WG1, Quality of Service Framework, Working Draft #2, July, 1993.

[Li 93]

G Li , Supporting Distributed Realtime Computing, PhD thesis, University of Cambridge Computer Laboratory, Technical Report 322, August 1993.

[RC.282 91]

A Herbert, Engineering Model: Conceptual Framework, RC.282, APM Ltd., Cambridge U.K., 1991

[RC.1072 93]

G Li, Engineering Aspects of Real-Time . RC 1072, APM Ltd., Cambridge U.K., October 1993.

[RC.1150 94]

D Otway, Towards an ANSA Bindnig Model (slides). RC 1150, APM Ltd., Cambridge U.K., 1994.

[TR.01 93]

O Rees, The ANSA Computational Model, TR 01, APM Limited, Poseidon House, Castle Park, Cambridge, CB3 0RD, 1993.

[TR.028 93]

C Nicolaou, Integrating Multimedia into the ANSA Architecture, TR 028, APM Ltd., Cambridge U.K., October 1993.