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

Towards an Open Architecture for Real-Time

Francis Wai

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

(now at: ARM, Fulbourn Road, Cherry Hinton, Cambridge, UK CB1 4JN)

Abstract

This paper describes the ANSA work-in-progress on real-time, high performance systems architecture. This is essentially a paper of the “whys” and the “whats”. It attempts to put into perspective why we want to pursue real-time, high performance architecture. The benefits and the goals are elaborated. It then goes on to outline the directions and the relevant technologies in which those goals are being realised. Work is currently under way to build an ANSA real-time platform.

APM.1106.03.03 **Draft** 5 July 1994

External Paper

Distribution:
Supersedes:
Superseded by:

Towards an Open Architecture for Real-Time

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. Integrating applications are built to respond to business goals by providing value added information services in a timely and accurate fashion or by automating some previously manual or undefined service function in the business.

Because these services are driven by business goals, which themselves often change rapidly in response to business forces, it is necessary to be able to design and deploy services quickly and cheaply. Since the focus is on integration, the platforms over which the service operates can be expected to be heterogeneous and to change during the lifetime of the service. These factors lead to pressure for integrating services to be implemented above standard interfaces which can be expected to persist over technological change, and for service creation and management to be automated to a significant degree.

Since the requirement for integration services comes from meeting business goals, the designer of the service must address the costs and benefits of providing the service. There are trade-offs between precision of results, timeliness of results, scale of access and so forth which need to be resolved against business requirements. These factors turn into performance goals for the service and for its supporting infrastructure; they will affect the structure of the service itself and the ability to deliver the service over alternative platforms.

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 93] 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 Combined Object Request Broker Architecture (CORBA)

do not address real-time or performance management requirements. As relatively new technologies, attention has focused entirely on functionality; efficiency, optimizations and meeting guarantees are not addressed.

Therefore the scope of the ANSA work on performance and real-time is to define the necessary framework and to investigate candidate technology for practical standardization.

The benefits of the work are as follows:

- service providers will be able to specify and implement services that give performance and real-time guarantees, meeting their business goals
- except in the most demanding of situations, real time and performance management will cease to be special case, 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, increasing the market for the technology

2 Terminology

The terms “real-time” and “high performance” are used inter-changeably in this paper.

The term “high performance” is understood to cover the following:

- raw speed i.e. maximise throughput,
- responsive i.e. minimise latency,
- dependable i.e. the delivery of results can be expected to be on time,
- temporal constraints i.e. the beginning or the termination of an execution can be specified against a real clock.

Other desirable system behaviours e.g. the system has a peak performance rate of some number of transactions completed in some specific time period can be described as bordering on responsive and dependable.

The terms “object” and “interface” are as described in [Rees 93].

3 Introduction

Real-time systems known hitherto are restrictive in that they are constructed for particular problem domains; mission oriented applications are one such area. This is largely due to cost, performance reasons.

The use of specialised technologies of one kind or another makes the solutions harder to adapt to new requirements. Can a flight control system for planes from one manufacturer be adapted for use on planes from another? For the same make of planes, can it be scaled up as well as down? Can it be linked up with ground based radar guidance systems? And what are the costs?

New fundamental requirements arise infrequently. Most requirements either evolve with changes in business goals, e.g. banks merge, or diverged as a result of better understanding of the nature of the business. In other words, there are continuities and hence there are scope for adapting existing solutions to new circumstances.

The principles established in the ISO Basic Reference Model for Open Distributed Processing permit the construction of distributed system solutions with the capability to cope with heterogeneity, and to prepare to share, to communicate, to evolve, to federate etc. There are five projections in which an ODP system can be analysed: enterprise, information, computation, technology and engineering. The ability to scale up, for instance, may be desirable from an enterprise point of view but may require more engineering resources to be allocated. The projections permit a framework in which trade-offs are made.

Traditional real-time systems are often highly optimised, closed systems. The motivation of the ANSA activity on performance is to try to bring the ODP world and the real-time systems world together.

The approach is to identify which ODP principles need supplementing and which boundaries in the combined world where timeliness can be expected. A real-time ODP architecture will be beneficial to system builders, systems integrators in markets such as telecommunications, financial trading, office automation where openness is gradually becoming crucial to competitiveness.

This paper is organized as follows. §4 offers a definition for a generic real-time ODP system. §5 is an attempt to map out the problem space in terms of performance categories. §6 describes the desirable characteristics for real-time systems. Sections §7 to §9 identify areas of previous ANSA work for further investigations. In §10, Quality of Service (QoS) is described as the means for guarantees to be arranged. In §11, the notion and role of a service contract

which captures QoS agreements between the client, the server and the service bearer is described. §12 discusses the management of resources in terms of options and strategies. Sections §13 to §15 describe aspects of an experiment into the construction of an ANSA real-time platform. A detailed work plan on the ANSA work is presented in §16 and a summary is given in §17.

4 Characteristics of High Performance ODP systems

The four principal characteristics of an ODP system are:

- *Heterogeneous*: Components of incompatible functionalities always exist in a system for a variety of reasons such as cost, historical constraints, business policy etc. Their existence frees the system from being locked into a particular technology.
- *Federation*: Autonomy increases the need for de-centralisation and is intrinsic in a distributed system. Federation respects autonomy and is a viable basis on which common goals can be achieved.
- *Scaling*: Design decisions are made to enable a system to scale up or down without a total loss of functionality. Telephone systems are good examples. However, once the scale is beyond a certain point, the system is no longer a single entity in a certain sense. For instance, the service it provides has to be continuous and cannot be restarted.
- *Evolution*: The first two characteristics imply that the system must have the ability to evolve: by accommodating technological changes, integrating new systems or amalgamating existing systems.

A ***generic real-time ODP system*** is one where controlled performance can be tapped for use in a spectrum of integrating application scenarios. The delivery of controlled performance is the key feature, not outright performance.

An ODP system is not a static system: new technologies are introduced and must be integrated with the rest, components wear out and get replaced, components merge or split, new management functions get incorporated etc. In short, an ODP system cannot be perceived as a system with a specific performance characteristic. It is likely that it will have a range of performance characteristics. It is therefore appropriate to talk about boundaries against which guarantees can be made.

The timely availability of resources is vital to meeting real-time constraints. Resources belong to managers and their availability concerns non-computational issues such as ownership/access rights, security, accounting etc.

Also, commitment to resource allocations is no guarantee to their availability for all sort of reasons, for instance network partition. Therefore, a resource management structure is necessary for the provision of guarantees.

5 Designing for Performance

The performance requirements of distributed systems will vary widely, but can be broadly separated into five categories which distinguish activities on the basis of whether they:

- have *predictable* performance under all circumstances
- are *responsive* to external events
- are as *productive* as possible
- have no specific performance requirements
- have a mixture of performance requirements

Most practical systems will contain activities with different performance requirements.

Detailed characterisations of each of the categories listed here can be found in Chapter 4 of [Wai et. al. 93].

5.1 Predictable Performance

Time critical or hard real-time systems have predictable performance requirements. They are used for applications where timing failures are considered to be catastrophic.

Activities with predictable performance requirements are expensive in both design effort and execution resources. They also place severe constraints on the rest of the system.

In order to guarantee that the performance of a set of activities in a system is completely predictable, it is necessary to avoid all uncertainties in the execution timing, resource contention and scheduling of those activities.

5.2 Responsive Performance

Timely or soft real-time systems have responsive performance requirements. They are used for applications where timing failures must be avoided wherever possible. Timing failures in a responsive system are not catastrophic and alternative courses of action are available, even if undesirable.

5.3 Productive Performance

Systems with productive performance requirements are those that strive to maximise their throughput. They do not have any critical timing problems, only those related to liveness and fairness. Their primary use of time is to help manage their workload and resources efficiently.

5.4 Unspecified Performance

This category is really only here to allow it to be mixed with the three categories with specific performance requirements. Systems in this category have no effective control over performance. Some systems differentiate activities on the basis of priority, but do not use time as a scheduling parameter, offer scheduling guarantees or attempt to optimize performance.

5.5 Mixed Performance

Systems with mixed performance requirements present the most severe problems but deliver the most functionality. The presence of activities with more severe performance requirements constrains the guarantees and optimizations that can be delivered to activities with less severe performance requirements. This category considers the problems of delivering performance in open federated systems.

6 Desirable Features for Real-Time Systems

6.1 Predictability

Predictability is the tendency of the system to perform a set of operations in a well-defined, or determined fashion, and 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. Conversely, a fully non-deterministic system is one in which operation times do not necessarily have a guaranteed upper bound.

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 guarantee.

6.2 User Control

User control means that a user has ultimate control of the behaviour of the system. This feature comes from the fact that many real-time applications are embedded systems (which are often static systems, and therefore it is possible to control the system's behaviour) and that real-time applications have immense behaviour diversity. Fixed system behaviour cannot cater for many real-time application requirements.

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

6.3 Timeliness

Real-time applications are different from the no real-time paradigm of computation in that they impose strict requirements on the timing behaviour of the system. The correctness of a real-time system depends not only on the functional behaviour of the system, but also depends on the 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 the system employs a deadline based scheduling policy to ensure deadlines are met or to identify or cancel obsolete operations. Other required functions include the description and enforcement of temporal relations among related computational activities.

6.4 Mission Orientation

Mission orientation means that an entire distributed computer system is dedicated towards accomplishing a specific purpose 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 over

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.

6.5 Performance

Real-time applications often have stringent raw performance requirements. In this vein, the optimised integration of application software and its supporting environment is certainly desirable. 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 to maximise performance.

7 Deficiencies in the ANSA Computational Model

The deficiencies explored in this section are structural in nature. Since the computational models in ANSA, ODP, OMG, etc. aim to address closely related issues, they share many structural similarities. For this reason, they suffer the same deficiencies.

It is sufficient to examine only the ANSA computational model here.

The ANSA computational model is comprised of two parts: a construction model and an interaction model [Rees 93].

The interaction model is concerned with a scheme for invocations through object interfaces and a scheme for typing interfaces.

The construction model is concerned with the provision to,

- construct objects that conform to the interaction model, and
- be computationally complete.

The interaction model is deterministic in the sense that a server cannot start a service until a client makes a request.

This implies that the ordering of events is monotonic increasing in the time space.

The computational model does not cover the following:

- quantification of time,
- synchronisation,
- ordering,
- QoS and explicit binding, and

- streams.

A notion of quantifying time is needed because it is then possible to relate time with computations.

Synchronisation is needed because it allows conditions to be specified upon which two computations must not be performed in parallel.

Ordering is needed because it allows the expression of precedence among computations.

The notion of QoS is needed in order to objectively qualify computation. In this respect, the notions of binding and stream are required to be computationally explicit.

The ANSA computational model differs from the ODP computational model, in particular, in that explicit binding and stream interfaces are covered in the latter.

8 From Computational Model to Program Development

It is useful to distinguish between programming language semantics, the computational model semantics and the execution semantics. A programmer thinks in terms of language notations. The computational model semantics gives a formal (e.g. functional) description of the kinds of computation that can be carried out. An execution semantics gives a description of how computations are carried out. A tool chain maps a program directly (as an optimisation) onto its execution semantics or indirectly through the computational model semantics.

It would be useful if the three semantics are identical resulting in a much simplified framework. An interpreter, for instance, unifies the computational model semantics and the execution semantics.

In addition, it is important to distinguish between a programming language, its programming environment and the supporting platform or the nucleus which forms the execution environment. The ANSA computational model is not a programming environment.

In admitting time, synchronisation and ordering into the ANSA computational model, it is proposed to extend the construction model only. The intention is to keep the interaction model unchanged.

The proposed extension to enrich the construction model semantics is so that new tools and new language constructs can be explained in terms of the construction model semantics and that of the existing interaction model.

Tools are in the programming environment and can be:

- deployed by the compiler in the construction of the executable, or
- called upon during program execution.

Declarative programming is preferred over the imperative style and rely on tools to generate the appropriate imperative code. The architectural benefits are that the toolset is allowed to evolve over time and across platforms, and because the tools are faithful to the computational model semantics, the programmer is free to concentrate on the problem domain independent of the platforms.

Programmer control has been suggested to be necessary in real-time programming. Areas needed to be covered at the language level and the candidate formalisms required are in Table 1.

Note: Table 1 goes here.

9 Deficiencies in the ANSA Engineering Model

The deficiencies in the ANSA Engineering Model were first identified in the IMAC work [Nicolaou 90]. IMAC was an attempt to providing a distributed platform for multimedia applications.

Two assumptions were made in the original ANSA Engineering Model.

1. Bursty interactions will be more common than sustained bulk transfer. Therefore latency is the key factor affecting performance.
2. Communication resources are assumed to be expensive. For the purpose of scaling, they must be shared, and therefore multiplexed, wherever possible.

The first assumption means that scheduling policy is biased towards communication requests. Responsiveness is increased by quickly allocating tasks to serve requests. As a consequence, long invocations get pre-empted to make way for newly arrived requests. No resources can be allocated and guaranteed for the duration of an invocation. This is a severe handicap for multimedia communication which requires guaranteed provision of resources over a period, i.e. not just for a single invocation but for a succession of invocations.

The second assumption has led to a layered communication system. Each layer provides a number of communication channels to be multiplexed to a greater number of higher layer channels.

When data passes from one layer to the next, the thread of control at the two layers must be synchronised. This synchronisation overhead is a source of jitter.

If several channels are multiplexed onto a single channel then traffic and delay variations in one channel may adversely affect the performance of the others. This is known as *performance cross-talk*. The presence of jitters increases the likelihood of such cross-talk. If the amount of jitter is large, then it may also lead to cross-talk between non-multiplexed channels. Cross-talk makes it impossible to provide communication guarantees.

Excessive multiplexing makes it very difficult to meet application specific communication requirements or Quality of Service (QoS). If several higher layer channels, each with their own QoS requirement, are multiplexed onto a single channel, with a single QoS, then the QoS requested for the higher layer channels is likely to be compromised.

The inability to accommodate QoS and communication guarantees makes it impossible to take advantage of advances in networking. The ATM network, for instance, was explicitly designed to offer a range of communication services.

The deficiencies are being addressed by the current work. QoS statements can be declared and attached to interfaces by the programmer. These declarations are then mapped onto the underlying engineering with resource allocations secured. The issue of when multiplexing communication channels should or should not occur is tackled in the wider context of QoS. The default, in the absence of QoS declarations, is to multiplex on the use of expensive resources.

10 Quality of Service

A system is composed of components of different performance and cost characteristics. System components are service users and service providers. Applications demand from the service providers certain functionalities and a certain quality of service. The following are identified as attributes which can be negotiated as part of determining QoS.

- *Time*: e.g. what sort of delay or jitter bounds can a service user tolerate? What sort of deadline does a service user have?

- *Space*: e.g. what is the bandwidth requirement of an interaction with the service?
- *Cost*: e.g. what is the cost for not carrying out a service within time? What are the benefits of carrying out a service on time?
- *Criticality*: e.g. what is the consequence of a service failure to an external environment?
- *Dependability*: e.g. what is the probability of failure? What is the probability of survival against failures?
- *Precision*: e.g. what are the acceptable QoS trade-offs? For example, how many video frames need to be transmitted in a certain time frame in order to maintain a "coherent" picture at the receiving end?

It can be argued that for timeliness and performance, only the first attribute is relevant. This is an over simplistic view. The allocation of system resources to meet various QoSs simultaneously means that some requests may have to be deferred. For instance, the consequence of failing in life-critical system can be catastrophic with loss of lives. This has to be avoided at all costs and so deadlines are missed. The use of shared resources therefore has inherently a real-time element and brings into play all of the other attributes.

11 Negotiation and Contract

Quality of service must be negotiated between service providers, service consumers and any supporting infrastructure (the service bearer). A contract is a binding set of agreements arising out of the negotiation. An arbitration (or billing or both) authority oversees the undertaking of the contract by all the parties concerned. Figure 1 illustrates the relationships among the service provider, service consumer, service bearer, the arbitration authority with respect to the contract.

Note: Figure 1 goes here.

In establishing a contract for QoS, the following kinds of guarantees can be negotiated:

- *Absolute*: The guarantee is delivered as demanded.
- *Pre-emptive, Negotiable or Voluntary*: The guarantee is delivered as requested but subject to withdrawal.
- *Time-variant*: The guarantee is delivered at a certain success rate over a sample period.

- *Statistical, Probabilistic*: The guarantee is delivered at an average success rate over the sample.
- *Conditional, Selective*: The guarantee is delivered upon the guarantee of some specific resources.
- *Atomic*: The guarantee is delivered all or nothing to every member of a group of service consumers or service providers.

ODP has already in place the concepts of trading, binding and interaction which facilitate service provision, service procurement and the detection of service failures.

11.1 Trading

Service providers advertise themselves through the trader. Service consumers look for available services that fit their requirements from the trader. A service provider provides a particular service and guarantees a certain quality of service. A service consumer requires a particular service and may require or be prepared to accept one or any provider within a range or on a list of quality of service.

There is no guarantee the required QoS can be met. The service consumer must accept all the termination conditions associated with the chosen provider.

11.2 Binding

The result of invoking a trader or a name server may be zero, one or more suitable service providers. A binding establishes communication ability between the consumer and a provider. The consumer can expect a commitment to service from the provider bounded.

A binding may be static or dynamic in the following sense. A static binding means the providers remain committed until the binding is explicitly destroyed. A dynamic binding means some provider may withdraw its commitment, or get substituted, or get replaced by a most-up-to-date version during the lifetime of the binding. To the service provider, a dynamic binding allows an exclusive service to be withdrawn if and when needed.

11.3 Interaction

Use of a service commences with an invocation from the consumer.

The provider expects to be invoked through a named operation in the advertised interface. It expects the types and number of parameters to

conform to the advertised operations. If a particular sequence of invocations is expected, it is explicitly advertised and an internal mechanism is in place to check the ordering and to raise a termination if necessary.

The service consumer expects the delivery of results to the quality of result specified. This may mean a certain precision and/or within a certain timing constraint. It should be prepared to handle any of the advertised termination conditions.

12 Resource Management

12.1 Functions

QoS guarantees are achieved by the provisioning of adequate resources in a timely manner.

There are two management functions: the policy for resource allocation and resource scheduling. Resources are allocated to satisfy demands. Resource scheduling organises the availability of resources in keeping with the current allocation policy.

In addition to explicit return of resources, resources can be temporarily re-assigned, whilst one activity is synchronised, awaiting another. Scheduling and synchronisation need to be carefully inter-related.

12.2 Allocation Policies

There is a spectrum of policies for resource allocation. At one end of the spectrum are the so called “demand driven” policies where resource allocations are open to demand. An example is time-shared, multiple users systems. At the other end are the “completely predictive” policies where resources are closed to demands and instead they are pre-allocated. Embedded mission-oriented systems are in this category.

Between the end points of this spectrum are degrees of “statistical predictability” where resource allocations are open to demands according to urgency or priority of the activities in hand.

12.3 Scheduling Strategies

The following scheduling strategies can be employed at any time.

- *FIFO*: Resources are made available to activities that have waited for them longest.

- *Pre-emptive, priority-based, deadline-based*: Resources are made available to the most critical or urgent activity or the one whose deadline is the nearest. An activity may have resources pre-empted to make them available to another.
- *Fair share:* Resources are quantized and allocated to all activities cyclically.
- *Round robin*: Resources are made available to activities which are ensured a chance to run until termination conditions arose.
- *Reservatio*n: Resources are partitioned and each partition is only available for scheduling particular activities.
- *Based on resource requirement*: Activities are scheduled according to the relative size of their resource needs.

12.4 Synchronisation Strategies

The following serves as a basis for synchronisation strategies for resources.

- *Shared*: No denial to access is possible.
- *Exclusive*: No access is granted, except to the current owner.
- *Pre-emptive*: Access is open to all but can be closed at any instant.

With exclusive access mode, there is a phenomenon in priority-based systems known as “priority inversion”. This happens when a low priority activity impedes the progress of a high priority activity by acquiring exclusive access to a shared resource, and the low priority task is itself pre-empted by middle priority tasks.

Two synchronisation protocols were reported in [Sha et. al. 90] which tackle this kind of problem.

12.5 Federated, Cooperative Scheduling

Federation implies an agreement between parties on a common goal without compromising private, proprietary, independent knowledge. The scheduling of tasks in a federated system requires negotiation which might involve legal, contractual matters.

Cooperation implies the advancement towards a common goal based on shared information. The scheduling of tasks in a cooperative system requires exclusion mechanisms so that no task is scheduled more than once.

12.6 Resource Reclamation

Resources may be reclaimed in one of the following ways:

- *Garbage collection*: Resources no longer referenced can be freed.
- *Negotiated*: Resources are re-allocated only when the owner agrees.
- *Non-reclaimable:* Once allocated, resources cannot be reclaimed until explicitly relinguished or reaching same termination event.
- *Time-out*: Resources are re-allocated when the owner fails to respond to polls conducted over a period of time.
- *Pre-emptive (with and without notification)*: Resources may be taken away and the owner may or may not be informed.

13 Technology Baseline

13.1 Contributory Technologies

Over the past decade or so, bodies of technologies had matured and stabilised in each of the foci viz. real-time systems, open systems and object-oriented systems. These technologies include,

- Real-Time Systems
  - real-time operating systems
  - microkernels
  - real-time scheduling
  - real-time communication
  - multimedia
  - ATM networks
- Open Systems
  - generic IPC
  - RPC and stub technologies
  - client-server model of interaction
  - transparency functions
  - ANSA, CORBA, DCE
- Object-Oriented Systems
  - classes, objects and class hierarchy

— encapsulation and object invocations

— implementation classes, class types, type hierarchy and subtypes

— object-oriented databases

— Smalltalk, C++

Real-time system technologies provide the functionalities necessary to permit guarantees to be made so that computations are carried out with adequate resources or on time.

Open system technologies provide the functionalities for distribution, evolution, federation and scaling.

Object oriented technologies provide the functionalities for large scale software development, reuse and maintenance.

13.2 Distributed System Environments

A distributed system environment is one that provides a set of abstractions and tools to support program development in a distributed setting. In addition, applications are supported by a set of distribution transparency mechanisms. These free application designers and users from the technological complexities.

Remote Procedure Call (RPC) and the client-server model of interactions are widely accepted as the *de facto* technical apparatus.

It is now recognised that transparencies for distribution include the following [Herbert 91]:

- *Location*: masking the physical location or configuration of objects to enable location/configuration independent references to objects to be transferred between objects.
- *Access*: masking differences in data representation and invocation mechanisms.
- *Concurrency*: masking scheduling of overlapping invocations on shared states to achieve performance and consistency.
- *Replication*: masking duplication of objects to enhance availability.
- *Failure*: masking recovery of services after failures.
- *Migration*: masking relocation of services between invocations to achieve load balancing and reduce latency.
- *Liveness*: masking the automated transfer of objects between storage media to optimize the use of processor and address space.

13.3 Real-Time Distributed System Environments

Despite the relative maturity of distributed systems research, real-time distributed systems remain a neglected, if not un-addressed, topic. Base technologies such as microkernel, ATM networks etc. providing real-time services already exist. However, most distributed system environments support few abstractions to exploit those services.

Therefore a principal aim of the work is to architect to allow real-time features of base technologies to be exploited at the distributed system environment level.

Work on distributed systems has hitherto been concentrated on RPC [Birrell and Nelson 84]. A common mis-conception is that an RPC-based distributed system environment is not a suitable baseline technology for real-time applications because RPC is often criticized for poor performance. This is true to a certain extent. As technology progresses, there will be faster and faster RPC systems. At the current state of affair, it is possible to provide milliseconds level RPC calls. There are already reports of systems that can support microseconds level RPC calls [Biagioni et. al. 93] [Johnson and Zwaenepoel 93]. Furthermore, it is always possible to perform multi-threaded RPC calls.

Fast execution helps in meeting timing constraints, but the most important property of a real-time system is predictability. The object of real-time computing is to meet real-time requirements, not necessarily fast *per se*.

14 The Supervisory Control Function

Real-time systems span a wide variety of application domains including military command-and-control, industrial process control, financial trading, auto-pilot in aviation and so on.

Whilst it is recognised that there may not be a system solution for all, the approach adopted for the work is one of supervisory control [Northcutt 87]. This is in contrast to low-level, synchronous sampled data loop functions such as sensor/actuator feedback control, signal processing, priority interrupt processing and so on.

Note: Figure 2 goes here.

Supervisory control is a middle-level function (see Figure 2), 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.

The quantitative behaviour of the supervisory control function is one of reactive system[1] (Figure 3). The objects the supervisor manages are asynchronous with respect to the nucleus and other objects in the system. The importance of the supervisor being reactive is that it maintains a permanent interaction with them all. The interactions between any object and the supervisor are instantaneous i.e. it takes zero time for the system to respond; and the supervisor synchronises the input event of an object with the output event of its communicating counterpart. The communication cost for co-located objects is insignificant. Moreover, the execution behaviour of an object is guaranteed to be deterministic every time.

Note: Figure 3 goes here.

15 Infrastructure

15.1 Real-Time Programming Model

The essence of a real-time programming model is to provide the basic abstractions so that timing constraints of real-time computations are amenable to guarantees.

A difficulty is that the execution time of a piece of software is determined not only by the raw processor speed, but also by the policy for shared resources. The real-time response of a time-shared system depends heavily on its scheduling policy. In most high level languages, this dependency is considered as non-essential detail that is to be hidden from the programmer. As a result, the performance of software written in those languages becomes vulnerable to changes in strategy for resource allocation. These are outside the control of the programmer. More complex resources such as communication subsystems further accentuate the problem with the introduction of (sometimes distributed) resource allocation algorithms to facilitate end-to-end guarantees.

The real-time programming model currently being investigated is based on the ANSA computational and engineering models. In ANSA, objects provide

1. The relationship between reactive system and synchronous programming is explained in chapter 3 of [Wai et. al. 93].

the basis for distribution, and interfaces of objects provide service access points. Named operations of an interface provide the actual services. Abstractions, mechanisms and policies are being developed to allow the programmer to access and control resource allocation.

Tasks (representing processor resources) and communication channels (representing communication resources) are considered the most important system resources. Both static resource allocation, viz. the allocation of system resources to interfaces, and dynamic resource allocation, viz. the allocation of system resources to invocations, are supported.

Predictability, user control and mission criticality are the main concerns of the real-time programming model.

15.2 Real-Time Communication

Real-time applications present more complicated functional requirements to the underlying communication systems. This section outlines some mechanisms to providing such functions within an RPC-based communication structure. Three extensions aimed at making the ANSA communication system more suitable for real-time applications are identified.

The three proposed extensions are to be integrated within a coherent architecture to provide a communication infrastructure.

15.2.1 Parallel Protocol Stacks

Multiplexing has hitherto been the popular technique for better utilisation over a protocol stack, an expensive resource. There are a number of drawbacks to multiplexing including the so called performance cross-talk.

Parallel communication protocol stacks would allow the pre-allocation of communication resources and the removal of layered multiplexing. Under the new regime, it is possible to differentiate urgent communication requests from the rest.

15.2.2 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 have to deal with the possibilities of concurrent access to shared resources.

- Latency in accessing resources is indeterminate and communication failures do occur.

The semantics of remote calls are implemented by RPC protocols. Two often referred to semantics are *exactly-once* and *at-most-once*. Arbitrary delays (or jitters) in RPC invocations with deadlines cannot be tolerated. The solution adopted is a deadline-sensitive RPC protocol. This allows the specification, the detection and the enforcement of timing constraints of individual invocations.

A timed RPC protocol obviates the need to monitor and manage timing constraints by the programmer.

15.2.3 Decomposable RPC Protocol

An RPC protocol is made up of several layers of protocols; each layer performs a function on top of the one below. These functions are typically reliable transmission, fragmentation and messaging. The RPC protocol currently being investigated is oriented towards functionally decomposable. The aim is to allow the synthesis of protocols to meet different invocation semantics i.e. exactly-one or at-most-once. For instance, at-most-once semantics can be achieved by composing the protocol for fragmentation on top of that for messaging without involving reliable transmission.

The decision to support a functionally decomposable RPC protocol is a corollary of the overall system philosophy of allowing the programmer to customize the system to meet the application specific requirements.

15.3 Temporal Synchronisation

Real-time activities have different and when in cooperation, related timing constraints. The cooperating activities are temporally related to each other so that their timing constraints can be met. Temporal synchronization must be supported. Requirements for this facility are as follows:

- the capability to express different types of timing requirements
- the provision of an abstraction based on a model that makes the program amenable to temporal correctness analysis
- the separation of concern to ensure that timing constraints of concurrent but non-cooperating activities do not intertwine with those in cooperation
- the enforcement of timing constraints among cooperating, possibly distributed activities

Timeliness and predictability are the main concerns of the temporal synchronization facility.

16 The Work Plan

The ANSA phase III workprogramme on real-time, performance management is comprised of five work packages. These are performance requirement modelling (P1), programming abstractions and management model (P2), engineering model (P3), CORBA (P4) and optimisation (P5).

This paper is a summary of [Wai et. al. 93] and [Li and Otway 93]. The two documents form the deliverable of work package P1. They outline the overall objectives for the workprogramme.

Activities arising from P2 include extension to the ANSA computational model for reactive systems, streams etc., resource management for performance, and programming language extension and the supporting tools.

Work package P3 covers an engineering model and the re-design of the nucleus of the existing ANSA testbench.

The proof of viability of the concepts derived from work packages P1, P2 and P3 is the aim for P4 and the candidate alternative technology is CORBA. The experiments are aimed at incorporating performance functions into CORBA with a view to recommend changes to it.

Work package P5 is to allow experience gained over all other work packages for optimizing the nucleus and the tools.

17 Summary

High performance systems are traditionally closed systems. A closed system exhibits many desirable deterministic behaviours. A faulty component, for example, can be quickly identified. Closed systems rely on technologies which if not carefully designed and accepted may have a short evolution path. On the other hand, the proliferation of hardware as well as software technologies and their advancements are at an ever increasing pace. Closed systems suffer from the inability to take advantage of such diversity in functions and improvements.

The demand for integration comes from the two ends:

- real-time systems are becoming more distributed
- more and more real-time requirements are placed on distributed systems

An example is the international telephone networks. As regulations on national telecommunication are being liberalised on a world-wide basis, the federated management of this global telephone network would become the

biggest distribution application there is. It would demand real-time resources and cross boundary cooperation is unavoidable.

The goal of the ANSA work on real-time, high performance systems is to provide a system control function without compromising the system's ability to evolve, to federate, to scale up or down and be heterogeneous. This system function when applies to islands of workstations, PCs, database servers would yield an integrated platform where millisecond response time can be achieved and pre-runtime guarantees or warnings on application timing requirements are given.

18 Acknowledgement

The work reported here is very much a team effort. Members of the Performance task group Dave Otway, Nicola Howarth and Guangxing Li, and Andrew Herbert deserve special thanks. Without their inputs, this paper would have been impossible.

19 Epilogue

The results of the ANSA Phase III programme are available to sponsors and vetted bodies. Documents describing the results can be obtained through our 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.

Among others, the following papers by the performance task group are currently available from the server:

1. A Performance Framework (APM.1137)
2. A Real-Time Programming Model for CORBA (APM.1059)
3. Engineering Aspects of Real-Time (APM.1072)

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

References

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

BIRRELL, A. and NELSON, B. Implementing Remote Procedure Calls. ACM Transaction on Computer Systems. 2(1):39-59. 1984.

HERBERT, A. Engineering Model: Conceptual Framework. RC.282. APM, Poseidon House, Castle Park, Cambridge, CB3 0RD, England. 1991.

HERBERT, A.J. An ANSA Overview. To appear in IEEE Network, January 1994.

ISO. Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model.ISO 10746-3. 1993.

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

LI, G. and OTWAY, D.J. Engineering Aspects of Real-Time. APM.1072. APM, Poseidon House, Castle Park, Cambridge CB3 0RD, England. 1993.

NICOLAOU, C.A. A Distributed Architecture for Multimedia Communication Systems. Ph.D Thesis. Computer Laboratory, University of Cambrige. 1990

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

OMG. Object Management Architecture Guide. OMG TC Document 92.11.1. 1992.

REES, R.T.O. The ANSA Computational Model. AR.001. APM, Poseidon House, Castle Park, Cambridge CB3 0RD, England. 1993.

SHA, L., RAJKUMAR, R., and LEHOCKZY, J.P. Priority Inheritance Protocols: An Approach to Real-time Synchronisation. IEEE Transactions on Computers, 39(9):1175-1185. September 1990.

WAI, F., OTWAY, D.J., HOWARTH, N.J. and HERBERT, A.J. A Performance Framework. APM.1137. APM, Poseidon House, Castle Park, Cambridge CB3 0RD, England. 1993.

Biography

Francis Wai

Francis Wai obtained his Honours degree in Mathematics from Bedford College, University of London. He went on to gain an MSc in Computing from Imperial College, and then a PhD in Computing Science from Glasgow University. At Glasgow he worked with Prof. Atkinson on persistent programming with a proposal on a generalisation of persistence over a networked environment. Since 1988 he had worked on a number of Esprit projects including PISA and COMANDOS. His research interests in those projects include language semantics, distribution architecture and types. He joined Dowty Communications towards the end of 1991 as a technical consultant and was seconded to work on the ANSA project. At the time of writing this paper, he was a member of the technical staff of APM Ltd. He is currently a Principal Software Engineer at ARM (Advanced RISC Machines Ltd.) in Cambridge.

Figures and Tables

Table 1: Language concepts and formalisms

Concepts	Formalisms
Temporal synchronisation, or synchronisation between cooperating threads over the time domain	Timed path expressions
Resource management	Deadlines, resource pools etc.
Deterministic execution behaviour over the time domain	Events, signals and reactive systems

Figure 1: Contractual relationships in QoS

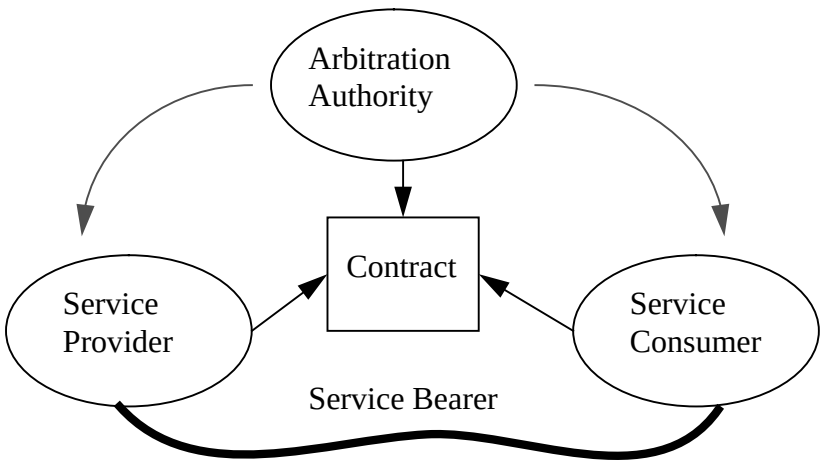

Figure 2: The Middle-Level Supervisory Control Function

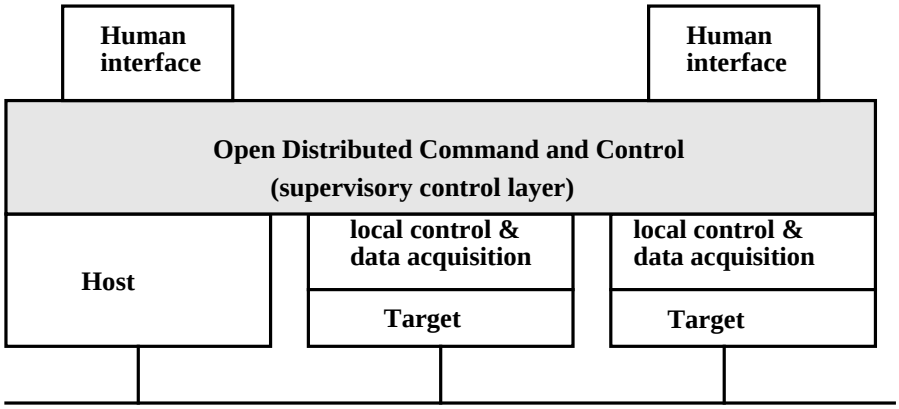

Figure 3: The Synchronous Nucleus