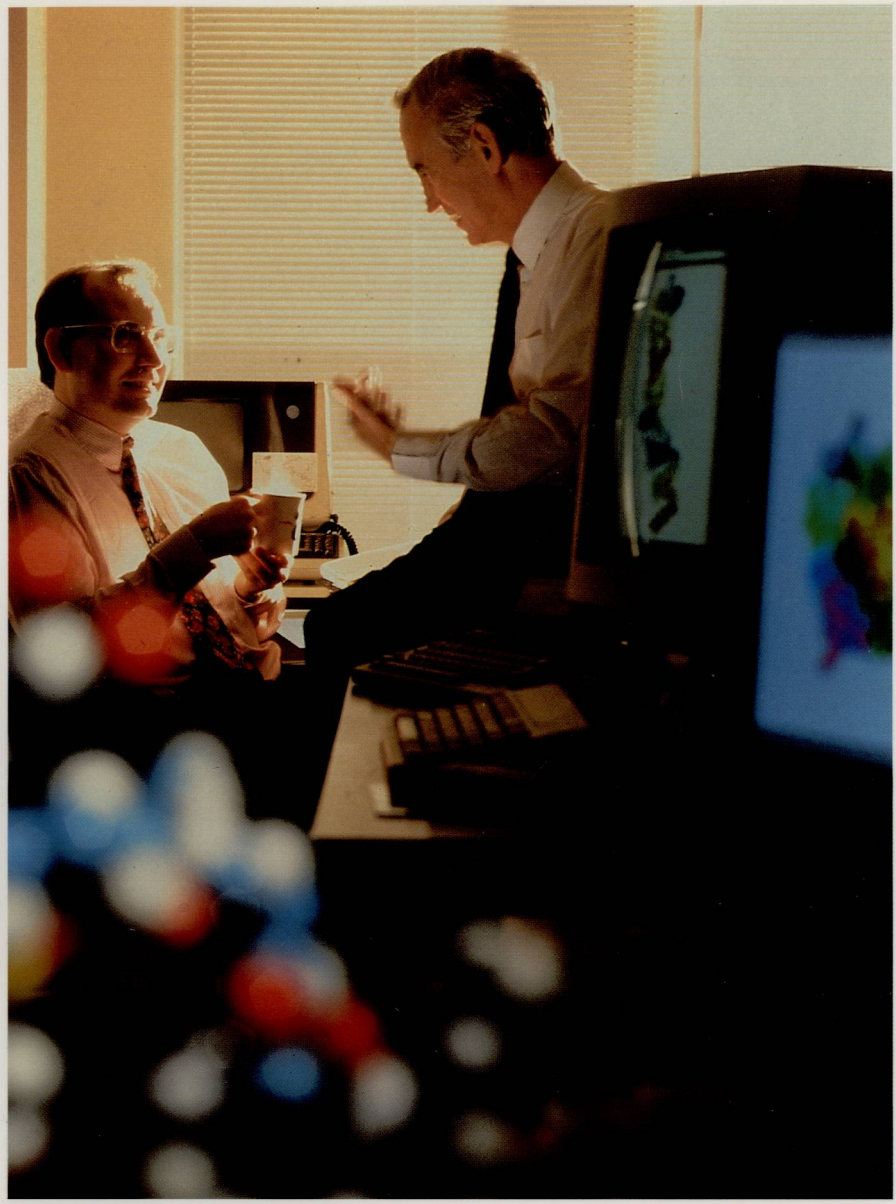
Volume 9 Issue 2 November 1994



ICL

Ingenuity

THE TECHNICAL JOURNAL

Ingenuity™

ICL's Technical Journal

Editor

J.M.M. Pinkerton

ICL, Lovelace Road, Bracknell, Berkshire. RG12 8SN, England.

All correspondence and papers to be considered for publication should be addressed to the Editor.

The views expressed in the papers are those of the authors and do not necessarily represent ICL policy.

Published twice a year by Multimedia Publishing Unit, ICL, Bracknell and Manchester.

1995 subscription rates (including postage & packing):

	UK and Europe	Rest of World
Annual subscription	£72	$144
Single issues	£43	$86

ISSN 1354-9952

Ingenuity

Volume 9 Issue 2

Contents

Other papers

Other services (advertisements)

Front cover: Professor Bill O'Riordan (right), ICL's Head of Research and Advanced Technology, discussing **ElipSys** with Dr. Dominic A. Clark, Senior Research Fellow of the Imperial Cancer Research Fund.

ElipSys, ICL's constraint logic programming language for planning, scheduling and modelling systems was developed in collaboration with the European Computer Industry Research Centre and has been used by scientists at the ICRF to enhance and extend existing methods for the prediction of protein structures that may be involved in cancer.

Traditional methods of prediction are often time consuming, expensive and impractical. **ElipSys** is 60 times faster than the best existing methods.

Editorial Note

This issue includes five further papers related to client-server topics in addition to those printed in the May 94 issue. Other papers discuss, ICL's novel **search** ***accelerator***; the concepts underlying ATM, an advanced general purpose communication architecture capable of handling all forms of information including speech, static and moving images and data in character form - a scheme still under development which some think will become the preferred carrier for information on the super highway; the Security Framework of POSIX and ICL's own form of Teleservice.

Opinions about how best to approach the design of client-server systems, range from the entirely pragmatic to the profoundly abstract and theoretical. As engineering history shows, a wholly pragmatic approach may be deficient because no theory was applied to indicate its inherent limitations. In the early stages of any new art, theory may be incomplete or, at worst, entirely missing. (Many will recall the dramatic film of the Tacoma Narrows bridge disaster). With software whose behaviour can often be many orders of magnitude more complex than that of any purely physical system, the human mind needs straightforward models to help imagine how a complete software structure may be described in such a way that its behaviour may be first imagined and then correctly analysed. This is the aim of the research described in the first three papers in this issue from the ANSA project in Cambridge - a project not yet finished. Nevertheless the approach is convincing and should already be a help in considering the optimum definition and the allocation of subordinate tasks between clients and servers, especially when pre-existing components have to be integrated into larger systems.

Finally, the issue includes a personal memoire by Professor Frank Land (prompted by the recent book on LEO by Peter Bird) describing some of the software written for LEO, the original data processing computer first put to work by J Lyons Ltd. in the early fifties. The approach was in many ways years ahead of its time.

Establishing Co-operation in Federated Systems

Mike Beasley, Jane Cameron, Gray Girling,, Yigal Hoffner, Rob van der Linden, Gomer Thomas

ANSA Project, Architecture Projects Management Limited, Cambridge, UK

Abstract

Organisations have to respond quickly and flexibly to change, and large organisations are evolving into networks of co-operating small organisations. Mergers and de-mergers are common.

These changes in organisations will have a corresponding impact on their computer systems. Distributed systems will be larger in scale than now and will cross more organisational boundaries. They will evolve as the demands of their users change, and components will be upgraded while the system is running. The software development for such systems will be distributed.

This paper addresses these concerns, and considers a number of areas. First there is the classification of boundaries, and consideration of suitable mechanisms to deal with them. The next concern is making the necessary information available to the interacting objects (trading). Finally the requirements of trading in terms of data management are discussed.

1. Introduction

1.1 Purpose and audience

With the ever-growing scale and diversity of distributed systems, organisations will benefit from the ability to extend and interconnect systems in order to respond quickly to business opportunities and challenges.

Federation is concerned with facilitating co-operation between autonomous organisations for the purpose of sharing services and resources. Barriers to such co-operative computing may be technical, organisational, administrative or business-process related.

There is a need to tackle the problems associated with federation in a comprehensive and consistent fashion as they are strongly inter-related:

- trading of services across boundaries
- interception and adaptation at boundaries
- management of Quality of Service (QoS) aimed at delivering guarantees (dependability, security and performance) across boundaries.

These problems are in addition to the current industry focus on platforms for application integration (CORBA, DCE, OLE2). Solutions to federation can help solve interworking problems between these platforms.

This paper identifies the problems associated with federation and outlines the current ANSA work aimed at finding solutions. The ANSA architecture (both directly and via ISO ODP) is part of the OPEN*framework* reference architecture for Distributed Application Services [Brenner, 93].

The issues discussed in this paper will be of interest to the following people:

- *domains and boundaries*: system designers and managers
- *data management*: systems integrators and database users
- *interception*: systems integrators.

2. The Need for Federation

2.1 The trend toward large scale, federated systems

The need for faster response to ever-changing business situations has led organisations to move from monolithic organisational structures to networks of co-operating sub-organisations. Organisations typically require computing systems which reflect their organisational structure. The demand for more flexible distributed computing is high.

At the same time, advances in technology are fuelling an ongoing trend towards increasing computing power, storage capacity, communication availability and bandwidth, all at decreasing cost. The increasing technology base for distributed computing enables computing systems which are:

- more widely used
- larger (both geographically and in terms of computing power)
- more diverse
- more interconnected.

In short, the trend is towards large-scale, inter-organisational distributed systems. Increasing organisational demands, fuelled by improvements in

the technology base, result in a number of interoperability and federation problems to be solved by software.

Computer and telecommunications vendors must meet the growing demand for distributed computing components and services which can interoperate in large-scale, federated systems.

2.2 Overview of areas addressed

The work described in this paper is aimed at enabling large scale, federated, distributed systems to be built.

Four main objectives are defined:

- *heterogeneous interaction*: supporting the technological, naming/ semantic and object description aspects of federated interoperation
- *federated object management*: supporting the overall management of objects and resources in a federated system
- *federated development*: supporting design and development of interoperating clients and servers by different organisations
- *federated enterprise negotiation*: supporting the institutional, remuneration and administrative aspects of federated interoperation.

Support for these objectives requires analysis of the boundaries which arise, and techniques for dealing with them (see sections 3 and 4).

The requirement for interactions to cross boundaries leads to a model of interception (see section 5). This model must support the removal of barriers when interoperation is to be facilitated, and the imposition of barriers when interoperation is to be restricted.

To meet the requirements described here, access must be provided to the diverse kinds of information needed to support the four objectives listed above. This leads to the development of an extended model of trading (see section 6) and a *proof of concept* prototype trader implementation. The extended trading model must support an extensive and extensible collection of data types for information about services and offers, and must also support highly flexible retrieval of this information.

The design and implementation of the extended trader prototype itself needs query language access to data in open distributed systems (see section 7).

2.3 Characteristics of large-scale, federated systems

A large-scale, federated system typically spans organisational boundaries. Hence different authorities are in charge of different parts of the system.

Multiple authorities responsible for the creation, ownership and control of different parts of the system are subject to different needs, constraints and opportunities. Hence there are likely to be many differences between the parts.

System components must interoperate to meet the objectives of the various organisations involved. Generally these organisations have to negotiate terms and conditions for interoperation.

Systems are no longer static - they are in a constant state of flux to meet the constantly changing needs of their constantly changing user bases, and to accommodate new hardware and software.

System components are designed and developed by different organisations. Designers and developers need ready access to specifications of all parts of the system, at every level of abstraction, at the level of detail which the *owner* permits. This could include contact details to tell them whom to ask about parts of the system that they are not given access to.

Different mechanisms are in place in different parts of the system to enable interoperation, to meet quality of service guarantees, and to manage the creation, deletion and reconfiguration of objects and resources. As with any complex technology, it is important to be able to mask these mechanisms selectively from end users and/or application developers, and present them with a high-level, application oriented interface, otherwise the complexity will be a barrier to rapid application development, will make maintenance more difficult and may restrict the reuse of applications to environments with a specific configuration.

2.4 Interoperation in large scale, federated systems

To reach successful interoperation between objects in a distributed system, it is necessary either to:

- find compatible objects (trading [ISO, 94]), or
- detect incompatibility and take appropriate action where possible to bridge it (interception [ISO, 94]).

where *compatibility* must take account of all the following aspects, summarised in figure 1:

- *authority*: creation, ownership and control of resources of all kinds
- *accounting/billing*: measuring resource usage, billing for usage, and handling payment
- *management*: policies for resource allocation, monitoring, quality control, etc.
- *infrastructure*: mechanisms and protocols required for interoperation between clients and servers using different technologies

- *application*: features specific to particular instances of applications or services, such as:
  - o interface specification (e.g. its signature)
  - o quality of service (QoS) specification (non-functional aspects of services such as security, performance, dependability, etc.)

  The specifications of interface signature and of QoS are part of the application programmer's view of the system. The complex task of relating QoS requirements to appropriate engineering mechanisms can be automated, at least in part.[1]
- *model*: semantics and naming of objects and services. This falls under an information view of the system.

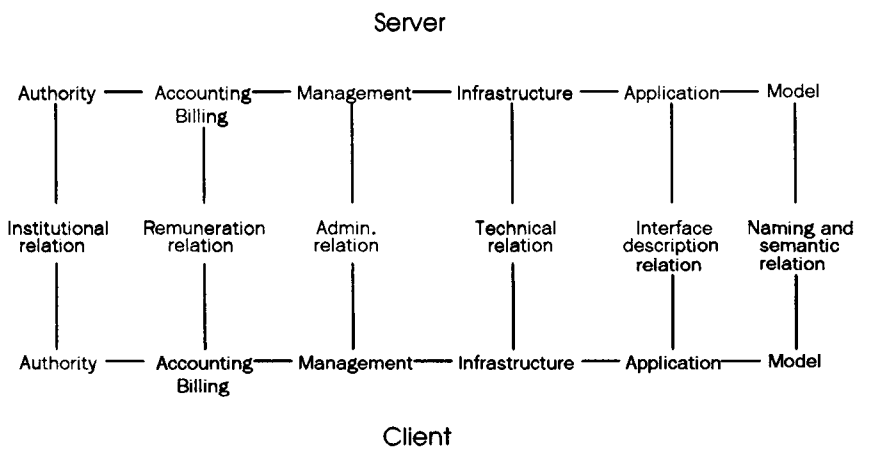


Figure 1 Key Aspects of Interoperation

Successful co-operation requires the imposition of barriers as well as their removal, for example, to ensure security, to provide performance guarantees, or to manage resource utilisation.

3. Boundaries in Federated Systems

3.1 Introduction

To support and manage interoperation in federated systems, one must first understand the different kinds of boundaries which arise in such systems. The next step is to analyse the different kinds of problems they pose and develop solutions to the problems.

---

[1] The extent to which this is possible is currently under investigation in ANSA work on Performance and Dependability.

This section presents a high-level view of the different kinds of boundaries arising in federated systems. It is a revised and expanded version of the classification of boundaries which appears in Chapter 6 of [Deschrevel, 93].

Section 4 of this paper discusses, in general terms, the kinds of problems which arise at boundaries. Section 5 discusses interception, a technique which can be used to handle many types of problems at boundaries.

3.2 High level classification

As a result of studying the relationship between components of distributed systems (see section 2.4) a number of areas have emerged in which differences between systems are likely to occur.

Almost always there will be differences in the information, procedures and mechanisms used in each one of those areas to achieve their goals.

3.3 Authority, management and administration

An administration is a body of people and facilities that manages operations, subject to the constraints and policies laid out by an appropriate authority. In a federated system there are multiple administrations, responsible to multiple authorities. Each administration and authority has local autonomy, subject only to agreements reached with other administrations and authorities. If services are provided and consumed across domain boundaries, the management in each domain is responsible for ensuring that all guarantees given by the clients and servers are kept.

There are several different types of management which concern distributed systems. Examples include:

- QoS management: providing end-to-end QoS guarantees, both within individual domains and where service provision crosses boundaries
- access control: deciding who should have access to what services when
- object management: monitoring and control of objects (objects can include parts of the infrastructure):
    - object creation, destruction and relocation
    - resource allocation and reclamation
    - configuration management
- miscellaneous management functions such as monitoring and debugging which are needed occasionally.

3.4 Remuneration

Remuneration includes three different processes:

- *accounting*: measuring the amount of work carried out by a server and calculating the charge for the service
- *billing*: charging the user for services provided

- *paying*: transferring some currency from the user to the provider of the service. This process may include sending a receipt from the provider accounting agency to the user remitting agency.

Different systems may have different views and implementations of each of these processes. In particular, conflicts may arise between systems which have different accounting and billing strategies. Remuneration is closely related to issues of trust and security. It is expedient to reach agreement on remuneration issues before consuming and providing services; for this negotiation to succeed, it is necessary for the relevant information concerning remuneration to be available at this point.

Remuneration is a topic in which there is currently a great deal of interest, especially concerning the Internet and the World Wide Web.

3.5 Service interfaces and properties

Functional and non-functional specifications of services have to be available in a form which allows the trading process to compare them and to find compatible objects. Service descriptions can be broken down into the following major parts:

- *interface type (signature)*: description of interfaces in terms of operations and data types. Type systems involve notions of conformance and substitutability; types are represented by interface definition languages (IDLs) and abstract data types.

  We understand how to formalise signatures, and how to check that the operations provided by a server meet the requirements of the client. Performing these checks has the benefit of preventing interaction errors.

- *quality of service (QoS)*: how to specify QoS requirements in a declarative manner which is translatable to a variety of platforms and mechanisms.

(This excludes the issue of the behaviour of services, which is included under modelling, below.)

Different systems will hold service descriptions in different formats - for example, there are different IDLs in current use (DCE, OMG, ...), and there are also different specification languages for system management (GDMO, SNMP, ...).

3.6 Infrastructure: engineering and technology

Distributed system infrastructures provide generic facilities which are common to a wide range of distributed systems and are used by the distributed application programmer.

We can classify the differences between such infrastructures according to the categories of information necessary for setting up the co-operation (binding) between objects (this information is contained in a DCE binding handle, ANSA interface reference, or CORBA object reference):

- *transparency mechanisms*: declarative Quality of Service (QoS) requirements translate into the information, procedures and mechanisms necessary to provide the service with the required guarantees. Different systems will use different and incompatible mechanisms; only ANSA [Li, 94] has any concrete proposals in this area.
- *communications protocols*: different protocols are used, with different guarantees and features (e.g. DEC RPC and ANSAware REX, both of which are implemented over TCP and UDP).
- *location*: different ways of addressing may exist, depending on the kind of network and communications protocols available.

To enable interoperation of clients and servers on opposite sides of an engineering/technology boundary, it is necessary to bridge the differences in some way so that bindings can be set up and messages can be meaningfully exchanged (one particularly important variety of messages is that which reports that an exception has occurred, vital to enable troubleshooting). One promising approach is to use the information used by the trading process to support the automated generation of some sort of appropriate adapter, or *interceptor* (see section 5).

3.7 Modelling: semantics and naming

Modelling concerns the way people describe the world around them. Mechanical systems such as computers can only deal with concrete representations, e.g., bit patterns. Previous work on naming models [van der Linden, 93] and the information model [Iggulden, 93] illustrate some of the limitations on the representation of concepts and the subsequent interpretation of those representations. These limitations apply also to service specifications, for instance:

- different services may have the same name in different contexts
- the same service may have different names in different contexts.

The major problem with naming and co-operation in large-scale systems is that different naming schemes make it difficult to compare names to discover whether the objects, or properties, denoted by those names are compatible.

Differences in semantics result in identical or similar concepts being represented in fundamentally different ways.

For example, two time services may provide different resolutions (1 second for one and 1 minute for another) and may vary in whether they include the date. In one sense they both mean the same thing by time, but in another sense they are different.

Moreover, production of a representation of something in one context (e.g., a specification of a service in project A) and the transfer of the representation to another context (e.g., project B) does not necessarily

convey the meaning attached to the specification. The two contexts may have different understandings of the representation used.

The major problem with regard to semantics and co-operation in large-scale systems is that it may be difficult to discover whether the semantics of an operation offered by a server are compatible with the semantics required by a client. This may be because a sufficiently complete semantic description is not available to the client, or it may be because the semantic description provided by the server is not understandable by the client.

One way to make semantic information available is to make it part of the service specification maintained by traders. A difficult issue is the form in which such information should be maintained so as to make it understandable to people and/or computer systems. An even harder problem is determining whether or not the parties have conceptual models that are compatible enough for there to be any way to compare the nature of the offered service with the client's requirements.

It will always be necessary to integrate automated checking and manual negotiation. Automated checking of interface signatures will accept more cases than manual negotiation allows, because manual negotiation will take into account greater semantic knowledge. Automated checking will be used to check that asserted relationships are sensible. Some current telecommunications research is concerned with negotiating agents and their associated knowledge bases [Griffeth, 94].

4. Dealing with Boundaries

4.1 Introduction

Boundaries define the limits of domains of homogeneity. These domains are defined by many different properties, corresponding to the different kinds of boundaries. This chapter considers the problems that the presence or absence of boundaries implies and outlines further work to be done in addressing these problems.

4.2 Multiple, overlapping domains

Two sets of objects may share some properties and not others. Thus, depending on the property under consideration, there may or may not exist a boundary between them. A division of a set of objects into domains under one property may or may not represent the same division as is formed under another property.

Thus, the different kinds of domains corresponding to different kinds of properties may have different kinds of relationships with one another. Some domains may coincide with others; for example, a security domain may coincide with a remuneration domain. Some domains may be subsets of others; for example, a single authority may have delegated administrative control of its networks to several administrative bodies, so that each administrative domain is a subset of the authority domain. Some domains may overlap others in essentially arbitrary ways; for

example, a security domain and a communications protocol domain may overlap in such a way that neither is contained in the other.

Because the properties which define domains do not necessarily relate to geographical location, domains of homogeneity do not necessarily occupy a geographical "area". However, certain properties may be characteristic of a particular computer network, and then the corresponding domains are likely to cover an "area" defined by the network.

Similarly, properties defining domains do not necessarily relate to authority. Boundaries may exist in places other than where the scope of different authorities meet.

4.3 Independent decomposition principle

The analysis of boundaries is simplified if the concerns associated with one kind of boundary are considered in isolation from other kinds. Thus, we believe that any of the boundaries mentioned earlier can be analysed and modelled independently of each other. However, while this separation is possible when modelling boundaries, thus helping to understand better the issues involved with each kind of boundary, it may not always be possible in practice.

In practice the following situations arise:

- when dealing with certain kinds of boundaries it will be necessary to deal with others. For example, the crossing of any boundary which requires looking at the contents of a message will conflict with security concerns where messages are encoded.
- in some cases it may be expedient to treat one type of boundary while dealing with another. For example, if, when dealing with a protocol boundary, the messages get decoded from one on-the-wire format to another, then other boundaries which require looking into the message content can also be dealt with.

4.4 Co-operation and independence

Two general issues at boundaries are:

- *co-operation*: there is a wish to co-operate with other sets of objects across a boundary, but there is a barrier at the boundary preventing it
- *independence*: there is a wish not to co-operate with other sets of objects across a boundary, but there is no barrier at the boundary preventing it.

Co-operation is concerned with creating one domain of interoperability out of many, and independence with creating many domains out of one.

These are the two polarised examples of likely problems. In practice the requirement for absolute co-operation, in which access is universally and transparently available, is likely to be tempered by some desire to ensure a quid pro quo arrangement (such as charging for a service) that is likely to require some level of independence to be asserted.

4.5 System management across boundaries

Another area of interest in federated systems is system management across domain boundaries.

One example is maintaining acceptable system performance, which requires managing the work load in relation to the available computing and communications resources, perhaps reconfiguring resources as necessary. In order to do this it is necessary to monitor and analyse work loads and performance measurements in different parts of the system, and to enable, disable or reconfigure the use of specific resources. In some situations this is largely a question of object interoperability, where client applications performing system management functions need to interoperate with server objects which have collected the relevant data and/or control the relevant resources. In other situations it may require interoperability among infrastructure components which are not implemented as objects, i.e., which do not communicate via object interfaces.

Another example is trouble shooting. When something fails, it is necessary to pinpoint the failure. Is it at a node or a link? Which component? Is it software or hardware? Here again both object interoperability and infrastructure interoperability may be involved. The situation is complicated by the fact that different domains may use different exception reporting mechanisms, so it may be difficult to propagate exception codes across boundaries.

4.6 Scope of work on boundaries

The boundary problems that need to be addressed are those occurring during the design, implementation and operational use of distributed computer-based services. The primary goal is to control (selectively enable and disable) interoperation between and within such services, and to facilitate system management.

Interoperation between objects (potentially from different domains) typically requires a number of different types of co-operation, including sharing responsibility for joint operation, sharing authority over it, sharing the management of the objects and infrastructure involved, and enabling the interactions that are required to carry it out (including both the provision of functionality and the transport of remuneration). Different types of co-operation may also be needed between designers and implementers that enable, encourage, discourage or prevent interoperation in later development phases.

Independence is often required when control over a domain needs to be established, perhaps to enforce security, safety, minimum quality, or charging controls or to establish common monitoring, auditing or administration procedures. In some of these cases one must consider the threat of malicious (intelligent) attempts to defeat the mechanisms which are inserted to provide independence.

The ANSA work on federation is intended to:

- identify types of co-operation and address each individually
- categorise suitable mechanisms for independence, consider their automated production, and establish mechanisms which are not overly susceptible to malicious attack
- identify the most important aspects of system management and the problems posed by the different kinds of federation boundaries, and propose solutions.

5. Interception

Interception is concerned with the information, mechanisms and processes necessary to carry out the following activities:

- detecting attempts by entities to interact across boundaries
- determining the differences between the entities and whether any intervention is necessary
- inserting the necessary mechanism(s) for enabling, disabling or monitoring the crossing of the integration boundaries
- acting on attempts to interact across the boundary.

The mechanisms to be used will in general depend on the kind of boundary being crossed, as well as the purpose of the interception.

There may be a need to modify objects or insert adaptors of some sort between them in order to make them compatible with each other, or to prevent certain undesirable interactions between them from taking place, or to monitor interactions.

A model of interception is being derived by refining the activities listed above, and by relating the model to the different kinds of boundaries which will have to be bridged in distributed systems.

The general model of interception is being tested to derive the interceptors, encapsulators, translators, adaptors, or whatever, each suitable for a particular purpose and boundary. The feasibility of combining interception strategies which deal with different boundaries is also being examined.

To prove the concepts and the generality of the model it will be necessary to develop prototypes of particular interception strategies, and test them in the context of different scenarios.

There is some experience in the area of interception which we will build upon. For example:

- the object management community (the X/Open Joint Inter-Domain Management Group) has experience in translating system management specifications into CORBA IDL (and vice versa) [X/Open 94a, X/Open 94b]

- our previous experience in other ESPRIT projects showed one way to generate application level gateways between applications in different distributed systems environments (ANSAware and DCE)
- stub compilers have been used to generate stubs (which convert arguments to a string of bytes) from IDL in DCE [OSF 92], CORBA-based products [OMG 92] and ANSAware [ANSA 93].

The model and techniques of interception are applicable at different stages of the development process. However, distributed systems are characterised by the dynamics of the applications and the changes which will inevitably take place in the configuration of the system. Particular emphasis must therefore be put on interception performed at run-time, resulting in interceptors being created, inserted and dismantled dynamically.

6. The Trading Model

6.1 Introduction

Trading can be viewed as encompassing all exchanges of information necessary to support interoperation. Thus, trading directly supports the first three objectives listed in section 2 and indirectly supports the fourth, by facilitating the interoperation of object management applications.

To see the implications of these objectives for trading and traders, it is necessary first to look at both the process model and the information model for interoperation in a federated system.

6.2 Processes required for interoperation

The following processes are required for successful interoperation in a federated system:

- search in appropriate repositories (e.g. name services, traders, X.500 directories) for information about existing or creatable objects
- select objects for interoperation by comparing the offers made by service providers with the requests made by prospective users
- resolve any barriers to interoperation. This may involve the use, or dynamic creation, of adaptors of some sort. It may also involve the dynamic creation of servers.
- carry out any negotiations and agreements required for co-operation
- bind the co-operating objects, by configuring the engineering infrastructure
- set up a connection between the objects
- carry out the interaction itself, i.e. the invocation of operations which implement the services.

These processes need not be co-located in time or space. Part of the matching process may take place at design time, and part at run time.

Binding may be dynamic at instantiation time or static at compile time. Some of the steps may be carried out by people outside the computing environment. Thus, trading needs to be viewed as a process which extends throughout the application life cycle, rather than something which happens only at run time.

6.3 Information required for interoperation

6.3.1 Symmetry of the information model

The basic information model for trading is symmetric, in that the client has to have information about the server in order to interact with it successfully, and similarly the server has to have information about the client in order to interact with it successfully (see Figure 2). Clients and servers have to trust each other - the trader is just an intermediary.

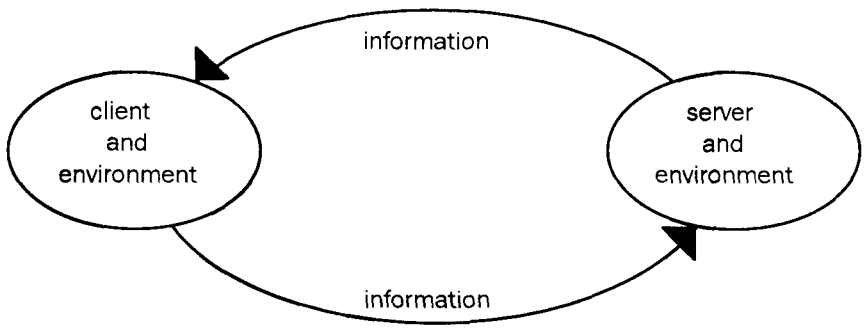


Figure 2 Symmetric Information Model for Trading

Note that there may be an asymmetry in the implementation and in the use of agents, e.g. the agent which conveys a certain kind of information from the server to the client is not necessarily the same as the agent which conveys the same kind of information from the client to the server. For example, a client may get information from a trader about allowable billing arrangements for a service, and then the client may be required to send billing information about itself directly to the server (e.g. fill out a direct debit authorisation), rather than going through a trader.

Moreover, information is not necessarily conveyed in the two directions at the same time. In fact, sometimes it is not even conveyed during the same phase of application development, deployment and/or operation. For example, a client may get information about the signature of a service from a trader during the design phase of the client, so that the client can be designed to match that signature, and then the client may provide information about the interface it needs at bind time. (Some information about the interface the service offers would have to be available again at this time, of course, to enable binding.)

6.3.2 Scope of the information model

As noted, the successful co-operation of service providers and consumers in a federated system involves finding suitable objects to interact with,

and then resolving any barriers which arise due to incompatibility. The compatibility checks necessary to ensure successful co-operation are concerned with a wide range of functional and non-functional aspects in which systems may differ.

For example, the negotiation may include authorisation policies and procedures, accounting and billing, and how to proceed if quality of service guarantees are not met. Heterogeneous interaction may require a great deal of information about the specific mechanisms being used for transparency and quality of service, as well as about communications protocols and addresses. Federated development requires detailed information about the semantics of services, as well as about the signatures of operations.[2] Federated object management may require extensive information about resource allocation and recovery policies being followed, etc.

Thus, the information model behind object specification has to be generalised. Object specification must be considerably more encompassing and incorporate information on a wide range of properties where differences between systems are expected to arise.

6.4 Trading as information service

The process of trading should be viewed not just as providing a match making facility but more as an information service which will be used by different agents at different stages of the development process. A developer interested in minimising development effort may want to browse through available services to identify candidates for re-use. A prospective customer may want to look at accounting, billing and quality of service information on a number of similar services to decide which one to use.

In these situations the trader needs to be able to support what might be called a *shopping* model, as well as the usual match making model. In the match making model a client gives the trader a complete specification of the service desired, including the interface description, and the trader returns interface references of one or more matching services to the client, together with perhaps some additional predefined information on properties of the services to assist the client in selecting one. This model is primarily intended to support dynamic binding. In the shopping model a client gives the trader an incomplete specification of the service desired. The specification may or may not include the full interface description. The client also tells the trader what kinds of information to return about the qualifying services. The client then takes this information and makes

---

[2] An increasingly important problem in large, federated systems, such as the public telephone network, is adverse feature interactions. A necessary step toward detecting and avoiding feature interaction problems is to have available a sufficiently complete specification of the services involved.

a decision on what service to use, based on some algorithm known only to the client. The client may not be interested initially in obtaining an interface reference. That may be requested later after a service selection is made. In fact, the client may first have to adapt to the interface of the selected service (perhaps, for example, using the CORBA Dynamic Invocation Interface), so there may be some time lag between the initial query to the trader and the actual invocation of the service.

The shopping model is analogous to a person shopping for something, say a fridge. The person looks at what is available from various stores, comparing features, prices, credit and payment terms, warranties, exterior appearance, energy usage, etc., finally making a decision based on some combination of these factors which would be impossible to explain in advance. The purchase is then made (analogous to binding). Certain adaptations may be made, such as determining precisely where to put the unit, based on the properties of the unit finally purchased, rather than specifying these properties in advance.

In the shopping model we have four phases rather than the current three:

1) *shopping*: trading for specifications (or for types)

2) *ordering*: trading for an object of the chosen type

3) *delivery*: binding

4) *using*: invocation

The shopping model is often required for negotiating a service (e.g., *electronic video rental* service, or on-line database service) from multiple commercial offerings, for facilitating re-use, and for federated development.

This will require a modification of the model of trading to include a variety of queries and on-going interactions aimed at supporting a dialogue with the trader. This points towards an implementation of the trading function as an information repository supporting query language access by clients, as well as the usual lookup functions.

Moreover, the object specification information should be available throughout the life cycle of an object, to assist in systems management or reconfiguration, for example.

6.5 Dynamic creation of servers and interceptors

Dynamic service instantiation (i.e. delaying the creation of a server until a client wishes to make use of the service) is described in [Deschrevel, 93]. A related capability in a federated system is the ability to instantiate interceptors dynamically. Traders play a role in both of these processes; they can provide the information necessary to determine the specifics of the server or interceptor to be instantiated, and they can also invoke an appropriate operation to initiate the instantiation.

There are several issues to be addressed in this connection:

- the provision of appropriate mechanisms to enable such dynamic instantiation

  For example, the user can be given explicit control, using the trader to locate a suitable *factory* (service creation agent), or the service creation could take place transparently to the client, as in ANSAware. There could be an agent function in the client to hide the factory.

- the provision of the necessary service specification information to support dynamic instantiation.
- the extent of the trader's involvement in the whole process of dynamic instantiation

6.6 Requirements on Trader

The above analysis leads to the following requirements on traders. A trader must:

- maintain many more categories of information than at present about services
- allow much more flexible retrieval of information than at present
- be able to participate in the automated creation or instantiation of servers and interceptors (by providing information and perhaps invoking the operation).

The above suggests a single trader, but the information and the functionality may be partitioned among multiple traders, and implemented by means of different technologies. [ISO 93] suggests that traders will be governed by various policies concerning security, searching, etc., and such policies could be supported by a trader *kit of parts* which could be used to build multiple traders with an ability to call out to agents which will enforce those policies.

These requirements are analysed at more length in section 7.

7. Data Management

As pointed out in section 6, a trading capability, which will be needed increasingly in the future, is very flexible access to information about services.

The working draft of the ISO ODP Trading Function specification [ISO 93] provides a "Search" operation which goes some distance towards meeting this need. For this operation, the client must provide as input parameters:

- service interface description (an interface type)
- matching criteria (an arbitrary boolean condition on the properties of the service)

- list of property names designating the properties for which the client wants values returned.

The trader returns a list of all the service offers which are compatible with the given interface type and which satisfy the matching criteria. For each offer in the list, it includes:

- the interface reference
- values of the requested properties.

There are several serious limitations to this "Search" operation as currently specified:

- The client may not always know the interface type of the service it is looking for. It may be looking for a service which performs a particular task and be prepared to adapt to the required interface type (perhaps using the CORBA Dynamic Invocation Interface).
- The client may not need the interface reference when the Search operation is invoked, as it may not intend to use the service right away. If dynamic service instantiation is involved, it is undesirable to provide the interface reference long before the service is needed, since the service must be instantiated before the reference can be provided.
- A "Search" operation may return a very large list of offers. There should be some specification for retrieving the list piece-wise, rather than all at once.
- There seems to be an implicit assumption that properties have simple data types. However, to support the needs of federated systems, some properties may have very complex data types. The client may want to use only certain components of certain complex properties in the matching criteria, and may want to retrieve only certain components of certain complex properties, not the entire properties.

For example, if the service is an information retrieval service, one of the "properties" of interest is the information model describing the structure and semantics of the information which can be retrieved. This is an enormously complex property. Clients may want to retrieve from a trader descriptions of certain entity sets in the information model, but not the entire information model.

In principle, one would like simply to generalise the "Search" operation to remove these limitations. However, the part about the complex data types presents significant problems, both with specifying the operation property and with implementing it.

In practice, one can achieve equivalent functionality much more easily by storing the property information in a standard database, e.g., a relational database, and allowing clients to access it via a standard query language, e.g., SQL. Such remote query language access will allow clients to select

and combine interrelated information about services in precisely the combination needed.

Moreover, trading is just one of many distributed applications which need remote query language access. The power and flexibility of this paradigm have led to its increasing popularity in client-server computing, as evidenced by the very large market today for client-server systems using SQL-based remote database access.

Therefore, an important problem to be solved is how best to support remote query language access to databases in an open distributed processing environment. Potential problems arise from the following characteristics:

- the numbers and types of input arguments and output results of a query language operation are determined by the query statement itself, often generated dynamically at run-time, in contrast to other types of operations with statically defined signatures.
- in order to provide the ability to return results of a query piece-wise, and in order to provide transactional properties across multiple operations, it is necessary to have a dialogue interaction paradigm between client and server, in contrast to other types of operations where the server need not maintain any information on the state of the interaction between operations.

These characteristics are accommodated by the basic ANSA/ODP computational model [Rees, 93,] [ISO, 94], but they are often not well supported by existing distributed processing environments.

The ANSA federation task group has developed a conceptual model for how to view databases and query language access to databases in an ANSA/ODP framework [Thomas, 94]. Current work on an experimental implementation of an enhanced trader prototype is providing a test bed for a practical implementation of this model.

The current ANSA work on Data Management, as described above, is complementary to that done within ICL for RIBA/DAIS, and described in [Crockford, 92]. The ANSA work is concerned with fitting query language access into the ANSA computational model, i.e. the model of client/server interaction; the ICL work is concerned with providing the client with a conceptual view of the DBMS that is easier to interact with.

8. Conclusions

This paper has analysed the requirements for establishing co-operation in large-scale systems in terms of a set of boundary types that may need to be crossed (federation) or established (independence). A number of consequent areas of investigation were isolated relevant to trading, interception and federated software development in such a system.

Interoperation between domains will require mechanisms addressing at least their administrative and remunerative differences in addition to differences in their interface descriptions, semantics, naming models and in the infrastructure supporting them. Similarly, isolation of domains may require barriers to be created in some of these areas.

Traders need to handle a greater wealth of information about clients and servers in order to achieve the above and this results in a requirement for a more advanced model of trading, including greater parity between the role of a client and a server and the ability to describe a greater range of objects that come into existence dynamically. In addition, the use of traditional data management technology to support this information requires the extension of existing ANSA RPC technology (e.g. dynamic typing) to conform to the existing computational model.

Boundaries need to be bridged by a variety of mechanisms, not all of which operate solely at run time. Specific instances of interception mechanisms can be derived from a generic model which is under development. Federation during the development of systems, for instance, requires the bridging of differences between tools used in the process of specification, design and implementation. This differs from, for example, a protocol gateway.

Establishing co-operation between systems is a fundamental pre-requisite for the wide-scale deployment of an electronic services market place and the availability of truly open distributed processing. This paper suggests that further advances in a broad range of areas are necessary before this goal can be achieved.

9. Acknowledgements

The authors of this paper would like to acknowledge the contribution of their colleagues in the ANSA team, particularly Andrew Herbert, Technical Director of APM Ltd. and Chief Architect of the ANSA project, who made valuable comments. Our colleagues in the ANSA sponsor companies also made a valuable contribution with their comments.

10. References

ANSA project, "ANSAware Version 4.1 Manual Set", Architecture Projects Management, Cambridge, 1993.

BRENNER, J.B., "OPEN*framework*: Distributed Application Services", ISBN 0-13-630518-0, Prentice Hall, 1993.

CROCKFORD, L.E., DRAHOTA, A., RIBA - "A Support Environment for Distributed Processing", *ICL Tech J* (Vol 8, No 2, pp 284-301), Nov 1992.

DESCHREVEL, J-P., "The ANSA Model for Trading and Federation, Architecture Report APM.1005.01", Architecture Projects Management, Cambridge, 1993.

GRIFFETH, N.D., VELTHUIJSEN, H., "The negotiating agents approach to runtime feature interaction resolution", in Feature Interactions in Telecommunications Systems, Bouwma and Velthuijsen (Eds), ISBN 90 5199 1657, IOS Press, Amsterdam, 1994.

IGGULDEN, D., REES, R.T.O., VAN DER LINDEN, R.J., "Architecture and Frameworks, Technical Report APM.1017.03", Architecture Projects Management, Cambridge, 1993.

ISO, "Open Distributed Processing - ODP Trading Function", ISO/IEC JTC1/SC21/WG7, Draft, 1993.

ISO, "Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model, ISO/IEC 10746-3", ISO/IEC JTC1/SC21/WG7, 1994.

LI, G., An Open Architecture for Real-Time Processing, *ICL Tech J*, Vol 9 3 pp. 49-63, Nov 1994.

OMG, "The Common Object Request Broker: Architecture and Specification, Document Number 91.12.1", Object Management Group and X/Open, 1992.

OSF, "OSF DCE Application Development Guide", Open Software Foundation, 11 Cambridge Center, Cambridge, MA 02142, USA, 1992.

REES, R.T.O., "The ANSA Computational Model", Architecture Report APM.1001.01, Architecture Projects Management, Cambridge, 1993.

THOMAS, C.G., BEASLEY, M.D.R., HOFFNER, Y., "Data Management for an Enhanced Trader", External Paper APM.1162.01, Architecture Projects Management, Cambridge, 1994.

VAN DER LINDEN, R.J., "The ANSA Naming Model", Architecture Report APM.1003.01, Architecture Projects Management, Cambridge, 1993.

X/OPEN, "Translation of GDMO/ASN.1 Specification into CORBA IDL", X/Open, Apex Plaza, Forbury Road, Reading, Berks, RG1 1AX.

X/OPEN, "Translation of SNMPv2 MIB Specification into CORBA IDL", X/Open, Apex Plaza, Forbury Road, Reading, Berks, RG1 1AX.

11. Biographies

Mike Beasley

Mike Beasley received a first-class honours degree in Mathematics at the University of Cambridge in 1977, and a Diploma in Computer Science a year later. He then joined ICL, working as a designer/implementor on a variety of software development projects on VME and UNIX in Kidsgrove, Manchester and Stevenage before being seconded to the ANSA

project in Cambridge in 1993. He is also a Member of the British Computer Society and a Chartered Engineer.

Jane Cameron

E. Jane Cameron is seconded by Bellcore to ANSA. Prior to joining ANSA she was Director of the Network Systems Specification Research Group at Bellcore. This group focused on problems of interoperability among Telephone service, in particular problems of adverse feature interactions. She has also worked in areas of language processing, formal specification, and graphical user interfaces. She holds a Ph.D. in Mathematics from the University of Washington in Seattle, Washington, USA and is a member of the ACM and IEEE.

Gray Girling

Gray Girling received an honours degree in Computer Science at Imperial College of Science and Technology in 1978, and a PhD from Cambridge University in 1983 with a thesis on authentication in computer networks. Following a period at Acorn Computers he joined Topexpress Ltd. where he worked for six years in the research and development of a secure computer network for the UK Government. He joined the ANSA team in 1992 after a year at Perihelion Software Ltd. He has been involved over much of this period in the production of National, European and International security standardisation. He is also a Member of the British Computer Society and a Chartered Engineer.

Yigal Hoffner

Yigal Hoffner received his BSc (Hons) in Computer Science and Cybernetics from the University of Reading in 1980. He stayed to work as a research assistant in the Microprocessor Unit at the Computer Centre from 1980 to 1983. He received his PhD in 1986 while working as a research fellow in the Computational Sciences Research Group at the department of Computer Science, University of Reading. The PhD topic was "The design of a reconfigurable multiprocessor system and its use in the solution of a class of numerical problems". He worked in the R&D department of ISTEL Ltd. before joining the ANSA project in 1986 as a full time employee; work here has included user interface, trading, management, monitoring and visualisation of distributed systems.

Rob van der Linden

Rob van der Linden received his degree in Electronics and Telecommunications in The Netherlands in 1976. He was awarded a Master's degree in electronics and computing at the University of Southampton in 1981. He has been active in communications and computing first at the University of Southampton, then at BSO in The Netherlands. There he developed various early networked and distributed computing applications including an office publishing system and a distributed database. Since 1986 he has been with the ANSA project in

Cambridge, where he has worked on object modelling, storage, naming and trading. He now manages the ANSA team.

Gomer Thomas

Gomer Thomas received a BA from Pomona College in 1962, a BA (Hons.) from the University of Cambridge in 1964, and a PhD from the University of Illinois in 1968, all in mathematics. He has spent 11 years on university faculties in mathematics or computer science, and over 14 years in industrial positions. For the past 7 years he has worked at Bellcore (Bell Communications Research), where he has been focusing on distributed data management. He is currently a secondee to the ANSA project from Bellcore. He is a member of the ACM and the IEEE Computer Society.

An ANSA Analysis of Open Dependable Distributed Computing

N.J. Edwards

Hewlett-Packard Laboratories & ANSA, APM Ltd., Cambridge, UK.

Abstract

System dependability is increasing in importance in the market place. A recent report predicts that the market for fault-tolerant systems will double in the next three years. Within the context of large open distributed systems, dependability will be particularly important: the more components a system has the greater the probability that one of those components will be faulty. Over the next two to three years, the ANSA work on dependability aims to develop the technology for building open dependable distributed systems on industry standards platforms such as DCE and CORBA. This paper looks at some of the requirements that will be placed on this technology.

A failure model has been developed; its use in the design of dependable systems is being investigated. An engineering model is being developed which will provide a choice of mechanisms, enhancing the functionality of the basic platform, so that it can meet the dependability requirements of applications. A programming model is being developed to help programmers meet the requirements of the chosen engineering. A core component of both the engineering and programming models is an extended transaction framework.

1. Introduction

System dependability is increasing in importance in the market place. A recent Gartner report predicts that the market for fault-tolerant systems will double in the next three years (from mid 1993) [Kaye, 1993]. In an increasingly fierce market, reliability and availability can have significant effects in reducing the cost of system ownership [Siewiorek and Swarz, 1992]. Within the context of large distributed systems, dependability will be particularly important: the more components a system has the greater the probability that one of those components will be faulty. In addition, openness further reduces the cost of ownership by allowing easy

integration and incremental evolution of the information system [Herbert, 1993], [Harris and Fraser, 1993].

This paper argues that the basic technology for open distributed processing is now understood: there are now several de-jure and de-facto standards that are emerging. The challenges now are to be able to deliver appropriate non-functional guarantees (e.g. reliability, availability and performance), and to be able to integrate existing services and systems into this world of open dependable distributed computing. Until these challenges are met, open distributed computing will not be used in business critical applications. No one set of non-functional guarantees is appropriate to all applications; any solution must allow the selection of guarantees to match different application requirements.

The purpose of this paper is:

- to explain why dependability is important in open distributed processing and to look at some of the characteristics of suitable technology for dependability (see section 2)
- to examine some of the problems which arise in building dependable systems and explore what the ANSA principles have to say about some of these problems (see section 3)
- to explain why an end-to-end view of dependability is important (see section 4)
- to describe the requirements for the dependability technology being developed by the ANSA project over the next two to three years (see sections 5-10).

2. Open Dependable Distributed Systems

This section looks at the need for dependability in open distributed systems, the advantages to be gained by delivering the right solution quickly, and some of the constraints on the technology used to deliver open dependable distributed systems. First, it defines what is meant by dependability within the context of the ANSA project.

The dependability of a service is described by various non-functional properties of that service, such as reliability, availability, safety and security - a comprehensive discussion of these and associated dependability concepts is given in [Laprie, 1992]. The ANSA project is concerned mostly with reliability and availability.

- Reliability is a measure of the continuity of service; a measure of reliability is Mean Time To Failure (MTTF).
- Availability is a measure of how often the system is ready for use; a measure of availability is (MTTF/(MTTF + MTTR)) where MTTR is the Mean Time To Repair the system once it has failed.

A failure occurs when there is a mismatch between what happens and what is expected. In the absence of a formal contract which makes expectations explicit it may be impossible to resolve which party is faulty: the client (system user) or the server (the system). This is discussed in more detail in section 6.

Often integrity is discussed as an aspect of dependability. The correctness of a service depends on maintaining the data in some valid, consistent form: if it cannot satisfy this constraint the service will fail.

2.1 Business-critical applications need dependability

Deploying a business-critical application or information service without any guarantees about the dependability of that application is analogous to participating in a business transaction without any formal contractual arrangements. In the absence of any contract to set expectations, there is more chance of something unexpected happening - something which may be viewed by one of the parties involved as a failure. The consequences of such failure could be severe.

Similarly it could be disastrous for a business to rely on an application without well-defined expectations - without clearly defined dependability guarantees. Hence exploiting open distributed computing to deliver business-critical information services will require the ability to offer both functional and non-functional (dependability) guarantees which are appropriate to the information service being provided.

2.2 Current technology does not address dependability

A number of de-facto and de-jure standards are emerging which incorporate technology for distributed computing: ODP [ISO 10746], CORBA [OMG 91], Atlas [UI], DCE [OSF 91] (including DME and ENCINA [Sherman, 1993]) and the various OSI standards (e.g. GDMO [ISO 10165], OSI RPC [ISO 11578], OSI TP [ISO 10026] etc.). All of these provide applications (objects) with a means of communication. Perhaps the highest level of functionality is delivered by CORBA and related products such as DAIS [ICL 93]. DAIS supports object based distributed computing: objects can invoke each other regardless of whether or not they are co-located. In addition, all these standards identify some basic services which are needed by applications, such as naming. Hence the technology for delivering basic *open distributed computing* is becoming well understood and standardised.

With the exception of ODP (which has been heavily influenced by previous ANSA work), very little work has been done on providing appropriate dependability guarantees [Herbert, 1993]. ODP with its notions of transaction, group and replication transparencies lays some of the foundations [ISO 10746].

2.3 Gain a competitive advantage: match customer requirements

One of the basic principles of ANSA is that different customers, hence different applications, will have different dependability requirements. Even within one application the different components will have different

availability, reliability and consistency requirements. Understanding the engineering and cost trade-offs in building dependable distributed systems will enable the vendors to match the dependability delivered to the requirements of the customer and the application, giving them a competitive advantage over those who cannot do so.

2.4 Gain a competitive advantage: deliver the solution quickly

Competitive advantages are also gained by being able to deliver a solution more quickly than the competition. One way of doing this is to minimise the amount of bespoke engineering in a solution. The approach should be to use tools, configuring basic standard engineering components to deliver the guarantees which are needed by the application.

2.5 The need to incorporate existing systems

The need to preserve investments in existing information technology infrastructure means that new information services will have to interwork with so-called legacy systems and yet still provide some guarantees about dependability. This means that there will be few opportunities to build systems from scratch; rather, it will be important to understand how to configure mechanisms to get appropriate non-functional guarantees from what already exists. Openness implies the ability to cope with heterogeneity at all levels: different machines using different operating systems interworking between different administrations [Beasley et al, 1994].

2.6 Hardware versus software techniques

The ANSA work on dependability is about developing concepts which can be used for open dependable distributed computing. It aims to put in place the technology which enables the construction of information services with various dependability guarantees. Since openness implies minimising the assumptions about the underlying hardware and operating system, this work concentrates on software rather than hardware techniques for dependability, and on techniques which do not require one particular underlying platform for distribution.

3. The ANSA Principles and Dependability

The ANSA principles are described in [van der Linden, 1993]. This section looks at those principles particularly relevant to dependability; the principles are divided into seven categories - each considered in turn.

3.1 Separation

Systems should be designed so that separation amongst their parts can be achieved; this means that they can be more flexibly configured. However, this can have the effect of introducing more components, thus reducing dependability.

Separation means that services may be remote. This introduces the possibility of partial failure: a failure may occur in a remote service request even though the requester's local system has not failed.

The ANSA work on dependability aims to ensure that the required dependability can be achieved in spite of the effects of separation.

3.2 Diversity

Large distributed systems will include many significantly different individual sub-systems. Data will be widely distributed with multiple representations and different consistency requirements. Different standards and different dependability mechanisms will be adopted in different parts of the system; designers need to be prepared for this; the dependability mechanisms need to allow for it.

3.3 Scaling

The dependability mechanisms used in a system must not impose constraints on scaling, and the extent to which it can be interconnected and its applications made to interwork. Scaling is about scaling up and down: mechanisms that are efficient in large systems should be designed so they are efficient in small systems or else should be replaceable by similar mechanisms which are efficient in small systems.

It is difficult to check consistency in large systems: changes may take a very long time to propagate. In the absence of any mechanisms to avoid it, different parts of the system may see changes at different times and in a different order.

In large distributed systems it is very difficult to implement a notion of universal time, or of an observer who, or which, can observe every event. This means that technologies which assume a global clock or a global ordering of events are not appropriate: they do not scale well.

Larger systems will contain more components, which increases the probability that there are one or more faulty components in the system.

3.4 Federation

Federation deals with heterogeneous authority and how to retain local control in a large distributed system spanning boundaries of authority [Beasley *et al*, 1994]. In a federated system, objects are responsible for their own dependability. In addition, objects under one authority will need to negotiate contracts with other objects subject to different authorities: there may be no common higher authority which lays down what the contract should be. The contract should state what each object is entitled to expect of the other (i.e. what the *correct behaviour* should be). Contracts can be used by arbitration or fault diagnosis services to resolve which party is at fault (see section 6.1).

3.5 Transparency

A property of a system is transparent if application programmers need not be concerned with it. The aim of the ANSA work on dependability is to hide the details of the dependability mechanisms (but not the requirements for dependability) from the application programmer.

Previous experience suggests that it is possible to make dependability mechanisms such as replication completely transparent to the programmer [Oskiewicz and Edwards, 1993]. However, there are limitations on making dependability fully transparent. These are explored in section 4.

There is no universal set of requirements for dependability, hence there is no universal configuration of mechanisms. This means that transparency must be selective: programmers can select and configure the mechanisms which are most appropriate to the job at hand.

Selecting and configuring the appropriate mechanisms is likely to be a complex and error prone task. So programmers may well be tempted to implement their own mechanisms, ignoring the ones provided because they are too difficult to understand and use. To avoid this, programmers must at least be given guidance on how to select and configure mechanisms to match the requirements of their programs. Where possible, tools should be provided to configure and select the mechanisms (this is automated transparency).

Ideally the dependability requirements should be declared as attributes of the object; tools would then configure the most appropriate mechanisms. The difficulty of capturing requirements and the lack of tools which work directly from them, means that programmers will probably have to specify the mechanisms themselves. Automated transparency techniques will configure the specific mechanisms selected. The programmer is protected from the details of the mechanisms (e.g. see [Warne and Rees, 1993]).

3.6 Concurrency

Concurrency is inevitable in distributed systems. This means that there is potential for conflicting, inconsistent changes to be made to data. Mechanisms are needed to prevent this.

3.7 Configuration

Systems evolve over time: new parts are added and old parts are removed. Detection and correction of faults should be as early as possible, ideally before a new component is configured into the system. This limits the potential for a fault in one component to cause damage to the rest of the system (fault propagation). To achieve this in a dynamic system, the description (of the correct behaviour) of a component must be on-line to allow the system management to check the behaviour of the component before installing it. Such descriptions will form the basis of the contracts described in section 3.4 and are important in fault diagnosis (see section 6.1).

4. An end-to-end View of Dependability

Dependability is an end-to-end concept. What is dependable and what constitutes a failure of an application can only be determined by understanding the application semantics: it is not sufficient to consider

dependability purely in terms of protocols provided by the underlying engineering and platforms. For example, a file transfer is completed successfully when all the file data has been safely and correctly stored in the file system of the recipient machine, not just when the data has been delivered by the network to the machine (which may crash before storing the data) [Saltzer et al., 1981]. Once the transfer is complete, the receiving machine may have an ongoing responsibility to maintain the availability of the data. Consequently, dependability needs to be considered at the system design stage and throughout the development of the system.

Some techniques have a minimal effect on the structure of the system. For example, within ANSA, technology for transparent replication has been developed [Oskiewicz and Edwards, 1993]. Provided the client or server satisfies certain (well understood) assumptions it is possible to hide replication from the application programmer, so that the decision to replicate or not can be made after the code has been written. This dramatically increases the complexity of the underlying engineering required to support the application components with consequent loss of performance (compared with non-replicated code). Hence, the scope and potential of technology for transparent dependability are limited.

Other techniques for dependability require more participation from the designer and programmer. Although the solution may be more difficult to analyse and prove, it may result in applications which need less complicated engineering to support them and have better performance. For example, suppose a service needs to be replicated. If the service can be designed so that it is immutable (does not change its state when invoked) the underlying support for replication can be made much simpler, since no mechanism is required to coordinate state changes between the replicas (a replica's state never changes). To make a service immutable may require the designer to take special steps, for example, to ensure that any state concerning the interaction between client and server is held by the client. In addition, the programmer of the service needs to avoid code which makes changes to state.

This paper describes technology which is being developed that allows designers and programmers to consider dependability issues, as well as technology which tries to make dependability transparent. The concept of selective transparency is important: hiding irrelevant detail from the programmer.

5. Dependability in ANSA

There are two basic techniques used to build dependable systems: fault tolerance and fault avoidance (sometimes called fault intolerance). Fault avoidance involves using good engineering practice to minimise the occurrence of faults. Fault tolerance exploits redundancy to negate the effects of faults.

It is important to realise that these two techniques are complementary and not alternative. Good engineering practice reduces the occurrence of faults; unless the rate at which faults occur is reduced to an acceptable level, any redundancy (fault tolerance) will be quickly overwhelmed.

The aim of the ANSA work on dependability is to develop technology which allows application programmers and system designers to use a set of simple concepts to declare their dependability requirements. These requirements will be mapped quickly and efficiently onto a rich set of engineering mechanisms which exploit various redundancy and consistency techniques to deliver the required dependability. The component technologies are:

- a *failure model* which provides the underlying concepts
- a *programming model* which provides programmers with abstractions for building dependable applications
- an *engineering model* which provides a set of engineering mechanisms and sets of standard configurations of these mechanisms
- an *extended transaction framework* providing a programming model and set of engineering mechanisms based on transactions
- a *management model* which provides the mechanisms and concepts for fault diagnosis and reconfiguration to maintain dependability.

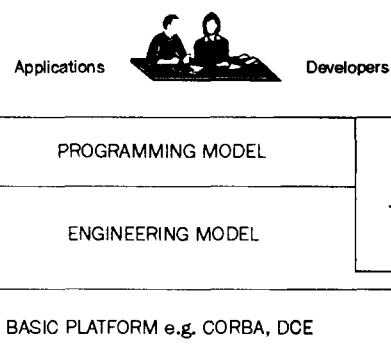


Figure 1 How the ANSA dependability work fits together

Figure 1 shows the relationship between the programming model, the engineering model, the management model and the transaction framework. The engineering model provides a set of services to enhance the functionality provided by basic platforms for distribution such as DCE and CORBA. Application programmers build dependable applications using concepts provided by the programming model and services provided in the engineering model.

The system designer needs to elucidate the requirements of the application components for dependability (from the supporting

engineering) and also the requirements which the engineering components impose on the behaviour of the application components. Designers need to be familiar with both the engineering and programming models to make the trade-offs between what (dependability) can be provided by the engineering and what can be provided by the application components.

The role of the engineering model is to provide a choice of services. It helps the system designer to choose the services which satisfy the application component's requirements (for dependability). The role of the programming model is to ensure that the application components meet the requirements of the engineering components. The role of the failure model is to provide the concepts for stating requirements.

The management model is concerned with such issues as the maintenance of dependability in the presence of faults (fault diagnosis and reconfiguration), how to install new applications and how to upgrade existing ones. The services required for management form part of the engineering model. In addition, some concepts in the programming model may deal specifically with management issues.

The use of different kinds of transactions to build dependable systems is being studied. The services provided by the extended transaction framework will form part of the engineering model. The framework will also provide abstractions to help programmers use these services. These abstractions are part of the programming model.

The remainder of this paper describes the above work in more detail. At the time of writing, the failure model has been developed, development of the extended transaction model is ongoing and work on the remaining areas is just beginning.

6. Failures and the ANSA Failure Model

Understanding the concept of failure is crucial to building dependable systems. This section looks at failures: how to understand who is responsible for a failure and the role of failure models. We begin with a summary of the ANSA failure model [Edwards and Rees, 1994].

The ANSA failure model assumes that a system is composed of components which can engage in *events*[1] which are observed by other components in the system. An *event* is considered to occur with some value at some time, by the observer; there is no notion of a global observer, a global ordering of events or a global time. An event which occurs is called an *occurrence*. The model defines *expectation regions* (a region in a *value x time* space) which define a time interval and a restricted set of values within which a component expects to observe an event. Boundaries can be drawn around an object's expectation region by

[1] Here the word *event* is being used in a specialised technical sense, rather than the general English sense. An event may or may not occur.

determining an object's expectations: what it expects from the objects with which it interacts. Three examples are shown in figure 2.

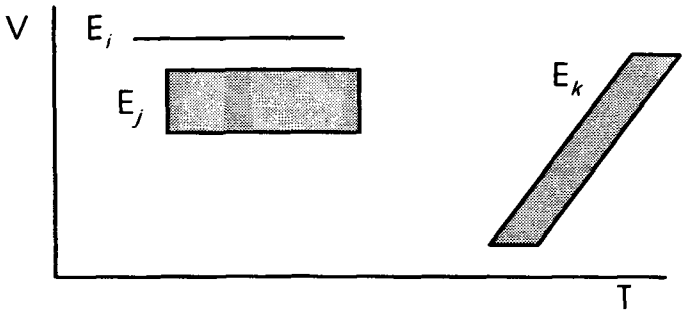


Figure 2 Expectations

The three examples are:

- E_i: a particular value is expected in some time interval
- E_j: one of a range of values is expected in some time interval
- E_k: one of a range of values is expected in some time interval but the value is related to the time at which the event occurs

A *failure* is a mismatch between an occurrence and an expectation. For example, an occurrence which does not match what is expected: any occurrence not in regions E_i, E_j, or E_k would be a failure. The absence of an occurrence when there is an expectation of one is also a failure; this kind of failure is conventionally called an omission failure [Laprie, 1992].

Currently we are investigating how to use this failure model in the design of dependable systems. The methodology we are testing is as follows. Having identified the major application components (clients and servers) using appropriate modelling and design techniques (e.g. [Rumbaugh et al., 1991]), we analyse the expectations which each component has of the components with which it is interacting.

At this stage the design may be refined so that the mutual expectations of some components are minimised. The less that is expected of a component, the less its potential to cause damage to the rest of the system. For example, suppose it is known that a client must be located on a host which has inherent low availability - perhaps because it is mobile. It may be appropriate to redesign the interaction between the client and the services it uses so that any interaction with the client is stateless. This means that the services are never in a state in which they are expecting the client to do something (e.g. commit or abort a transaction).

The next step is to identify the facilities which the engineering and underlying platform needs to provide to meet the expectations of the

application components. For example, suppose a client has an expectation it will be charged for a service only if it successfully uses it (i.e. both events occur or neither occurs). One option to satisfy this expectation could be to use atomic actions [Warne and Rees, 1993].

The engineering model helps the system designer or programmer to choose the appropriate configuration of engineering mechanisms (see section 8). The programming model helps the programmer to write application components which have the properties required by these engineering mechanisms (see section 7).

6.1 Fault diagnosis

Fault diagnosis is the process of identifying the faulty component which is responsible for a failure. This can be difficult because the fault may have propagated from the original faulty component by causing other components to fail. If a fault is wrongly attributed to a particular component, erroneous reconfigurations are sure to follow [Schnieder, F.B., 1993]. In particular good components may be decommissioned while the faulty component is left in the system.

The relevant components in an ANSA system are the bindings between clients and interfaces. Bindings are the place where contracts take effect (between client and server), and where reconfiguration is possible. They are also the place where federation boundaries may exist. Fault diagnosis tries to isolate the fault to the particular client, interface or part of the binding from which the fault originated. Sometimes fault diagnosis may have to stop at a federation boundary: beyond that boundary diagnosis is the responsibility of another organisation.

The traditional concept of a failure focuses on service: a failure is said to occur when a service deviates from its specification [Laprie, 1992], [Siewiorek and Swarz, 1992]. In ANSA the consequences of federation and separation mean that mutual suspicion is extremely important. One should not take a client's word for it that a service has failed - it may be that the client itself has failed. The ANSA failure model [Edwards and Rees, 1994] captures this: it does not prejudge whether the faulty component is the one which engages in the event or the one which observes or expects to observe the event.

This leads to a situation which is potentially ambiguous: either the observer or the component which engaged in the event may have failed. To avoid this, the parameters used to determine correctness must be made explicit.

The notion of what is correct ideally should be captured by a formal contract between two objects. Interface definitions are an offer to form a contract between a client and server, the stronger these contracts the easier it is to avoid ambiguity. Unfortunately, usually correctness is captured only partially in an interface definition and written text.

The ANSA work on federation is investigating contracts between clients and servers [Beasley et al, 1994]. The programming model for dependability will involve investigating enhanced interface definitions (see section 7). This will allow stronger statements to be made about expected behaviour.

6.2 Failure models and hierarchies of failure modes

A failure mode describes the characteristics of a class of failures (e.g. omission failures, value failures, crash failures, fail-stop [Laprie, 1992]). There are many hierarchies of failure modes in the literature (e.g. [Barborak et al, 1993], [Shrivastava et al., 1990], [Cristian, 1990]). Such hierarchies are sometimes referred to as *failure models*. In contrast the ANSA failure model is not a failure hierarchy; it provides a set of concepts for understanding the semantics of failure.

Failure hierarchies arise from partial orders on failure modes; they are useful, because they say when one engineering mechanism can safely replace another. For example, suppose there is a partial order $\subseteq$ on failure modes, and suppose there are two failure modes x and y such that $x \subseteq y$. Then any mechanism consistent with $\subseteq$ which can detect and tolerate y will also detect and tolerate x. Suppose x is omission failures and y is value failures, whether or not a mechanism actually detects and tolerates both x and y will depend on the implementation of that mechanism. Hence it is the engineering model (which prescribes the configuration and implementation of engineering mechanisms) which will determine, $\subseteq$, the ordering on failure modes. Different engineering models will give rise to different orderings; within an engineering model different arrangements of components may produce different orderings. The ANSA engineering model for dependability is discussed further in section 8.

Failure modes are useful in the engineering of dependable systems. For example, if the failure behaviour of a server is known to be restricted to a well understood mode (e.g. fail-stop), the engineering mechanisms used by its clients need to be able to deal only with this behaviour.

The ANSA failure model can be used to describe the failure modes which are discussed in the literature (see [Edwards and Rees, 1994] for further details). During the development of the engineering model for dependability it is intended to use the ANSA failure model to analyse configurations of engineering mechanisms to determine what failure modes they can detect and tolerate.

7. The Programming Model

This section discusses a number of requirements that have been identified for the programming model. The role of the programming model is to provide the concepts to assist programmers in making sure that application components meet the expectations (requirements) of their supporting engineering components. Together with the engineering

model, it enables system designers to make trade-offs between what is provided by the engineering components and what is provided by the application components. As an example, suppose there is a choice of replication mechanisms including one that consumes fewer resources, but can only be used if the state of a service is immutable. There are a variety of possible tools and techniques that can be used to exploit this opportunity.

1. A tool to analyse the application code automatically and report whether or not it has the immutability property.
2. Guidelines and rules for the programmer which explain how to write the code so that it has the appropriate characteristics.
3. A tool that transforms the code so that the most appropriate mechanism is used. This approach has already been investigated for an atomic activity infrastructure: code transformation was used to insert calls to appropriate lock mechanisms whenever mutable state was accessed [Warne and Rees, 1993].

The scope of this technology will be set in part by the programming language which is used. Some languages are less amenable to transformation and automatic analysis than others.

It is usual to perform type checking based on information provided in an interface definition. The interface definition is a very limited specification of the expected or allowed behaviour. Part of the intended work on the programming model will be to extend this technology by adding information about the expected behaviour of the interface. For example:

- whether an operation updates or observes a mutable state
- how an operation affects the outside world: does it read information about the outside world (sensor) or does it make changes to the outside world (actuator).

If type checking tools can be enhanced to check these attributes, then at least some expectations can be checked automatically.

Programmers may choose to exploit application level redundancy - redundancy which is associated with the application semantics. For example:

- knowledge which restricts the time or value of a result, e.g. time should not run backwards; this kind of redundancy could be captured by behavioural descriptions of objects
- alternative algorithms for achieving a particular aim (this is sometimes called resourcefulness [Abbott, 1990], recovery blocks is one example which exploits this idea [Randell, 1975])
- the use of stability and self stabilisation [Schneider, M., 1993].

At present there are no plans to investigate application level redundancy; redundancy is provided by the engineering model.

8. The Engineering Model

This section discusses some of the requirements which have been identified for the engineering model. The engineering mechanisms are positioned between the underlying platform and the application components as shown in figure 3. Failures can occur in the application components, the underlying platform or the engineering mechanisms themselves.

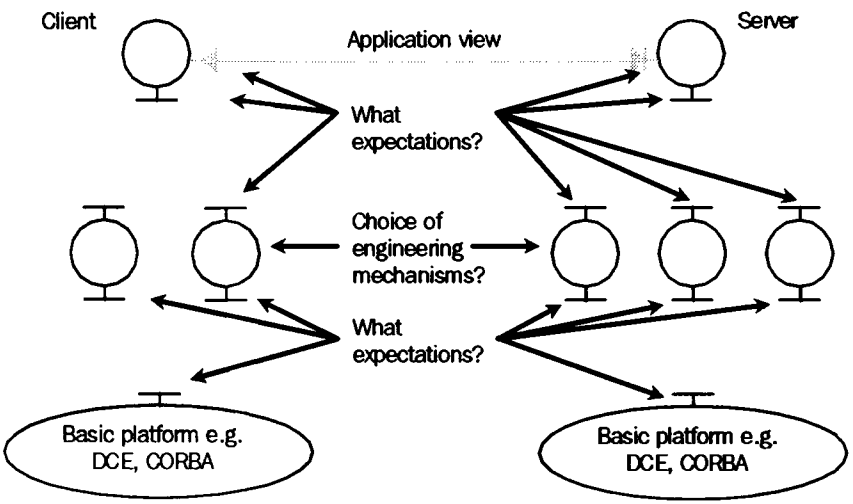


Figure 3 The relationship between the programming and engineering models

As shown in figure 3 the application components and engineering will have requirements (expectations of each other). The role of the engineering model is to help the designer select a configuration of engineering mechanisms which ensure that the expectations of the application components are satisfied even when failures occur. The designer also uses the programming model to make trade-offs between what is provided by the engineering and what is provided by the application components. The engineering mechanisms enhance the functionality of the basic platform so that it meets the requirements for dependability. The engineering model also needs to ensure that the expectations of the engineering mechanisms and underlying basic platform are matched.

The engineering model will consist of a set of basic services useful for building dependable systems, for example: reliable multicast state transfer mechanisms, audit services, persistent storage services.

Each service will have a typed interface (just like any service which is visible to the application). This will allow type checking to be applied to engineering objects (as well as application objects). Currently, checking tends to be limited to application level objects, because the configuration of supporting engineering objects tends to be rather static and uniform. Well understood, standard configurations do not require constant checking and validation. In the future the aim is to allow dynamic configuration of engineering objects specifically to match client and server requirements; thus each new configuration will require checking.

The engineering model will prescribe standard configurations of mechanisms which will have well understood behaviours (and expectations). These configurations must themselves be dependable. It is intended to use the concepts in the failure model and the methodology outlined in section 6 in their development. Examples of such configurations could be:

- a configuration of mechanisms to ensure fail-stop behaviour
- replication for a non-mutable service
- replication for a mutable service.

Initially the engineering model is likely to consist of a set of mechanisms and a set of standard configurations for those mechanisms. However, structuring the mechanisms as services will allow recombination of these mechanisms in application-specific ways. Application-specific configuring of the engineering mechanisms is likely to be a complex and error prone task. Unless proper support is provided to help programmers select the right configuration of mechanisms they are likely to be tempted to ignore what is provided and build their own. This suggests that eventually it will be necessary to provide tool support for configuration.

The opportunities to build completely new systems are becoming fewer and fewer. In general, new applications and systems will have to interwork with what already exists. If a new application is to be dependable, the dependability of the existing services with which it interworks needs to be evaluated. If they do not provide sufficient dependability they need to be enhanced in some way, or at the very least the new application needs to be protected from them. The engineering model needs to provide mechanisms and configurations of mechanisms to do this.

Much of the engineering model is concerned with the provision of redundancy to give fault tolerance. Redundancy can be provided in the form of extra storage, processing or communications; it can also be provided in space or time (e.g. doing the same thing twice simultaneously or sequentially). Examples include:

- Replication [Oskiewicz and Edwards, 1993]
- Checkpointing [Birrell et al., 1987]

Comprehensive lists of redundancy technology are given in [Siewiorek and Swarz, 1992] and [Smethurst and Wharton, 1993].

The engineering model will also provide some mechanisms which enforce requirements; these can be regarded as fault avoidance mechanisms. An example of such a mechanism is a protocol which enforces ordering between messages to ensure a group of servers see messages in the same order. (Note that the above mechanism may itself be part of a replication protocol which is intended for fault tolerance.)

The engineering model provides a set of mechanisms to supplement and support application redundancy. The engineering model is not only concerned with the provision of redundancy, it is also concerned with the management of redundancy; the latter is considered separately in section 9.

9. The Management Model

This section discusses some of the requirements which have been identified for the management model.

The part of the engineering model discussed in section 8 is concerned mostly with mechanisms which provide redundancy. The management model includes the part of the engineering model which is concerned with how to manage this redundancy to tolerate faults, how to maintain dependability, how to install new applications and how to upgrade existing ones. For example, consider an active replica group; managing the group involves providing mechanisms which can detect failures, reconfiguring the group to remove the faulty member, and adding new members to maintain the level of dependability when existing ones fail. A redundant system may go through as many as eight stages when a failure occurs [Siewiorek and Swarz, 1992].

1. *Fault confinement* is concerned with limiting propagation of a fault to other parts of the system. For example, this can be achieved by liberal use of detection mechanisms to try and detect a fault as soon as possible.
2. *Fault detection* is measuring value and time and comparing what is observed to what is expected.
3. *Fault diagnosis* is used if fault detection does not identify the faulty component. Fault diagnosis is discussed in section 6.1.
4. *Reconfiguration* takes place once the faulty component has been identified. The aim is either to isolate the system from the faulty component or to replace it with a spare.

5. *Recovery* attempts to remove the effect of the fault. Redundant information can be used to correct the erroneous state (space redundancy). Alternatively the system can roll (backwards or forwards) and either retry or try an alternative strategy (time redundancy).
6. *Restart* takes place once all the damaged state has been removed. In extreme cases large parts of the system may need to be restarted from its initial state.
7. *Repair* restores the faulty component to an undamaged state. Redundancy might be used to correct erroneous state.
8. *Reintegration* involves reconfiguring the system to introduce the repaired component.

Management will usually be embedded into the redundancy mechanisms which they manage. However, it is often necessary and convenient to have separate management and service interfaces for the redundancy mechanisms e.g. [Oskiewicz and Edwards, 1993], even if the interfaces are onto the same object.

Just like all the engineering mechanisms, the management mechanisms themselves need to be dependable - system recovery mechanisms can be responsible for 35% of system failures [Toy, 1992].

10. Extended Transactions Framework (ETF)

This section summarises the work to date on the extended transaction framework (ETF) and possible future directions. Further details are reported in [Warne, 1994].

Transactions exploit both fault avoidance and fault tolerance. For example, computations enclosed within traditional transactions have well defined relationships with each other: they are constrained by the fundamental properties of atomicity, consistency, isolation, and durability (collectively known as the ACID properties [Bernstein et al., 1987]). This helps to avoid faults which can be introduced by concurrent interfering computations.

Transactions use redundancy to undo their effects should they need to abort (fault tolerance). For example, the Tandem transaction processing monitor (Pathway) distributes work to available processors. Should any of this work be lost or compromised by failure it is automatically restarted after being rolled back to its initial state [Bartlett et al. 1992].

The traditional transaction model, with its strict ACID properties, is highly effective in some application areas such as conventional databases. However, it is frequently found lacking in functionality, flexibility, and performance when used in other applications areas, especially those involving collaborative or long-lived activities. Such applications typically require some, but not all, of the ACID properties. This has led

to the development of many different kinds of transaction models, for example: Split Transactions [Pu et al., 1988], Coloured Transactions [Shrivastava and Wheater, 1990] and Transaction Groups [Skarra, 1989].

As observed in [Chrysanthis and Ramamritham, 1990], irrespective of how successful these extended transaction models are in supporting their intended application domains, they merely represent points within the spectrum of interactions possible within competitive and cooperative environments. Therefore, they each capture only a subset of the interactions to be found in any complex information system.

ETF is intended to address these concerns; it is intended to support a wide range of telecommunications and other business applications. Inevitably, different classes of applications will require different transaction models. These models will have differing concurrency control methods; differing recovery procedures; differing resource placement, migration and replication strategies; and differing guarantees of timeliness (execution responsiveness). ETF must enable such application diversity to interoperate effectively.

ETF identifies a number of new primitives for controlling the behaviour of different transaction models. The inspiration for these primitives stems from the abstract concepts of the ACTA meta-model [Chrysanthis and Ramamritham, 1992]. The transaction model is characterised by controlling four basic attributes of a transaction. ETF provides primitives to control these attributes: Visibility, Permanence, Correctness and Recovery (VPCR).

- *Visibility*: the degree with which members of the extended transaction are able to observe each other's effects before the transaction as a whole terminates its execution and commits or aborts.
- *Permanence*: the rules by which members are allowed to record their results in the stable state of the system.
- *Correctness*: the acceptable effects on system state that members are permitted to produce.
- *Recoverability*: the capability of an extended transaction as a whole, or its members in part, in the event of failure, to recover and take the system to some state that is considered correct.

Hence ETF allows programmers and designers to construct transaction models which match the requirements of the application object (providing a service). It is intended to use the concept of transaction-based work flows (e.g. [Dayal et al., 1990], [Sheth, 1993]) to describe how different application objects supporting different transactions models fit together. A workflow will consist of several related transactions which interwork to achieve a specific goal. Each transaction in a work flow may be different when characterised in terms of VPCR.

From figure 1, it can be seen that ETF spans both the programming and engineering models. It is intended to build engineering mechanisms for ETF which support the control of the VPCR attributes. Additional mechanisms will be needed to support the concept of work flows. The programming model will contain the concepts needed to drive the engineering mechanisms supporting ETF.

11. Summary and Conclusions

Several de-jure and de-facto standards are emerging which will provide the technology to make open distributed computing possible. However, for open distributed computing to be exploited in business-critical applications, there is a need to show how this technology can be extended to enable services to be delivered with appropriate and defined dependability guarantees.

No one set of dependability requirements is appropriate to all applications. Rather the dependability requirements will be determined by each application. The concept of selective transparency can be used to select an appropriate set of engineering mechanisms and configure those mechanisms to satisfy a particular application's requirements. This selection needs to be automated.

The ANSA principles reveal a number of issues for dependability in open distributed systems.

- Objects are responsible for their own dependability; contracts must be used in a federated environment to state what each object is entitled to expect of others (see sections 3.4 and 6.1)
- Technologies based on a notion of a global observer or global clocks are not appropriate: they cannot be realised in large distributed systems (see sections 3.3 and 6)
- Faults should be detected as early as possible, ideally before a component is installed into the system (see section 3.7)

Designers of dependable systems should take an end-to-end view (see section 4): careful design and partitioning of functionality can reduce the need for complicated and sophisticated engineering mechanisms to support an application.

The ANSA work on dependability aims to develop the technology for building open dependable distributed systems. It is anticipated that such systems will be built using industry standards such as DCE and CORBA platforms.

A failure model has been developed and its use in the design of dependable systems is being investigated (see section 6). One of the roles of the designer is to identify the requirements of the application components in terms of what each component expects from the underlying engineering. The notion of expectations in the failure model

can be used for this. The engineering model then helps to identify a suitable configuration of mechanisms to meet these expectations (see section 8). The engineering components enhance the functionality of the basic platform so that it can meet the application's requirements for dependability.

The designer also needs to state what constraints the application components must meet, i.e. what the engineering expects of the application components. Again the notion of expectations can be used for this. The programming model helps the programmer to build application components which satisfy these constraints (see section 7).

Taken together, the programming and engineering models enable system designers to make trade-offs between what dependability is provided by the engineering mechanisms and what is provided by the application components.

As part of the work on programming and engineering models, an extended transaction framework is being developed (see section 10): we believe transactions are a fundamental technology in the building of dependable systems.

12. Acknowledgements

The author would like to acknowledge the contribution of his colleagues in the ANSA team to this work: Ed Oskiewicz, seconded to the ANSA team by BT; Owen Rees of APM Ltd.; John Warne, seconded to the ANSA team by BNR. In addition, comments and discussions with the following on various aspects of the work reported here were most helpful: Jane Cameron, seconded to the ANSA team by Bellcore, Brian Coan of Bellcore, Gray Girling of APM Ltd., Andrew Herbert of APM Ltd., Dave Otway of APM Ltd., Santosh Shrivastava of The University of Newcastle and Paul Vickers of Hewlett-Packard. Finally, thanks to the anonymous ICL reviewers who also helped to improve this paper.

13. References

ABBOTT, R.J., "Resourceful Systems for Fault-Tolerance, Reliability and Safety", ACM Computing Surveys, 22(1), p35-68, March 1990.

BARBORAK, M., MALEK, M., DAHBURA, A., "The Consensus Problem in Fault-Tolerant Computing", ACM Computing Surveys, 25(2), p171-220, June 1993.

BARTLETT, J., BARTLETT, W., CARR, R., GARCIA, D., GRAY, J., HORST, R., JARDINE, R., JEWETT, D., LENOSKI, D., MCGUIRE, D., "Fault Tolerance in Tandem Computer Systems", p586-648 in [Siewiorek and Swarz, 1992].

BEASLEY, M., THOMAS, G., CAMERON, J., HOFFNER, Y., VAN DER LINDEN, R., "Establishing Co-operation in Federated Systems", *ICL Tech. J.*.Vol. 9 Iss. 2 pp. 195-217.

BERNSTEIN, P.A., HADZILACOS, V., GOODMAN, N., "Concurrency Control and Recovery in Database Systems", Addison-Wesley Publishing Company Inc., 1987.

BIRRELL, A.D., JONES, M.B., WOBBER, E.P.,, "A Simple and Efficient Implementation for Small Databases", in Proc 11th ACM Symp on OS Principles, ACM OS Review, 21(5), p149-154, 1987.

CHRYSANTHIS, P.K., RAMAMRITHAM, K., "ACTA: The Saga Continues", Database Transaction Models for Advanced Applications, Edited by Ahmed K. Elmargarmid, Morgan Kaufmann Publishers, 1992.

CHRYSANTHIS, P. K., RAMAMRITHAM, K., "ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior", Proceeding of the ACM SIGMOD International Conference on the Management of Data, 1990.

CRISTIAN, F., "Understanding Fault-Tolerant Distributed Systems", IBM Research Report, RJ 6980 (66517), Almaden Research Center, California, USA, 1990.

DAYAL, U., HSU, M., LADIN R., "Organizing Long-Running Activities and Triggers and Transactions", ACM SIGMOD Proceedings, 1990.

EDWARDS, N.J., REES, R.T.O., "A Model for Failures in Dependable Systems", APM.1143, APM Ltd., Cambridge, U.K., 1994.

HARRIS, R.J., FRASER, R.J.C., "Command and Control Infrastructures: The need for Open System Solutions", Keynote Address, IEE International Workshop on Systems Engineering for Real Time Applications, 13 -14 September 1993.

HERBERT, A.J., "Open Distributed Processing - the Solution to a Business Need", APM.1055, APM Ltd., Cambridge U.K., 1993.

ICL, "DAIS: System Overview", ICL manual R30428/03, December 1993.

ISO, "OSI Distributed Transaction Processing" (OSI TP), ISO/IEC 10026.

ISO, "Guidelines for the Definition of Managed Objects", ISO/IEC 10165 Part 4.

ISO, "Basic Reference Model of Open Distributed Processing", ITU-TS Recs. X.902, X.903, ISO/ IEC Draft International Standards 10746-2 and 10746-3.

ISO, "Open Systems Interconnection - Remote Procedure Call", ISO/IEC 11578 (draft).

KAYE, J., "Supply and demand", (Figures quoted and attributed to the Gartner group), Informatics, September 1993.

LAPRIE, J.C. (ed.), "Dependability: Basic Concepts and Terminology", Springer-Verlag, 1992.

VAN DER LINDEN, R., "An Overview of ANSA", AR.000.00, APM Ltd., Cambridge U.K., May 1993.

OMG, "The Common Object Request Broker: Architecture and Specification", Document Number 91.8.1, August 1991.

OSF, "Introduction to OSF DCE", December 1991.

OSKIEWICZ, E.O., EDWARDS, N.J., "A Model for Interface Groups", AR.002.01, APM Ltd., Cambridge U.K., February 1993.

PU, C., KAISER, G., HUTCHINSON, N., "Split Transactions for Open-Ended Activities", IEEE Proceedings of the 14th Conference on VLDB, 1988.

RANDELL, B., "System Structure for Software Fault Tolerance", IEEE Trans. on Software Engineering, SE-1(2), p220-232, June 1975.

RUMBAUGH, J, BLAHA, M., PREMERLANI, W., EDDY, F., LORENSEN, W., "Object-Oriented Modeling and Design", Prentice-Hall International, 1991.

SALTZER, J.H., REED, D.P., CLARK, D.D., "End-To-End Arguments in System Design", in Proc. 2nd International Conference on Distributed Systems, Paris, France, p509-512, 8-10th April, 1981.

SCHNIEDER, F.B., "What Good are Models and What Models are Good?" in Distributed Systems, Second Edition, Mullender, S., (ed), Addison-Wesley, 1993.

SCHNIEDER, M., "Self-Stabilization", ACM Computing Surveys, 25(1), p45-67, March 1993.

SHETH, A., RUSINKIEWICZ, M., "On Transaction Workflows", Data Engineering Bulletin, June 1993.

SHRIVASTAVA, S.K. , WHEATER, S.M., "Implementing Fault-Tolerant Distributed Applications Using Objects and Multi-Coloured Actions", in Proc. 10th International Conference on Distributed Systems, Paris, France, 28th May-June 1st, 1990.

SHRIVASTAVA, S.K., EZHILCHELVAN, P., LITTLE, M., "Understanding Component Failures and Replication in Distributed Systems", ISA Project Report: UNT/TR1, University of Newcastle May 1990.

SHERMAN, M., "Distributed Transaction Processing in a DCE Environment with Encina", Tutorial presented at 13th International Conference on Distributed Computing Systems, Pittsburgh, USA, May 1993.

SIEWIOREK, D.P., SWARZ, R.S., "Reliable Computer Systems - design and evaluation", Second Edition, Digital Press, 1992.

SKARRA, A. H., "Concurrency control for cooperating transactions in an object-oriented database", SIGPLAN Notices, 24(4), April 1989.

SMETHURST, R.,WHARTON, P., "OPENframework Availability", Prentice-Hall 1993.

TOY, W.N., "Fault-Tolerant Design of AT&T Telephone Switching System Processors", p533-574 in [Siewiorek and Swarz, 1992].

UI, "UI ATLAS Distributed Computing Architecture: A Technical Overview", Unix International.

WARNE, J., "Flexible Transaction Framework for Dependable Workflows", APM.1263.02, APM, Ltd., Cambridge, U.K., June 1994.

WARNE, J.P, REES, R.T.O, "ANSA Atomic Activity Model and Infrastructure", AR.004.01, APM, Ltd., Cambridge, U.K., January 1993.14.

14. Biography

Nigel Edwards

Nigel Edwards received a BSc (1st class Hons.) in Electrical and Electronic Engineering and a PhD in Dynamically Reconfigurable Distributed Systems from The University of Bristol in 1985 and 1989 respectively. He has been with Hewlett Packard Laboratories since March 1988 working on various aspects of distributed computing. Edwards has represented Hewlett Packard on the ANSA team since February 1992.

An Open Architecture for Real-Time Processing

Guangxing Li, Dave Otway

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

Abstract

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

1. Motivation and Benefits

Open Distributed Processing (ODP) is concerned with the use of commodity technology to build integrating applications that link together existing applications, databases, control systems and users.

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

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

- *general purpose distributed computing environments are evolving towards real-time systems.*

  For example, the advances in digital communication networks and in personal computer workstations are beginning to allow the generation, communication and presentation of real-time voice and video media simultaneously. Many non-real-time systems have been disembowelled to extend their use to real-time processing [Leung, 90]. This trend requires distribution and real-time control functionality to be intrinsic elements of the system. There is a great demand to provide real-time functionality as normal system services, rather than as special add on features.

- *real-time applications are evolving towards large distributed systems.*

  One-million-line real-time software systems in telecomms, manufacturing, transportation and other application areas are becoming common today [Gopinath, 93]. Such systems are large by any standard and inevitably distributed. Therefore, in addition to the problems associated with real-time operation, such applications are subject to all of the problems of any large software system, such as maintainability and distribution. Furthermore, in many real-time applications, tight real-time constraints may apply to only part of the whole system. For example, it is estimated that only 10 to 30 percent of a typical vehicle control software system is directly related to actual real-time control of the vehicle. There is an increasing need to adopt an open and architectural approach so that real-time software engineering can be augmented with other techniques to address evolution, scale, distribution and other issues.

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

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

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

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

This work will offer several benefits:

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

2. Desirable Features for Real-Time Systems

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

2.1 Predictability

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

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

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

2.2 Programmer Control

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

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

2.3 Timeliness

The correctness of a real-time system depends not only on the functional behaviour of the system, but also on its temporal behaviour. A real-time system environment must provide mechanisms which take these time related issues into account and must help application programs to meet these timing constraints. A simple example is to allow an application to associate deadlines with real-time activities, and to have the system employ a deadline based scheduling policy to ensure deadlines are met or to cancel obsolete operations when missed deadlines are identified. Other required functions include the description and enforcement of temporal relations among related computational activities.

2.4 Mission Orientation

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

2.5 Performance

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

3. Technologies

This section is structured as follows:

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

3.1 Contributory Technologies

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

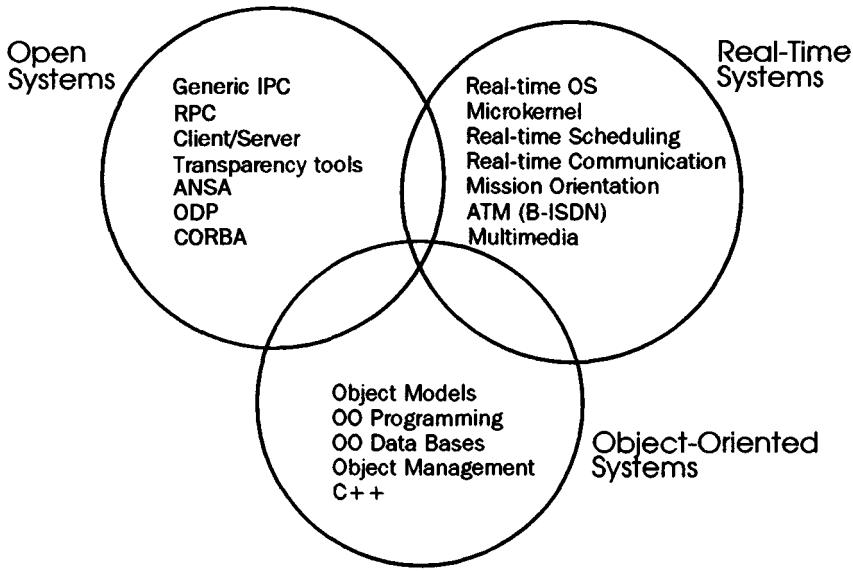


Figure 1 Contributory Technologies

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

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

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

3.2 Distributed System Environments

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

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

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

3.3 Real-Time Distributed System Environments

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

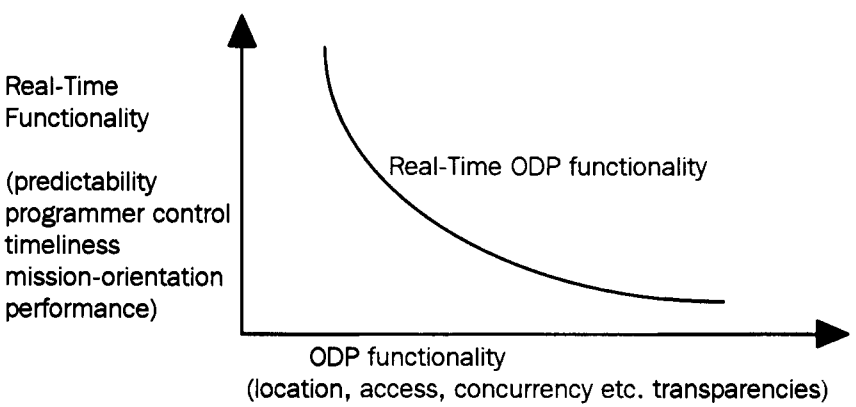


Figure 2 Real-Time ODP Functionality

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

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

4. Target

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

The scope of this research for real-time applications is supervisory control [Northcutt, 87] as opposed to low-level, synchronous sampled data loop functions like sensor/actuator feedback control, signal processing, priority interrupt processing and so on.

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

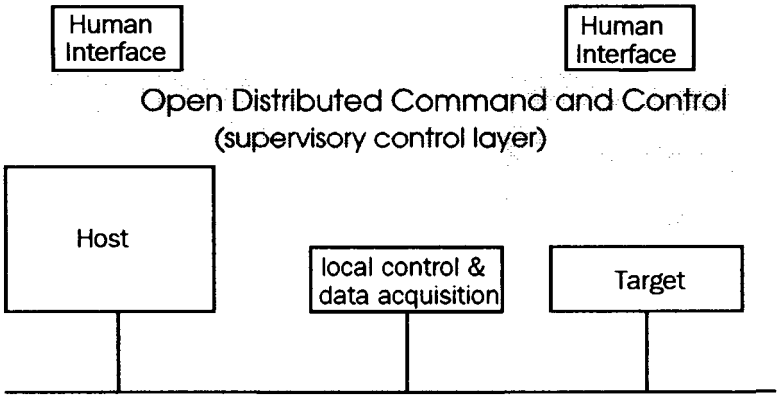


Figure 3 Supervisory Control

5. Designing for Real-Time and Multimedia Processing

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

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

5.1 ANSA Objects

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

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

5.2 Streams

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

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

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

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

5.3 Signals

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

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

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

5.4 Synchronous Computing

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

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

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

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

5.5 Explicit Binding

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

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

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

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

5.6 QoS Framework

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

A QoS statement is a generic mechanism which can express:

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

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

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

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

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

5.7 Real-Time Programming Model

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

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

The real-time ANSA architecture defines a real-time programming model to provide a predictable object execution model and to extend the traditional real-time computing concepts to distributed processing. It extends the traditional ANSA engineering model to include various priority-based and deadline-based object execution models. The programming model incorporates tasks and communication channels (the

two most important resources in real-time distributed computing) as its basic programming components. It synthesises aspects of resource requirements, resource allocation and resource scheduling into an object-based programming paradigm.

6. Real-Time ANSAware

ANSAware is an implementation of the ANSA computational model and an example of the ANSA engineering model. ANSAware is a suite of software for building ODP systems, providing a basic platform and software development support in the form of program generators and system management applications. ANSAware provides a uniform view of a multi-vendor world, allowing system builders to link together distributed components into network-wide applications.

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

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

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

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

- extended tasking system
    - o *entry*: a new abstraction which represents a scheduling point. Separate scheduling points can be used for the separation of concerns of different scheduling requirements
    - o *real-time scheduling*: preemptive priority-based scheduling
    - o *multiple scheduling policies*: the co-existing of real-time and non-real-time scheduling support

- *real-time tasks*: full real-time p-thread functions.
- *stack-based rescheduling system*: a thread may use its task (stack) resource to execute another thread to facilitate dynamic real-time scheduling
- *real-time threads*: a thread may be associated with a priority and/or deadline
- multiple thread scheduling policies based on policy/mechanism separation.

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

7. Summary

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

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

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

8. Related Work

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

9. Acknowledgements

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

10. References

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

BOUSSINOT, F. and SIMMONE R. "The ESTEREL Language", *Proc. of the IEEE, Vol. 79, No. 9*, September 1991.

COULSON G. et al. "Extensions to ANSA for Multimedia Computing", Computer Networks and ISDN Systems, Vol. 25, pp 305 - 323, 1992.

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

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

HERBERT, A.J. "An ANSA Overview". *IEEE Network*, pp18-23, January 1994.

HERBERT A. J. "The Challenge of ODP", Document APM 1016, *Architecture Projects Management Ltd., Cambridge U.K.*, February 1993, Also Appeared as an Invited Paper for the Berlin ODP Conference, October 1991.

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

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

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

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, O. "The ANSA Computational Model". Document APM.1001, *Architecture Projects Management Ltd.*, Cambridge U.K., February 1993.

11. Biographies

Guangxing Li

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

Dave Otway

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

12. Epilogue

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

The ANSA project has a World Wide Web server (URL is http://www.ansa.co.uk). Documents describing the results can be obtained through an FTP server (ftp.ansa.co.uk). The FTP server is not publicly accessible. Each sponsor has its own password. The contact person within ICL is John Brenner@bra0511.

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

Updating the Secure Office System

John A Jones

Business Development, Enterprise Engineering, ICL Enterprises, UK

Abstract

This paper describes the application of a single sign-on facility and a graphical user interface to a secure office system embodying multi-class security measures. The system structure is briefly described, along with the revised user interactions with the system.

1. Introduction

A description of the Secure Office System has been presented in an earlier paper in this journal [Moore, 1991]; that paper described the first version to be provided to a significant number of users. The Secure Office System is subject to continuous improvement, and is gradually being equipped to meet the needs of client-server computing. Two particular developments since the original paper are worthy of note, and are briefly described in the current paper. They are the provision of a single sign-on facility, for user access anywhere across the enterprise network, and the introduction of a graphical user interface. The following description simplifies some details of the interactions, in the interests of an easier understanding for the non-specialised reader.

1.1 Background

Before going on to look at the new developments, it is worth summarising the characteristics of the original Secure Office System. (A more detailed treatment is available in [Moore, 1991].)

The Secure Office System provides office and other services to several thousand users, working in organisations where security of information can be of considerable importance. The services are provided by servers running a version of the UNIX operating system with security enhancements, and the majority of the user functions are delivered by database and OfficePower-style applications. (OfficePower is an integrated office automation suite marketed by ICL.) All applications use a character-style user interface, and users access the system through

character-mode terminals, which also support a limited graphical capability for the output of charts in graphical form.

Each user of the Secure Office System has a unique identity (username) by which he is known to the system; he also has a uniquely coded badge. To log into the system he inserts his badge into the terminal, and enters his username and password. If all the credentials submitted are acceptable, and if the user is registered for use of the particular terminal at that time of day, the login is successful, and the user session now starts. The session may comprise one or more subsessions, in each of which the user assumes one of his pre-assigned roles, e.g. Duty Security Officer, Personnel Officer (Dept. E), or Director (XYZ establishment). All significant actions taken by the user are subject to audit, recorded against him as an individual (not against the role), so that the responsibility for any security incident can be tracked. In certain cases, an immediate security alert will be transmitted to the security administrator, so that suitable action can be taken.

The system maintains a regime of Mandatory Access Control, to ensure that sensitive information cannot become available to unauthorised roles. Every object in the system, at the time of its creation, is assigned a particular security classification, or Label; this is expressed in terms of a hierarchical set of sensitivity levels, combined with other more specific qualifications, relating, for example, to particular projects. Similarly, every role is configured with the ability to operate at one or more specific Labels. The system ensures that a role, operating with a particular Label, cannot read any information with a more sensitive Label, and cannot write to any object with a less sensitive Label. Thus, information from an object with a given Label can only ever be transferred to an object with the same or a higher Label.

The operation of this rule is made more manageable for the user by dividing his work into a number of distinct *subsessions* within his overall session, each with its own Security Label. The current subsession Label is displayed in a protected part of the display screen, as also is the Label of the current target object. This makes it easier for the user to appreciate which objects are accessible for reading or for writing in any particular subsession.

The user can also make use of Discretionary Access Control: any of the user's objects can have an Access Control List assigned to it by the user; this specifies the particular people who are permitted to have access to the object. Access to an object is dependent on satisfying both the Discretionary and the Mandatory Access Controls; they are not alternatives.

Finally, the system provides a secure Trusted Path facility: a means by which the user can be sure that he is in communication with the trusted computing base, and not with an unknown or *spoofing* application of uncertain reliability. This is necessary for certain critical operations, where security might otherwise be compromised: for example, the trusted

path is always in operation at the point of logging in, and can be invoked by explicit request at other times, particularly for changing passwords and for establishing or selecting new subsessions. A well-known ploy of hackers on less secure systems is to run a program which appears to be displaying a login prompt on an unattended terminal. When an unsuspecting user types in his credentials, the program stores them away in a place known to the hacker, and then terminates, connecting the user to the standard login program. The trusted path indication on the screen assures the user that this form of masquerade cannot be achieved: the dialogue must be genuine, because the secure operating system has ensured that the indication cannot be counterfeited by another, unprivileged, program.

1.2 The new developments

In previous phases of the Secure Office System, most users have simply logged into their local server and accomplished their work within the scope of that single machine. It has also been possible to select a remote service and log into it separately, but the process was cumbersome, and not popular with users. The aim of providing single sign-on has been to achieve a closer integration, so that the user can simply select the required services from a menu, wherever they may be located, and be connected without further ado; in principle the user should not need to know anything about the distribution of services.

The original Secure Office System provided a dumb terminal interface, in keeping with the standard office system of the 1980s and early 1990s. As the industry's focus for application development has moved ever more onto the desktop, so it has become appropriate to provide a platform to allow this trend to be exploited by the Secure Office System.

The remainder of this paper shows how these new aspects of the infrastructure have been provided.

2. Single sign-on

2.1 Overview

A number of different single sign-on products are now available in the marketplace. Two particular schemes have been exerting the greatest influence on the standards organisations, and hence on the commercial exploitations of the technique. These are Kerberos (from MIT; also used in DCE) and the ECMA Technical Report 46 (TR46) scheme (as represented by the Sesame project and ICL's Access Manager). A comparative review of the available products is provided in [Parker, 1994]. The motivation in all cases is to provide the user with the ability to gain access to a number of different services, which may be physically dispersed, without needing to go through an authentication process at each individual service: once the user has been authenticated at the initial point of access, suitable credentials can be passed to other services on his behalf. Clearly this simplifies matters for the user, who now needs only to remember one set of authentication data. A security benefit flows from

this, because the user is now less likely to need to write down the necessary passwords, and also less likely to use trivial or unsuitable ones.

Early implementations of single sign-on do little more than to automate the login and password submission that the user would otherwise have to undertake himself. This approach has the disadvantage of course, that, as with the previous system, the dialogue is carried on in clear text over the interconnecting network, with all the attendant risks of interception, replay, masquerade, and so on.

More advanced implementations involve explicit cooperation between the authorities responsible for the various services: they agree on a set of authentication data that can be passed between them, conveying relevant security information about the user and the facilities to be made available. This process implies a need for trust between the authorities, and in the mechanisms used for the information interchange, and this usually necessitates the use of cryptography. The theoretical basis for the interrelationships is covered in [Rogers, 1994] later in this issue.

2.2 Implementation

The single sign-on scheme used in the Secure Office System follows the basic principles of ECMA Technical Report 46, (TR46) but adds some refinements, and borrows heavily from the ICL Enterprises developments for the Access Manager product. The main components of the scheme are as follows:

- Combined Authentication and Privilege Attribute Service
- Name Service
- Subject Sponsor
- Association Management
- Context management.

The component interrelationships are depicted in Figure 1. Their functions and purpose are described below.

2.2.1 Combined Authentication and Privilege Attribute Service

The Combined Authentication and Privilege Attribute (CAPA) Service holds details of the users of the system: their identities, their current passwords (encrypted), their badge codes, and the locations where they are permitted to log onto the system. It also holds information relating to the security clearances of the users: the range of Labels they are permitted to use for their subsessions and the roles they are permitted to assume. In addition the CAPA service keeps details of the security clearance for user access points (terminals and workstations). The static part of this information (apart from user passwords) is provided by, and can only be amended by, authorised administration personnel. Dynamic status information (e.g. history of the current session) is managed by the CAPA service itself.

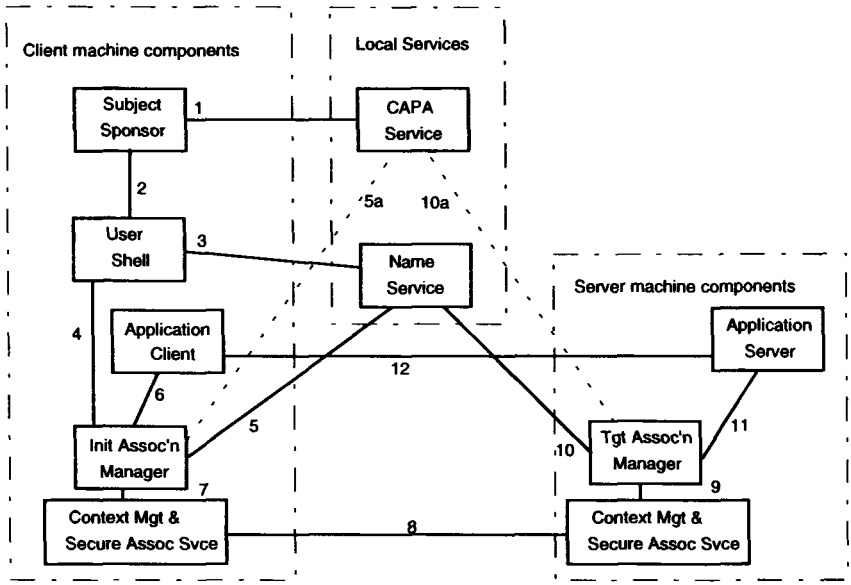


Figure 1. Single sign-on system structure

The CAPA Service is responsible for the issue of Privilege Attribute Certificates (PACs), which denote the privileges and limitations applicable to user processes.

The CAPA Service is represented physically by a number of server machines; the work is split between them according to performance criteria and to fit in with the needs of the administrative organisation.

2.2.2 Name Service

ICL has introduced a refinement to the basic architecture: the Name Service. The Name Service is an information repository, providing mappings from each rolename to a set of service attributes, and from each service name to an address to be used in accessing it. The role information is, in fact, organised according to roletypes. The concept of role was introduced above; in the Name Service it would be superfluous to include specific information for every individual role, so roles are now grouped together into roletypes, each group provided with the same set of services. Thus there could be a roletype Personnel Officer, covering personnel officers from all departments: they would all use the same types of application. The final distinction between different roles within a roletype is achieved by providing a different home directory in the file system for each individual role.

For each roletype, a set of menus is stored; selection of a menu entry leads (possibly via further layers of menus) to a specific application entry. Associated with each application entry there is a set of information, which defines the way of invoking the particular application instance: the

address of the service; the name of the program that has to be executed, and any parameters that have to be passed to it. The Name Service is notionally a single service covering the whole of the user population, but, for performance reasons, separate instances, all holding the same information, are provided at each locality.

2.2.3 Subject Sponsor

The TR46 architecture recognises a component called Subject Sponsor. This looks after the user's dialogue with the single sign-on system. It accepts the user's credentials at login time, and connects him to the required services. The Secure Office System has two different implementations of this architectural component, one for users of character terminals, and the other for graphical interface users.

The character terminal version of this function runs on the server to which the terminal is connected. It is divided into a number of sub-components, in recognition of their different requirements from the point of view of security evaluation. The most important of these are called Login, Subsession Controller and User Shell. Login provides the necessary dialogue for identification and authentication of the user, and the Subsession Controller controls the creation of subsessions of suitable Security Labels as well as the switching between them. A separate instance of the User Shell is created for each subsession that the user has active; the User Shell presents the user with the application menu retrieved from the Name Service for the particular role, and subsequently permits the user to invoke and to switch between the applications available.

The graphical interface version of the Subject Sponsor function is separately implemented, and is further described below.

2.2.4 Association Management

Association Management is the function that links together the client and server components of an application. A client wishing to connect to a service makes a call on the local Initiating Association Manager (IAM) component. This obtains the necessary routing and invocation information from the Name Service, and then calls the Target Association Management (TAM) component at the destination service system. The TAM checks the credentials submitted by the IAM, and goes on to invoke the required application, again if necessary obtaining further information from the Name Service. The communication between the Association Management components makes use of a secure association channel, designed to ensure that the security attributes cannot be corrupted or misused.

2.2.5 Context management

The Context Management system is responsible for establishing, at critical points, (e.g. on subsession creation, or in conjunction with Association Management activity) that the security context of each process involved (as represented by the process attributes) is consistent

with the privilege attributes included in the relevant Privilege Attribute Certificate issued by the CAPA service.

A trusted application may request modification of the context, either because it needs a different privilege set for its own execution, or because it wishes to restrict the powers to be passed to a process that it invokes by means of Association Management.

2.3 Operation

We can now trace through the operation of a simple user session, using the character terminal version.

The user switches on the terminal and waits for the badge insertion prompt. He inserts his badge, which causes the initial sign-on prompt to be displayed, with the trusted path indication. He now types in his username and password; the Subject Sponsor submits these credentials, and the identification of the terminal, to the CAPA Service. The CAPA service checks that the intended session is permissible, and provides a list of the permitted roles and clearances, indicating which is the default. The Subject Sponsor displays this information to the user, who then selects the parameters for the initial subsession. The CAPA Service generates a corresponding Privilege Attribute Certificate (PAC) defining the applicable security context.

A subsession is now created, operating at the requested role and clearance, according to the PAC provided, and the User Shell is invoked. It obtains menu information from the Name Service, and presents it to the user. The user makes menu selections as appropriate, until he arrives at the point of selecting a particular service. The User Shell now invokes Association Management to start up the service. Association Management obtains further information from the Name Service, to discover the appropriate address and invocation parameters. In a simple case these will simply specify a process on the current processor. In more complex cases, Association Management may need to invoke a process on another processor, as outlined above. In either case an application component may in turn invoke another application component, again using Association Management and the Name Service, as a preliminary to the support of a client-server dialogue. Whenever a new association is created, the subsession PAC is transmitted over the Secure Association Service to ensure that Context Management can establish the appropriate security context for the association.

At any time the user may suspend his current interaction with the system, to invoke another service. If he intends to operate with the same role and clearance, he can use the User Shell to open up the other service. Alternatively he can suspend the current subsession, and use the Subject Sponsor to open up another subsession with different role and clearance; this will involve creation of another User Shell, which the user can then use for the selection of the required service. At any subsequent time the user can switch between any of the existing services or invoke another as

needed. Figure 2 shows the relationship between session, subsession and application.

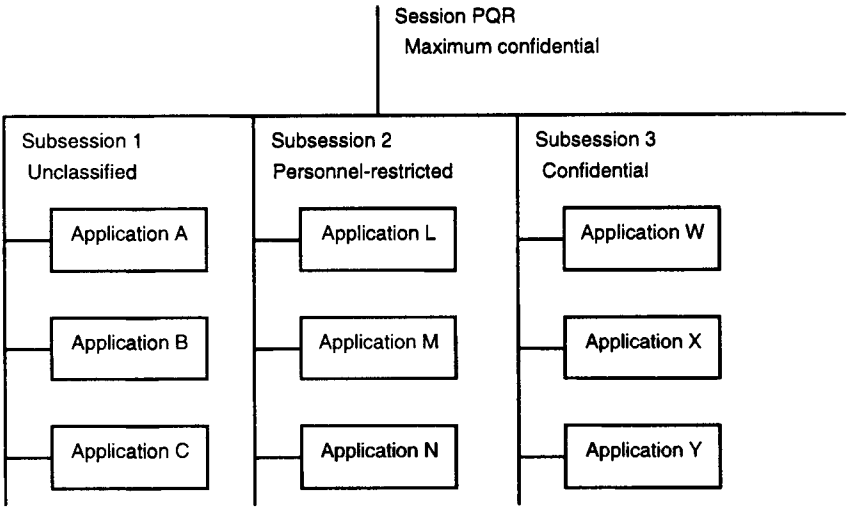


Figure 2. Session and subsession structure

3. Graphical User Interface

3.1 GUI Overview

The characteristics of a modern graphical user interface (GUI) will be familiar to most readers. What may not be so obvious is that a GUI presents difficulties for the implementors of a secure system, for a number of reasons, some of which are discussed below.

One of the significant attributes of the graphical interface is that it supports the moving or copying of material from one place to another, whether in the same or a different object. The secure system, in contrast to this, devotes a lot of effort to ensuring that improper transfers cannot be made. No commercial system at present reconciles these two requirements, and the Secure Office System includes innovative facilities to remedy the deficiency.

Another aspect concerns the freedom of the user to manipulate objects on the screen, and the interactions between such manipulations and the display of security labelling. The implications of obscuring a Label need to be resolved.

When it comes to the detailed implementation, it is necessary to guard against the possibility of *covert channels*: any means by which information is able to flow, whether intentionally or unintentionally, in contravention of the security policy. All such channels need to be assessed, and the appropriate defences put in place.

3.2 Graphical User Interface - the choices

The provision of a secure Graphical User Interface capability requires choices to be made at several levels: the style of the user interface itself; the software system that will provide the GUI, and the supporting hardware platform. The most important of these is the middle layer, the software system, because this is where the security policy is enforced.

The GUI systems in most common use are: the Microsoft Windows system, the Apple Macintosh system and the X Window system. The first two of these were soon rejected for the Secure Office System, as there was no reasonable prospect of implementing a secure version: neither the Windows system itself nor the underlying operating system would be suitable to undergo the formalities of security evaluation, and in order to derive a suitable version, a degree of collaboration with Microsoft or Apple would be required that would not be commercially practicable. The X Window system, on the other hand, has been used for some time as the basis for the secure Compartmented Mode Workstation (CMW), specified for the US defence market, and is usually built on top of a secure UNIX operating system, such as is already in use in the Secure Office System. X is then the natural choice.

The X Window system was developed initially by MIT, and the rights are now owned by the X consortium. A number of commercial implementations are available. The system uses a presentation style known as Motif, now defined in the open standards. From the user's point of view, the functional repertoire is very close to that of Microsoft Windows, as are most of the conventions, for example, how menus are structured. There are superficial differences in the shapes of the window handles and other *decorations*, but a Microsoft Windows user soon becomes accustomed to working with the Motif symbols, and, if necessary, can switch easily between the two styles.

The development team had to decide how the security requirements would affect the user interface. For the first time the users of the Secure Office System would be able to view more than one task on their screens concurrently. To meet the functional requirements, it would be necessary to arrange that these could be of differing security classifications, if required. At the same time there would have to be a clear distinction between the levels. Some consideration was given to dividing the screen into zones to segregate the different levels, but it was evident that individual areas would then become too small to be useful, and so the eventual design allows the user complete freedom in the allocation of the screen area to particular tasks. The user is reminded of the different classifications of his windows by a combination of colour-coding and explicit labelling: each visible window has a coloured border indicating the hierarchical component of the classification; and in addition to the usual title bar the window has a *security stripe*, with a label spelling out the full security classification of the task.

The remaining choice concerned the type of platform on which the system would be deployed. Options included:

- a simple monitor attached remotely to a secure terminal concentrator
- a secure form of X terminal
- various forms of intelligent workstation.

The simple monitor was rejected because it would require the development of several items of non-standard hardware; these would inevitably be expensive both to develop and to procure, because of the relatively small volumes involved. The X terminal was also rejected, because it would put excessive traffic on the network and excessive processing load on the servers, and would go against the continuing industry trend of maximising the work done at the actual desk unit.

The Compartmented Mode Workstation was investigated in depth, but was eventually found to be unsuitable. It was decided that the particular security regime provided would not meet the requirements of UK and European authorities. The most significant single problem was that it gave the user a confusing presentation of security object labelling: there were two separate labels for each window: one for the window as a whole, and one for the most sensitive information so far accessed. More recently it has emerged that the US authorities are themselves moving away from this particular approach.

The eventual solution was an adaptation of the CMW, to conform to the target security policy. This adaptation was developed by a company specialising in secure workstations. The system makes use of standard personal computer hardware, with minimal enhancements, which include a badge reader, and security sealing of the enclosure. A fairly powerful configuration is needed - at the upper end of the range appropriate to a Windows PC - to give rapid responses even when complex graphical operations are being performed, and to support the more complex and demanding requirements of the secure UNIX system. We can be confident that applications will become more demanding over time, so that even with the expected improvements in PC processing power, the secure workstation will continue to need to make use of the upper models of the available range.

3.3 Graphical User Interface - the implementation

The main components of the GUI system are shown in figure 3. They are known as:

- X Display Manager
- X Server
- X Window Manager
- X Selection Manager.

Together they form the Trusted X Service. The following sections describe how the system fits together.

3.3.1 X Display Manager

The X Display Manager is in overall control of the trusted X service. It is responsible for setting up and starting the other components of the service. In conjunction with other specialised components, not detailed here, it controls the user's login operation, and ensures the tidy closedown of the X service on completion of the session. If a new session is required, all components are restarted from scratch, so that there is no possibility of improperly gaining access to any information left over from the previous session.

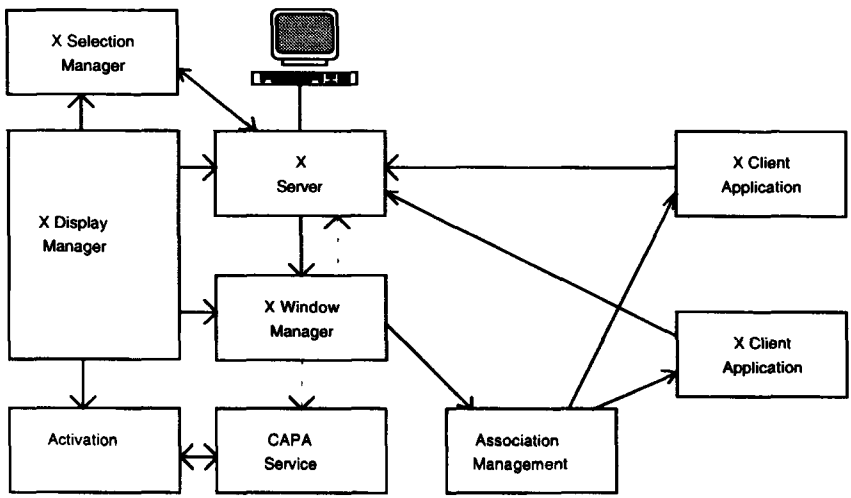


Figure 3 Trusted X Service system structure

3.3.2 X Server

At the heart of any X system implementation is the confusingly named X Server. This is the drawing engine: the component that takes instructions from applications - the X clients - and looks after the details of displaying the characters and other shapes on the display screen. Everything drawn on the screen is put there by the X Server: it receives its instructions encoded in the X11 protocol, and outputs the required patterns to the display hardware. The operations handled by the X Server itself are all low-level functions; but fortunately the application programmer is relieved of the tedium of reducing everything to this low level; he can make use of programming libraries, which support all the important aspects of the Motif style, and can be bound into the application code. In the secure implementation the X Server is the principal component for enforcement of the security policy.

3.3.3 X Window Manager

The next component to consider is the X Window Manager. This is responsible for launching applications, and for the subsequent high-level control of the screen area. This includes the positioning of the screen area to be used for particular tasks, drawing the window outlines with their *decorations* (standard buttons, etc.), keeping track of moves, resizing operations, and changes in the *stacking order* (which windows are laid over which others), and making sure that all the affected windows are redrawn by their responsible clients as and when necessary. The Window Manager does not actually draw the window contents itself; it sends messages to the relevant applications, advising them what has to be done. The X Window Manager is in fact an X Window Client, just like the applications, and it communicates with the other clients via the X Server, using a special inter-client protocol, defined as part of the Inter Client Communication Conventions Manual (ICCCM).

In the original specification of the X system, there is nothing to prevent *any* application using the ICCCM protocol to interfere with another application's display area in this way; the system relies on applications being well-behaved. This is clearly not acceptable for a secure system, and some control has to be exercised. The X Window Manager is in fact implemented as a trusted program, as is the X Server, and attempts by untrusted applications to communicate with others are simply ignored. (Any other reaction might be usable as a form of signalling by Trojan Horses.)

In addition to the standard Window Manager functions, as outlined above, the secure X Window Manager is also responsible for the security labelling of windows and icons, and for maintenance of trusted path operation and indications. The ordinary client applications are not aware of this aspect of the Window Manager's work.

3.3.4 X Selection Manager

One of the most important features of a windowing system is the ability to cut-and-paste from one window to another. As noted above, this is potentially in conflict with the security requirements, and suitable controls need to be provided. Cut-and-paste between windows is implemented in three stages: the cutting (source) application delivers the specified data to the X Selection Manager component; the Selection Manager verifies, by comparing the Labels of the two windows, that the requested transfer is permissible, and finally the pasting (destination) application receives the data and updates its display accordingly. Every transfer has to be explicitly confirmed by the user; this is to prevent a Trojan Horse program making a transfer without the user's knowledge. [A more limited cut-and-paste operation can be implemented via the X clipboard, but again subject to strict MAC rules.]

3.3.5 Graphical Interface Subject Sponsor Function

The Subject Sponsor functionality for the graphical workstation is provided by components of the Trusted X Service. The overall

functionality is the same as that provided in the character interface implementation, but the component boundaries are slightly different, because the two implementations are derived from different original designs. The X Display Manager is the overall session controller, and ensures that the trusted path is created and that the user is properly authenticated before being permitted to use the system services. The X Window Manager undertakes the functions of Subsession Controller and User Shell. Both of these trusted X components interwork with the server components (CAPA Service, Name Service, Association Manager, etc.) through a layer of interface adapting code.

3.3.6 Window labelling and Subsessions

In the original character interface system, the concept of subsessions was used to help the user keep track of the differing security contexts that he was working with. Changing from one subsession to another was made into a conscious act, so that the user would not inadvertently work with the wrong security assumptions. In the trusted X system, the necessary awareness of context is provided in a different way. Every window is provided with its individual sensitivity label and trusted path indication, so that at all times the user is aware of the relevant security context, and the switching between subsessions no longer needs to be a deliberate act; the user simply clicks on the window of interest as he requires. Thus, although the security requirements are satisfied, the overall usability is improved, and the security aspects of the system are made less obtrusive.

3.4 Application Programs for the graphical interface

The original Secure Office System used OfficePower applications to meet the user's office automation and other needs. There is now increasing pressure for access to off-the-shelf applications, keeping up with the latest trends in the interests of productivity improvement. The graphical workstation provides a suitable platform for this approach, and in due course the majority of user applications will be standard commercial ones - either UNIX/X applications or Microsoft Windows ones running through a bridging interface - and will run on the workstation directly. The first examples to be provided are graphical interface client applications built with RDBMS 4GL tools.

The new graphical applications run entirely at the workstation, in keeping with normal commercial practice. Moving applications off the server in this way will reduce the disturbing variations in performance that one user's work can currently cause in another's. The server is no longer greatly involved during the majority of the user's running of the application: there is a short burst of activity as files are opened at the start of the run, and another when the datafiles are written back at the end.

For a transitional period, though, there will be a continuing need to run some OfficePower and other applications on the local server. To meet this requirement, a specialised terminal emulator program, OfficePower X-Window Secure Emulator, will be supplied. This will run on the workstation, and communicate with the application running on the

server. The communication protocol is an extension of the standard protocol used by OfficePower for character terminals, and is supported by revised terminal handling library code at the server; the application is not rewritten, but simply relinked with the new library, and a Motif-style presentation results. The user can have one or more OfficePower subsessions open, each appearing in its own Motif-style window on the display screen and supported by its own copy of the terminal emulator program, and all the facilities of the character-interfaced OfficePower suite are available.

4. Conclusion

The developments outlined above form a foundation on which client-server working can be built up. The next stages will be the provision of further integration facilities for off-the-shelf applications, particularly for mailing and printing, and the gradual introduction of a wide range of commercial applications. Later, there will be advances in the infrastructure to meet the needs of object oriented applications, and the achievement of full location transparency across the entire network.

These developments show that with careful design it is practical in a secure environment to exploit a system (the X Window system) originally designed with a very different environment in mind. The same principles will need to be applied in subsequent developments, as the Secure Office System is extended to permit the use of an increasing range of commercial applications and facilities.

5. Glossary

Common industry abbreviations used:

DCE	Distributed Computing Environment
ECMA	European Computer Manufacturers' Association
GUI	Graphical User Interface
MIT	Massachusetts Institute of Technology
RDBMS	Relational Database Management System
4GL	4th Generation Language

Other terms and abbreviations used (with reference to section in which introduced)

CAPA	Combined Authentication and Privilege Attribute (Service) [see para 2.2]
CMW	Compartmented Mode Workstation [see para. 3.2]
DAC	Discretionary Access Control [see para. 1.1]
IAM	Initiating Association Manager [see para. 2.2.4]

ICCCM	Inter Client Communications Convention Manual [see para. 3.3.3]	
MAC	Mandatory Access Control	[see para. 1.1]
PAC	Privilege Attribute Certificate	[see para. 2.3]
TAM	Target Association Manager	[see para. 2.2.4]
TP	Trusted Path [see para. 1.1]	
OfficePower	An office application suite, marketed by ICL. [see para. 1.1]	

6. Acknowledgements

The developments described above have been undertaken by the technical staff of Enterprise Engineering, whose help the author wishes to acknowledge. Particular thanks are due to Dave Rogers and Bill Marcham, for their advice and assistance in the preparation of this paper.

7. References

MOORE, BRIAN, "Making a Secure Office System", *ICL Tech. J.* Vol. 7 No. 4, pp. 801-815, 1991.

PARKER, TOM, CCTA Technical Report: Single Sign-on systems, Feb. 1994

ROGERS, DAVE, *ICL Tech. J.* Vol. 9 Issue 2, pp. 272-289, 1994.

8. Biography

J.A.Jones

J.A.Jones has worked for ICL and its predecessor companies for more than 30 years. After taking a degree in Oriental Studies and Classics at Cambridge University, he worked as a hardware designer in a number of fields, then went on to specialise in mainframe system design for several years, as Medium Systems Design Manager. Next he worked for a period on Personal Systems: initially on the One-per-Desk project, where he took charge of several of the more innovative subsystems, and later on the developing ICL Personal Computer range.

He is now one of the Systems Designers in the Business Development unit of Enterprise Engineering, responsible for coordination and management of development activities for major customer project bids, and currently specialises in secure UNIX systems.

POSIX SECURITY FRAMEWORK

David Rogers - Data logic Ltd and Jane Ross - Independent Consultant CHOTS Project - ICL Basingstoke, UK

Abstract

Confidence in the security of an IT system or product demands that the security services and security mechanisms it provides are both effective, and strong enough to counter perceived threats to its operation. The effectiveness and strength of a security mechanism depend on where they are sited within a system architecture. This paper discusses some of the concepts from the POSIX security framework, and illustrates how a practical security architecture can be built from it. The example used also serves to demonstrate the flexibility of the model in implementing diverse security policies and distributed computer system configurations.

1. Introduction

Organisations, large and small, rely increasingly on IT systems to meet their business needs. As greater computing power becomes available in desktop machines, many enterprises are *right sizing* their IT systems by replacing or supplementing standalone departmental systems with networks of smaller machines and PCs. Corporate networks continue to expand, bringing increased demands for greater connectivity and greater inter-operability between (and among) disparate computer systems.

Such open systems make the computing resources of the enterprise accessible to many more users and, consequently, expose it to greater threats. Countering such threats requires the computer system to provide controls to safeguard both processing and the data it holds.

To devise a suitable set of controls, an organisation needs a clear statement of its security requirements. This set of rules, or **security policy**, identifies the assets of a company that must be made secure, who is responsible for them, and how they should be accessed and managed. Where assets are held on the IT system, appropriate security features or mechanisms may need to be introduced to safeguard them effectively. A secure operating system, virus checkers, trusted network components such as SecureWare MAXSIX [MAXSIX, 1993], distributed systems products such as OSF DCE [OSF, 1992] may provide some, but not all, of the

protection an enterprise IT system requires. Their effectiveness depends on what security functions they provide, where they are sited within an IT system architecture and how they interact with other components in the system, namely, on defining a system security architecture.

The POSIX[1] Security framework can assist an organisation to define a suitable security architecture. Its concepts represent an intellectual framework in which to reason about the security of the system; this is essential to make sense of security issues in open, distributed systems, where security depends on complex interworking of many components. The POSIX Security Framework defines security services, and how they interact with and depend on other services in order to ensure the security of the system overall.

An objective of the framework is to identify security service application program interfaces (APIs) that could be standardised, together with a model of how security and the different levels of application security awareness affects APIs for other services. The security framework therefore establishes a basis for the definition of security service APIs as POSIX standards and the modification of existing APIs to address security. The existence of these standard APIs will, in and of itself, encourage product developers to use them to build a greater range of IT security products; the framework will also help system integrators to build practical, secure IT systems from such products. To the enterprise, these developments mean that secure systems can be assembled from off-the-shelf components, at lower cost, and readily and suitably integrated with other system components to meet the security objectives.

This paper presents an overview of the concepts of the POSIX framework [POSIX, 1994], which is in the process of being balloted. Using the example of a distributed database, it demonstrates how the framework can define suitable security architectures for this simple system. As will be seen, several architectural approaches are valid; all are accommodated by the security framework. Such flexibility means that the framework is a powerful tool in reasoning about and defining the security architecture of complex distributed systems.

---

[1] POSIX is a standards project sponsored by the IEEE Computer Society's Portable Applications Standards Committee. Its goal is to promote portability of applications across open systems environments by defining application programming interface (API) standards. POSIX progresses these standards by relying on voluntary effort from industry; standards are agreed by consensus amongst its members.

2. Security Framework Concepts

A *security domain* embraces those business activities and resources, procedures and IT systems and services, falling under a single *security policy* and a single *security authority*. The POSIX security framework adopts the terminology defined within the ISO security framework [ISO 10181-1, 1992] to provide consistency within open system standards.

Responsibility for enforcing the security policy of an organisation usually rests with the board, who are its ultimate security authority. They, in turn, may delegate the responsibility and authority for enforcing specific aspects of the enterprise security policy to heads of departments as it relates to the assets the departments maintain and control. Where some of those assets, that is information, are held on departmental IT systems, then aspects of security policy enforcement are further delegated to the IT systems themselves. The enterprise, the department and the departmental IT system are all regarded as security domains within the POSIX security framework. The department and department IT system are *subdomains* of the enterprise domain, being contained within that domain and subject to the overall enterprise security policy.

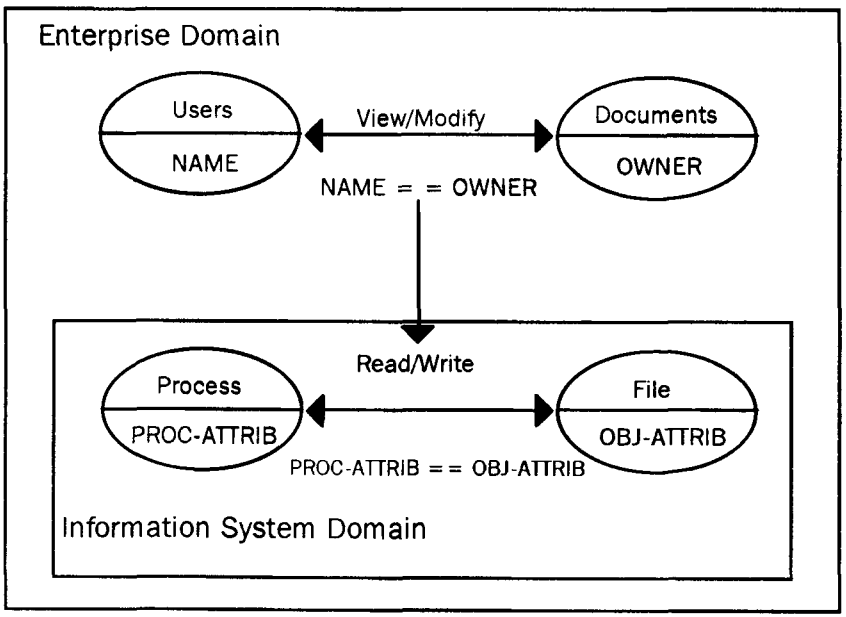


Figure 1 Example of IT System as Subdomain of Enterprise Domain

For an IT system to enforce aspects of the enterprise security policy, the information, policy rules and procedures of the enterprise security policy must be translated or mapped to IT system elements, rules and activities.

A simplified example is illustrated in Figure 1. Within the enterprise security domain, the policy states that only the *owner* of a document is permitted to view it or to modify its contents. Each individual has an attribute, his or her name; each document has an attribute, its owner. Within the IT system domain, processes represent users; files represent documents. Each system element is given an attribute corresponding to that of the entity it represents in the enterprise domain. Thus, the process attributes identify the user; the file attributes identify the owner. The policy rule is mapped to controlling read and write access by the process to the file, and permits access if, and only if, the process attributes correspond to the file attributes.

The POSIX security framework models an IT system domain as a set of security domains. The so-called *platform domains* [POSIX, 1992] [Rogers, 1993] [Rogers & Ross, 1993] provide the infrastructure for the IT system; they comprise hardware, operating systems and underlying communications. The platform domains support *Service Domains* [POSIX, 1992] [Rogers, 1993] [Rogers & Ross, 1993], which comprise applications, substantiated by the resources of platform domains, for example processes and files. The operations of, and interactions between, service domains (applications) therefore comprise interactions between platform domain resources and are therefore subject to the security policies of those platform domains. Thus, service domains are subdomains of a platform domain.

A platform domain is primarily responsible for the allocation of resources and the segregation of data and data flow within and between those resources. A service domain, as an application, utilises the resources of a platform domain to provide an information processing service. A service domain may provide a specific security service, such as authentication; equally, it may consist of a security relevant application such as a print server, email server or database.

3. Inter-relationships Among Security Domains

IT security domains, represented by platforms and applications, can interact in many ways. On a standalone computer system, for example, the service domains interact with one another and with the platform domain which supports them. Where two or more computer systems are networked additional interactions can occur, between platform domains and between a service domain on one platform and a service domain on another. A fully distributed system, such as one using DCE, provides security services that facilitate the creation of such interactions. These three examples are illustrated in Figures 3, 4 and 5.

Interactions between security domains depend upon the existence or creation of inter-relationships between the domains. In accordance with the ISO security framework [ISO 10181-1, 1992] the POSIX security framework recognises two types of inter-relationships among security domains, which reflect the way in which enterprises, departments and

individuals interact. *Administrative inter-relationships* are those which are established by management action and are persistent in nature. They define the permitted activities and the policy which must operate between any two security domains. *Operational inter-relationships*, on the other hand, are transient in nature, and exist solely to service a particular set of transactions within the context of the administrative inter-relationships previously established. Both administrative and operational relationships establish *trust* between the security domains. The POSIX security framework uses trust in the sense defined by the ISO security framework where an element x trusts element y for a set of activities within the context of a security policy, that is, x trusts y to behave in a well-defined way that does not violate the security policy.

Trust must be established between any two domains which need to interact with one another. Administrative actions establish the basic persistent trusts which are to exist between security domains. For a standalone computer system, this involves configuring the security attributes of the software and data files on application installation such that the platform is capable of protecting the boundaries of each service domain it supports, and of constraining access to the data resources within that domain. A further example of trust establishment is the task of adding a user to the user database. Persistent security information regarding that user, such as a unique identity, a password, a set of permitted terminals, perhaps an operating clearance, is configured on the system for use by the service domain which authenticates that user at login.

Where systems are interconnected, additional configuration is required of network services. Where security domains do not share a common meaning, a common representation of security attributes, or a common set of security rules, as may be the case in inter-connected systems, then an *inter-domain service* is required to map meanings, attributes and rules from one domain to the other.

Operational inter-relationships are established by the exchange of security information or attributes which have been previously defined by administrative action. These establish the security context of the interaction. Login, for example, provides two services:

- a service which authenticates users
- a security attribute service which establishes the security context for a user's session.

The security attribute service may map the security information associated with that user and held in the user database, to platform security attributes. The platform then assigns these platform security attributes to the processes which are executed on the user's behalf.

4. Security Domain Model

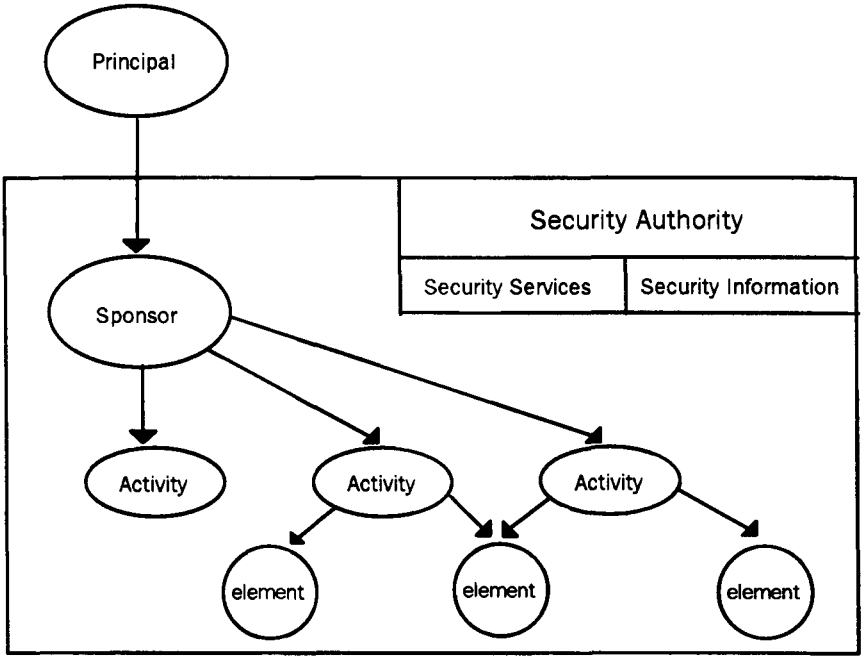


Figure 2 Security Domain

A security domain provides services to *principals* which are external to the domain. See Figure 2. A principal may be an end-user, i.e., an individual, or another service acting on behalf of an end-user or service. In general, a security domain is required to:

- protect the integrity and confidentiality of its resources and its activities
- ensure the availability of, and account for the use of, the resources under its protection.

The POSIX security framework requires a principal to interact only with a *sponsor* located within the security domain. The sponsor acts on behalf of the principal within the security domain both facilitating and constraining the principal's interactions with the activities and elements of the domain. The sponsor is responsible for the enforcement of the domain security policy through invocation of the domain security services. For example:

- verifying the authorisation of the principal to interact with the domain as a whole

- establishing the security context of the principal's interaction with the domain, for example a representation of his access rights within the security domain.

To fulfil these responsibilities, the sponsor uses the authentication, security attribute and authorisation services of the security domain to establish for the principal, its rights to perform specific activities with respect to specific elements of the domain. [ECMA, 1988] [ECMA, 1989].

5. Interpretation of Common System Configurations

The POSIX security framework can be applied to diverse system architectures. Here, we examine the framework in relation to different types of systems, namely:

- a standalone computer system
- distributed systems

and consider the administrative implications of each.

6. Standalone System

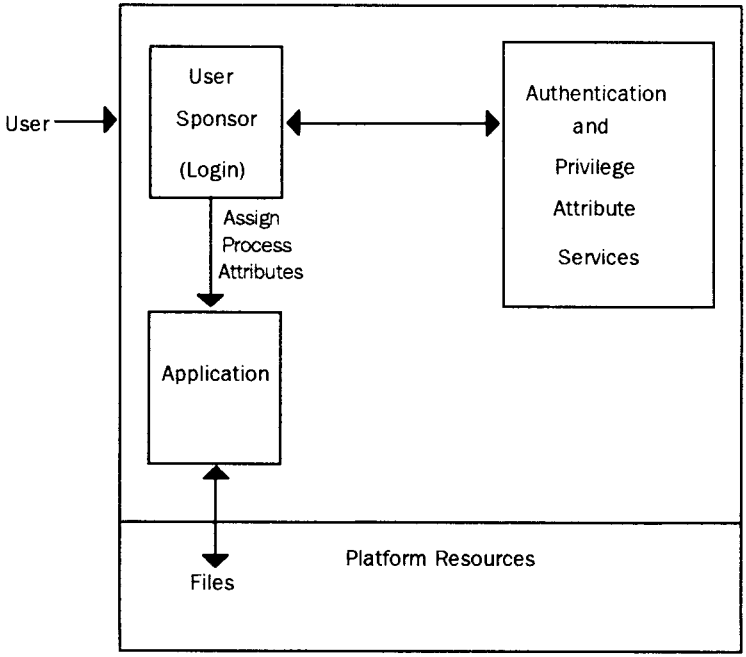


Figure 3 Standalone System

A standalone system is a single platform domain which supports a set of applications, or service domains. See Figure 3. The platform domain and its applications are controlled by a single administrative authority, which establishes the trust relationships within the system by configuring security attributes at the time of software installation. Specific capabilities may be assigned to an application, for example, update user database (password file). The attributes of the elements of the platform, i.e., on its processes, files and communication channels, may be pre-set, or set using trusted utilities.

The platform domain then enforces security policy with respect to those elements, and the security attributes assigned to them. All applications rely on the platform domain to maintain the association of the principal's security attributes with the chain of processes executed for the principal. In addition, the platform provides services allowing applications to interrogate those attributes as required.

7. Interconnected Systems

The principal difference between standalone and distributed systems, illustrated in Figure 4, is the interconnection of platforms to permit the users of one platform to use services supported by another platform. An *interconnection policy* needs to be established, defining how, and when the systems can communicate, and the nature of those interactions. Each system then needs to be configured such that the security information it contains supports the formation of operational inter-relationships between the two systems. A communications channel provides the means by which elements in one system may be accessible to the other.

The introduction of a communications channel into the overall system introduces additional security threats via the communications media and may also increase the potential number of access points to the system. Within a standalone system, applications and Service Domains, generally rely upon the platform domain to protect the integrity and confidentiality of the application data, essentially based upon the physical protection of the platform. The same level of physical protection is not normally applicable to communication media and cryptographic methods may need to be deployed to protect the application data. Such services may be provided by the platforms, as part of the lower layer communication services (OSI layer 4 and below) or by the Service Domains (applications) themselves (OSI layer 7).

8. Distributed Systems Security

The interconnection of systems requires the exchange and sharing of security information between the systems. Variations in the nature of the exchange and sharing of security information represent variations in the strength of the security services and of the distribution of security management responsibility.

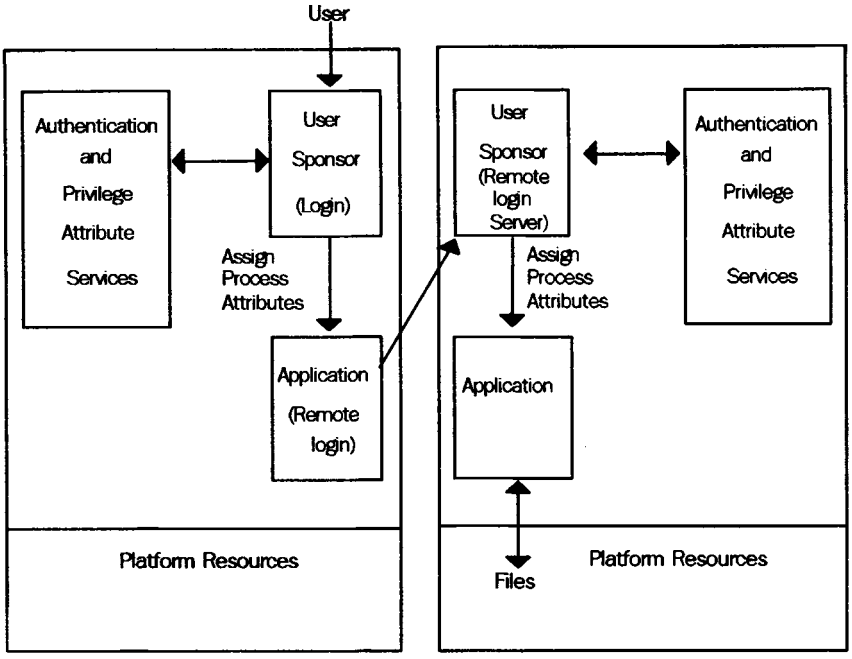


Figure 4 Interconnected Systems

The weakest approach is illustrated in Figure 4 in which each system is independently responsible for management of access rights to its services and effectively continues to operate as a standalone system. A user is required to present an identity and credential to each system with which the user interacts, just as for the standalone system. Each system remains responsible for assigning security attributes to authenticated users. The User I&A may be achieved by the means of scripted access to hide the requirement from the user. Such an approach requires the establishment of mutual trust between the systems and their administrators as cleartext security information is processed and exchanged.

A stronger approach for the exchange of security information, based on trust of each platform domain, is to provide a trusted IPC service, of which SecureWare MAXSIX [MAXSIX, 1993] is an example. In this case the security authorities of each platform trust the other platforms to authenticate users correctly and then rely upon the identity and security attributes assigned by the originating platform. The platform domains associate security attributes with process execution paths and propagate these security attributes across the network IPC service. Interdomain services are required to map the representation of security attributes on one platform with their corresponding representation on another. Default attributes for a particular mechanism can be defined where one platform does not support the security mechanism involved.

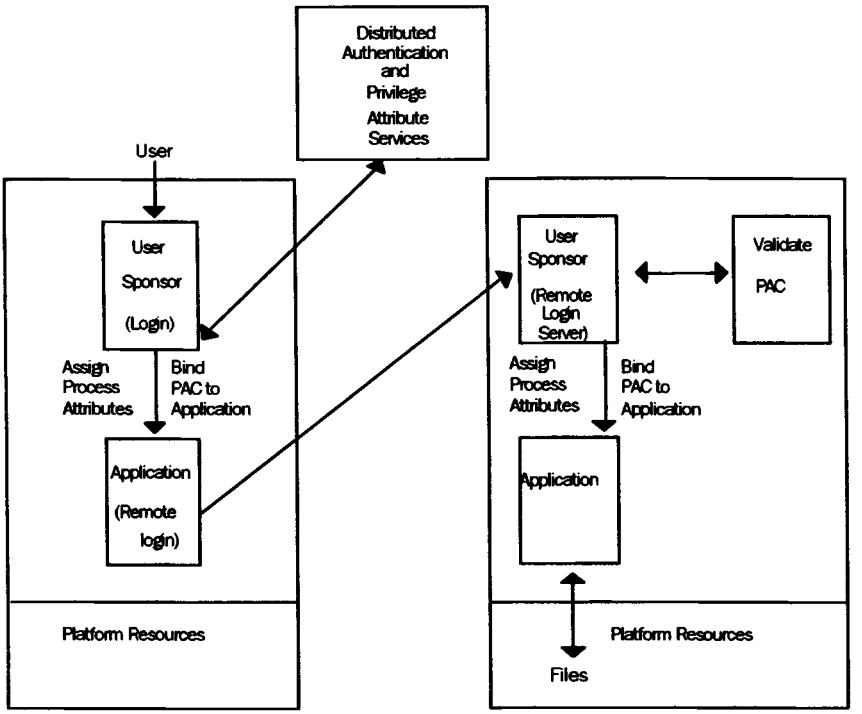


Figure 5 Distributed System

A further strong approach, based at a service domain level, is illustrated in Figure 5 in which trust is shared in a set of common security services to which authority for management of access rights to each platform's services is delegated to the common security services. Kerberos [Steiner et al, 1988], OSF DCE [OSF, 1992] and SESAME [Kaijser, 1993], [Parker, 1991] are examples of this approach. The sponsors of each platform use common authentication and security attribute services that may be located on a single platform. The security information is transferred in the form of security certificates or tokens, i.e., cryptographically sealed. The seals can be verified on receipt to establish the authenticity of the information. These types of service are provided at the service domain level and are independent of the platforms used to support the services. Thus, if the platform is required to enforce security policy in respect of the principal, then the security information will need to be mapped to a local platform representation, for example, the mapping of a DCE UUID to a UNIX uid.

9. A Practical Security Architecture

The security framework is a powerful tool in helping to define a suitable security architecture. This is illustrated, here, by considering the security features required to provide controlled access to a distributed database. A distributed database is a common example of a distributed application

which enforces security policy with respect to the information it contains and processes. The examples presented here are simplified to demonstrate how the security framework can be used.

A DBMS Security Policy may typically require that:

- the DBMS shall support services to multiple principals and shall protect one principal's information from another's, i.e., maintain the integrity and confidentiality of the information
- the DBMS shall account for the use of its services and access to the information it contains.

In order to support the above security policy, the DBMS is required to enforce access controls such that the identities of the principals requesting to use its services are authenticated. Access is then mediated in accordance with their assigned access rights. The provision of such services depends upon the configuration and maintenance of security information related to the identities and access rights of the principals authorised to use the DBMS. The effectiveness of the security services depends upon the correctness of such security information. This leads to the following assertion:

- that the integrity (and confidentiality as required) of the security information used by the DBMS domain shall be protected.

Two example architectures are considered to support the DBMS, both distributed in nature. For each, the authentication service is provided by system components external to the DBMS domain, although this is not the only option. In the first, the authenticated identity of the principal is provided by platform domain services. In the second, the authenticated identity is provided by a distributed security authentication service, such as Kerberos. The examples demonstrate how the assertion above leads to corresponding assertions on other system components and system administration.

The types of assertion illustrated by the examples define the trusted behaviour of the other components comprising a system necessary for the DBMS domain to uphold its own security policy. The objective of the definition of a security architecture is to ensure that these trust dependencies can indeed be upheld. Administrative inter-relationships between the system components are established by configuration on system installation. A security product should include such assertions within any definition of a Philosophy of Protection statement for the product: they define the trust environment for which a product has been developed and within which it should be used and evaluated.

These assertions also define the constraints required on the activities of system administrators and hence emphasise the significance of system management in the security of an information system. The constraints on system administrators are essentially to maintain the administrative inter-

relationships required by the security architecture and to co-ordinate their respective activities.

10. Platform Security Service Model

In this architectural model, the DBMS exists as a service domain within one platform domain; the client exists as another service domain on a second platform domain inter-connected to the first. See Figure 6. The platform domains provide a secure association service, based on process attributes, such that the client and DBMS domains can interact.

The DBMS trusts the platform domain supporting the DBMS to provide it with the authenticated identity of each principal requesting to use its services. This is an example of an administrative inter-relationship established by context. The authenticated identity is derived from the initial user login on the platform domain supporting the client.

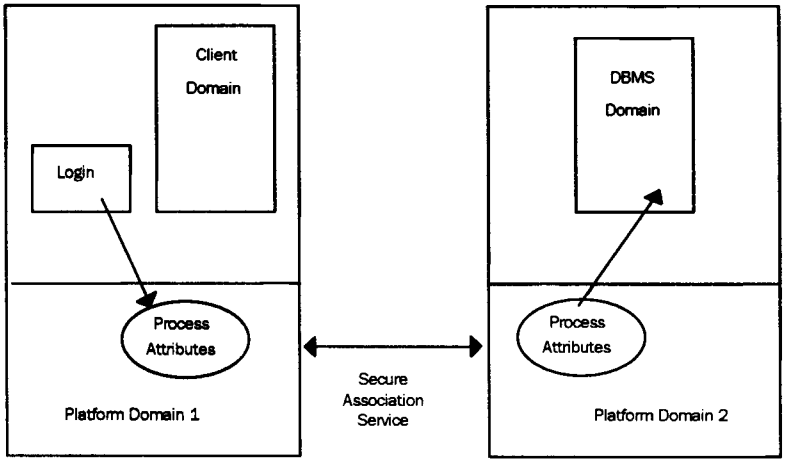


Figure 6 Platform Security Service Example

The trust relationships between the local sponsor, here the login program, and its supporting platform domain, between the platform domains, and between the DBMS and its supporting platform domain are established by the persistent administrative inter-relationships in the model. The transient operational inter-relationship between the client's principal and the DBMS is based upon these inter-relationships and established by the exchange of security information.

A user interacts with the local sponsor (login) to establish a session on his local platform. The sponsor invokes the local authentication and security attribute services to authenticate the user and stores the security information as process attributes, which the platform associates with all processing on behalf of that user session. The platform also protects the integrity (and confidentiality as required) of those process attributes, satisfying the assertion.

When establishing a communications channel between two processes, the platform domains are required to provide a secure association service which makes available to each platform domain, the process attributes of the process in the other platform domain using the communications channel. This may require the invocation of an inter-domain service to map the representation of process attributes in one platform domain to a representation in the other. In this model the security information is exchanged by the platform domains.

The DBMS establishes the authenticated identity of the client's principal by interrogating the platform domain. It may then proceed to verify the authorisation of the principal to access its services based upon its own security information.

10.1 Some assertions arising from this model

This model architecture makes a number of assertions on each of the security domains. On the platform domain administrations, it makes the assertion that:

- the administrators of the platform domains shall maintain consistency between the security information maintained on their platforms, i.e., principal identifiers shall not be changed or reassigned independently.

It makes a similar assertion that administrators of the DBMS domain and supporting platform domains

- shall maintain consistency between the security information maintained within their respective domains, i.e., principal identifiers shall not be changed or reassigned independently.

Local authentication services are required to:

- authenticate principals correctly.

The following assertions are placed on platform domains, that they:

- protect the integrity (and confidentiality) of security information representing the authenticated identity of a principal
- protect the integrity of the association of the security information representing the authenticated identity of a principal with the chain of execution on behalf of that principal.

11. Distributed Security Service Model

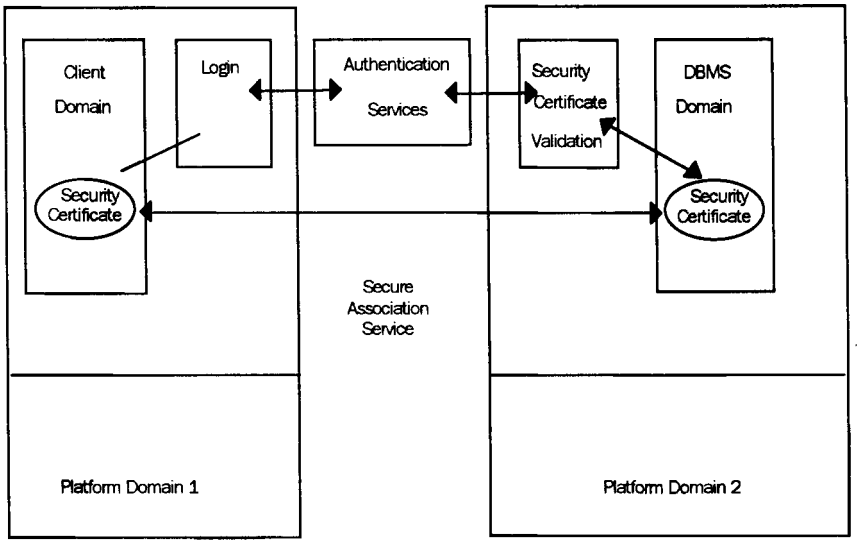


Figure 7 Distributed Security Service Example

In this architectural model the DBMS exists as a service domain within one platform domain and the client exists as another service domain within a second platform domain interconnected to the first. A common authentication service is provided by a further service domain which may be located within either of the two platform domains, or a third platform domain. See Figure 7.

The DBMS trusts the authentication service to certify the authenticated identity of each principal requesting to use its services. This is an example of an administrative inter-relationship established by security information e.g., cryptographic keys.

The trust relationships between the local sponsor and the authentication service, and between the DBMS and the authentication service are established by the persistent administrative inter-relationships in the model. The transient operational inter-relationship between the client's principal and the DBMS is based upon these persistent inter-relationships and established by the exchange of security information.

A user logs in to establish a session on his local platform. The local sponsor (login) invokes the distributed security authentication service to authenticate the user. The security information returned is used to establish the user session possibly by mapping to a local platform representation of security information, such as process attributes.

When the user invokes the services of the DBMS, the platform domains create the communication channel. The client then sends the security

certificate, returned by the authentication service, to the DBMS process. The DBMS verifies the authenticity and integrity of the security certificate by invoking its own validation service. This will verify that the certificate originates from the distributed security authentication service and has not been modified and hence that the principal identity contained therein is valid. The DBMS may then proceed to verify the authorisation of the principal to access its services based upon its own security information.

In this model the authentication service may trust the client and the client platform to protect the security certificate from unauthorised use; alternatively it may implement cryptographic measures.

11.1 Some assertions arising from this model

This model architecture makes a number of assertions on each of the security domains. On the DBMS domain and authentication service domain administrations, it makes the assertion that:

- The administrators of the DBMS and distributed security authentication service domains shall maintain consistency between the security information maintained within their respective domains, i.e., principal identifiers shall not be changed or reassigned independently.

The Distributed Security Authentication Service is required to:

- authenticate principals correctly.

The following assertions are placed on the local sponsor and DBMS client:

- The local sponsor and client shall protect the integrity (and confidentiality) of security information representing the authenticated identity of a principal
- The local sponsor and client shall protect the integrity of the association of the security information representing the authenticated identity of a principal with the chain of execution on behalf of that principal.

In practice these also lead to assertions on the local platform and platform administration to protect the security information. For example, the system administration usually has sufficient capabilities to access all data held within a platform and therefore an assertion has to be made that such capability is not abused.

12. The Future

The POSIX Security Framework is being balloted during October 1994 [POSIX, 1994]. Dependent upon the outcome of that ballot the POSIX Security Framework may be expected to be published during the course of 1995. The participation of other standards bodies has been actively sought in order to bring together other framework developments in the standards arena [ISO 10181-1, 1992] [ECMA, 1988] [ECMA, 1989] and among other interested parties such as X/Open and OSF. The resultant

framework should then reflect the composite thinking of all such groups. A significant co-operation has been the joint development of the Security framework with X/Open. The X/Open version [X/Open, 1994] is going through the X/Open company review procedure during September and should be published as a Guide towards the end of 1994.

As an internationally recognised security framework emerges, it will stimulate the development of off-the-shelf security products which can be readily integrated into system solutions. By identifying the security services needed and their interfaces and dependencies, the security framework will highlight those for which products do not currently exist and how they are to interact with other security services and platform security features. Product manufacturers will be able to then build products or enhance existing products to fill these gaps, using the interfaces identified by the security framework and defined by appropriate POSIX API standards. [Rogers & Ross, 1993]

Enterprises will eventually benefit from these developments. The emergence of off-the-shelf security products will ease procurement, and reduce the costs of building a secure system. The framework, itself, will assist those defining security architectures for such systems, by giving them the concepts with which to reason about security features, their placement, and their interactions with other components in the system. As the simple distributed database model demonstrates, several security architectures are possible, each capable of meeting the same security policy. The POSIX security framework provides a structure for developing these architectures and comparing their relative merits. The approach, based on security domains, also identifies the potential management interactions required to support a particular security architecture and their importance.

13. Acknowledgements

This paper is a revised version of one by the same authors and the same title presented at COMPSEC 1993 and published in the conference proceedings by Elsener. It is printed here by permission.

14. References

Papers:

KAIJSER, P: "Distributed Access Control Issues", Proceedings of the Information Security Conference, DataPro, 1993.

PARKER, T A,: "A Secure European System for Applications in a Multi-vendor Environment (The SESAME Project)", Proceedings of the 14th American National Security Conference, 1991.

ROGERS, D, ROSS, J: "POSIX Promises Security", Open Systems Networking and Computing, Vol. 7, No. 7, July, 1993.

STEINER, J, NEUMAN, C, SCHILLER, J: "Kerberos: An Authentication Service for Open Network Systems", USENIX Winter Conference, 1988.

Technical Reports:

OSF, 1992 "OSF's Distributed Computing Environment" (DCE), DataPro, 1809, McGraw-Hill, May 1992

MAXSIX, 1993 "MAXSIX: Trusted Networking from Project Max", SecureWare, Inc, January, 1993

ECMA, 1988 "Security in Open Systems -- A Security Framework" (ECMA-TR46), ECMA Technical Report, July 1988

ECMA, 1989 "Security in Open Systems -- Data Elements and Service Definitions" (ECMA-138), ECMA Standard, December 1989

ISO 10181-1, 1992 "Information Technology - Security Frameworks in Open Systems - Part 1: Security Frameworks", ISO Committee Draft, ISO/IEC CD 10181-1, December 1992

POSIX, 1992 "A Distributed Security Framework for POSIX, A White Paper", IEEE POSIX Distributed Security Study Group, Nov 1992

POSIX, 1994 "Guide for POSIX Open System Environment -- A Security Framework, Draft 5", IEEE POSIX 1003.22, August 1994

ROGERS, D. "Guide to the POSIX Open Systems Environment - A Security Framework Submission for Security Domain Definitions", March 1993

X/OPEN, 1994 "X/Open Distributed Security Framework", Company Review draft, August 1994

15. Biographies

David Rogers

David Rogers graduated from Oxford with an MA in Physics in 1970 and is a Fellow of the Institute of Chartered Accountants of England and Wales.

He has spent over five of the last seven and a half years as a security and system architect on the CHOTS (Corporate Headquarters Office Technology System) for the UK Ministry of Defence. This is a large-scale multi-level secure distributed office automation system. The other two years were spent leading a design study for a high assurance secure UNIX system for CESG. During the last five years Data Logic has funded his participation in the IEEE POSIX standards effort as part of the security working group. He has been leading the work on distributed security within POSIX for the last two and a half years. He is project leader of P1003.22, the POSIX Security Framework project, and is a vice-chair of the security working group itself. He has also been commissioned by

X/Open to work on the production of the X/Open Distributed Security Framework.

drogers@datlog.co.uk

Data Logic Ltd

CI Tower, St George's Square, High Street, New Malden

Tel: +44 (0)81 715 9696 Fax: +44 (0)81 715-1771

Jane Ross

Dr. Jane Ross is an independent consultant specialising in IT security and systems technology. Since completing her doctorate in electrical engineering at Imperial College, she has worked extensively in the computer industry on the design and development of hardware and software systems.

For the past six years, she has acted as a consultant, concentrating on security and integration of systems based on open systems technology. She developed security policies for her clients, and successfully planned and implemented several migrations to open systems technology. Her background in electrical engineering and software development has been put to good effect in the design of several state-of-the art secure open systems products systems. Counted among these are two high assurance secure operating systems designed to meet the POSIX standards for secure UNIX, and the CHOTS secure distributed office automation system. She has published a number of papers including two joint papers with Dave Rogers on the POSIX Security Framework.

Solutions That Work
23 Chester Drive
Harrow
Middlesex

Tel: +44(0)81 427 5894

SQL Gateways for Client-Server Systems

J.L. Venn

Government and Major Companies Division, ICL, Slough, UK

Abstract

This paper addresses the problem of incorporating non-relational databases, which are often difficult to use because their interface languages demand specialist knowledge, into modern client-server systems.

It describes a gateway driven by SQL, which is well-known and for which international standards exist, as the interface language to an IDMS database. The gateway forms part of the DAIS distributed system (formerly known as RIBA) which was the subject of an earlier article [CROCKFORD & DRAHOTA, 1992].

The language analysis and data retrieval techniques which are used are described.

It is believed that the approach is practical and inexpensive and may well be used for gateways to other non-relational databases.

1. Introduction

As the use of distributed systems using some kind of *client-server* architecture increases, the need for a common interface language for accessing the databases which form an important part of these systems has arisen. The most likely candidate for such a language has long been seen to be SQL (Structured Query Language), which originated in IBM in the 1970's during the early development of relational databases. Its use has spread steadily in universities and in industry, and vendors of relational databases such as Ingres which originally had their own interface language were eventually forced to adopt it. It became first the de facto standard for relational databases, and later, with the publication of X-Open and ISO standards, a recognised international standard. It is used not only for communication between machines and human beings but for interworking between distributed databases. Today SQL has the great advantage that most relational database systems (e.g. Ingres, Oracle, Informix) already incorporate an SQL interface. There are considerable dialectal differences between these, but there exists a reasonable common subset which allows

data to be retrieved or updated fairly successfully by SQL without the user being conscious of what kind of relational database it is being retrieved from.

With non-relational databases, there is in general no such common interface. Yet there are a large number of such databases in existence: hierarchic databases such as IDMS, IMS and Adabas, and a variety of specialised databases using indexed sequential or hashed access mechanisms. These have not been swept away, as was promised, by the relational revolution. Typically they are larger than relational databases, sometimes containing several gigabytes of data and conversion would be difficult and expensive. Mostly the databases are in themselves perfectly satisfactory, and their only disadvantage is that they have a specialised and somewhat dated interface language. Clearly it is desirable that such databases should be easily accessible to the new class of distributed systems which are now being written.

The most obvious language for this purpose is SQL. It would be difficult, if not impossible, to implement full SQL for a non-relational database because this includes commands such as CREATE TABLE which changes the database schema dynamically, CREATE VIEW which creates a virtual *table* which has no real existence in the database, and GRANT which is concerned with security permissions. It is not sensible to try to implement commands such as these with a database such as IDMS which does not support the underlying facilities. But if one chooses a less ambitious objective, such as to implement the data manipulation commands of SQL (which is all that is necessary to provide a usable gateway), the task becomes achievable and cost-effective.

This article describes an approach to the design and implementation of such a gateway. The approach has been used successfully to implement an SQL gateway as part of the RIBA distributed system (now marketed as DAIS), described by Tony Drahota and Len Crockford [1992]. The main database accessed is IDMS (or more specifically IDMSX) running on a VME mainframe. This gateway has been in service at a customer site since July 1993. A variant has been used to access VME indexed sequential files.

2. Requirements

Within a client-server system, an SQL gateway is a server which allows a client application programmer or interactive user, who knows SQL but may be ignorant of the native command language of the database concerned, to retrieve data from a database and update it with a reasonable degree of efficiency.

To achieve this it should:-

- implement most of the data manipulation commands of SQL, conforming to generally accepted standards of syntax
- handle any existing data format that is likely to be encountered on the target database
- give a reasonably good level of performance
- be reliable
- be capable of being implemented cheaply and quickly.

There will probably be additional requirements arising from the nature of the client-server system of which it is to form part. DAIS is a distributed system which allows clients to access data without being conscious of which database it resides in, using a language called CSQL (Conceptual SQL). DAIS, or more specifically the component known as DAIS/IS (the DAIS Information Service), acts as an intermediary between the user (client) and the gateway (server), and by reference to a model converts the user's CSQL into SQL which is passed to the gateway. A single CSQL statement may quite possibly give rise to two or more SQL commands which are directed to different databases. (For a detailed description of this architecture and the role of modelling in DAIS, see [CROCKFORD & DRAHOTA, 1992]. The SQL gateway is the Logical Information Server for IDMS).

The DAIS/IS environment has several important effects on the requirements of a gateway:-

- The SQL it receives from DAIS/IS is machine-generated. In principle advantage could be taken of this fact to implement a restricted subset, but it has always been intended that the gateway should be capable of working in other environments, and it in fact handles a much fuller subset than is strictly necessary for DAIS/IS.
- There is a self-imposed DAIS/IS *transparency* requirement [CROCKFORD & DRAHOTA, 1992], which demands that the SQL syntax should be the same as would be used to retrieve the same data from an equivalent relational database. Deviations from standard syntax which might have made it easier to handle the special features of IDMS have deliberately been avoided.
- As a DAIS client may be an interactive application, for example a WINDOWS-driven query system, the gateway is required to interpret and execute SQL commands *on the fly* and any approach which depends on a compilation phase is precluded.
- On the output side, the detailed presentation and formatting of data is handled by client applications or by DAIS/IS; the gateway is only required to return each *row* of data as a sequence of character strings.

3. The DAIS Approach

3.1 Data Definition

The first necessity of a gateway intended for interactive use is a definition of the database structure (loosely referred to as *the schema*, though in IDMS terms it consists of the schema, storage schema and subschema) in a form which can be accessed efficiently at run-time. Relational databases usually have built-in run-time tables. IDMS, which is usually accessed by pre-compiled programs, does not, and the gateway therefore has a *prepare* phase in which the static definition of the schema is read and a set of run-time tables is produced. For each database, this only has to be done once (or whenever the schema changes).

These tables are co-ordinated at the logical level with the DAIS/IS model, but contain additional information which relates only to IDMS.

There are six tables in all which describe the schema:-

- The Data Table, which contains the names of data items and the numeric codes by which they are referred to in other tables.
- The Data Details Table, the Record Table the Key Table, and the Set Table contain details of data items, record-types, keys and IDMS sets. (IDMS sets are groups of related records linked together by embedded pointers or special set indexes - see section 3.3).
- The Record/Realm Table, which shows which *realms* or areas of the database each record-type is contained in.

The prepare phase also generates a COBOL procedure which is used to access the database at run-time. IDMS, unlike relational databases, is only concerned with records, not with individual data items (except for keys), and provides no mechanism for retrieving or updating them. This function is usually performed in application programs by COBOL, i.e. the record is read as a whole into a COBOL working-storage area and the items are picked out by COBOL statements. Since IDMS itself does not do this, the gateway has to provide a way of doing it.

One way would be to provide run-time code to emulate COBOL. This code would need to be told the exact displacement, size and type of each data item within the record, and would need to be capable of handling a wide variety of data formats. These include integers of any length from 1 to 18 bytes, which may or may not be word-aligned; single and double length floating-point; and packed decimal with several alternative ways of representing the sign; all to be done on the fly, without the benefit of compilation.

The gateway avoids the emulation issue by generating not only COBOL code to access records, but also COBOL code to read and write every item in the schema (or speaking in IDMS terms, the subschema). All of this code is contained in a procedure called the Database Access Procedure, which is accessed by means of a macro language from the

main part of the gateway written in 'C'. The Database Access Procedure is simple in structure and it is produced by passing a standard skeleton program through the IDMS COBOL preprocessor, which produces code for access at the record level, and adding code to access items.

Although in the case of IDMS, the run-time tables and the Database Access Procedure are produced by a prepare-phase program, they can also be produced manually. This is quite feasible for databases with relatively small numbers of record-types and data items, and this approach has been used successfully for gateways for indexed-sequential files.

3.2 Language Analysis

In the run-time phase, the first step to be carried out is to analyse the user's SQL and transform it into a series of tables which can be used to control retrieval and update operations.

The method of SQL analysis is best explained by a simple example of a retrieval, a sequence of commands referring to only one record-type:-

```
DECLARE CURSOR CUR1 FOR
SELECT FORENAME, SURNAME, DEPT
FROM EMPLOYEE
WHERE (SEX = M AND AGE > 40) OR (SEX = F AND AGE > 35);

OPEN CURSOR CUR1;
FETCH CUR1;
CLOSE CURSOR CUR1;
```

This example uses the SQL cursor syntax, which is the form currently used by DAIS for retrieval. The effect would be to retrieve one EMPLOYEE record (or *row* in relational terms) for either a male over 40 or a female over 35. In practice, the FETCH would usually be built into a loop so that all records which satisfy the conditions would be retrieved.

The DECLARE and SELECT clauses would cause three entries to be made in a simple table called the Select Table. The names would be replaced by numeric codes, obtained from the Data Name table already described. Similarly the FROM clause would cause one entry to be made in a FROM Table.

The WHERE clause is less straightforward. This is not a comma-separated list like the SELECT and FROM clauses. The basic element of a WHERE clause is a comparison (such as SEX = M or AGE > 35). Comparisons can be linked by logical operators (such as AND or OR) and grouped together by brackets which determine the order of evaluation. Brackets can be nested (potentially to infinite depth). The monadic operator NOT and a variety of functions such as BETWEEN, IN, and LIKE may also occur.

Comparisons are analysed into a simple table. Logical operators and brackets are represented by a logical tree, the nodes of which are chained together by pointers. Each comparison evaluates to TRUE or FALSE, and

the logical tree reduces multiple comparisons to a single TRUE or FALSE value. Figure 1 shows diagrammatically the logical tree of the WHERE clause

WHERE ((SEX = M AND AGE > 40) OR (SEX = F AND AGE > 35)) AND NOT DEPT = SALES;

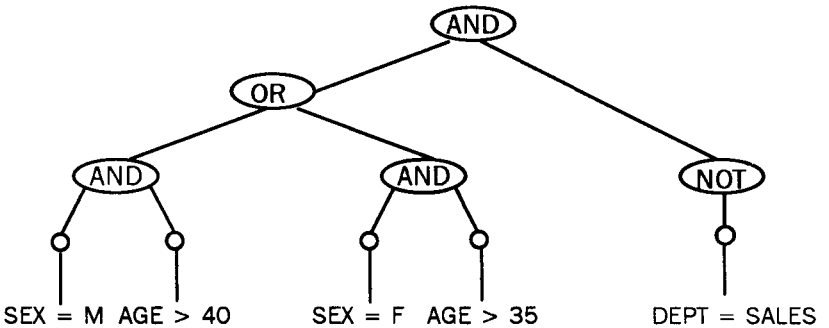


Figure 1 Logical tree representing a WHERE clause

A further table, the Literal Table, is used to store literals used in comparisons.

Actual SQL statements may be much more complex than this simple example. Where two or more record-types are referred to, the FROM clause may contain several elements, and names in the other clauses may be prefixed by record-type plus "." to show which type of record they belong to. Aliases may be used for record-types. The basic structure of a retrieval statement, however, is always similar to the example given, and can be transformed into the tables described. Once transformed, the SQL text itself is never used again.

SQL update commands follow a slightly different pattern but the gateway handles them in a similar way, transforming them into tables and thereafter not referring to the SQL text.

3.3 Joins

If an SQL retrieval refers to two or more record-types, the records are said to be *joined* and the results returned to the user may contain items from both of the joined records.

Take the following example of a SELECT command:-

```
SELECT CUSTOMER_NAME, PRODUCT_NAME, QUANTITY
FROM CUSTOMER, ORDER
WHERE CUSTOMER.ORDER_NO = ORDER.ORDER_NO;
```

This assumes that the database contains two record-types, CUSTOMER and ORDER. CUSTOMER contains items CUSTOMER_NO, CUSTOMER_NAME, and ORDER_NO; ORDER contains items ORDER_NO, PRODUCT_NAME and QUANTITY. If ORDER_NO in

ORDER is equal to ORDER_NO in CUSTOMER, this indicates that the order refers to that customer. This is a relational-type structure, and might exist either on a relational database or on IDMS.

The example SQL would return to the user the items named in the SELECT clause from all pairs of CUSTOMER/ORDER records which satisfy the WHERE clause, i.e. for each customer it would return the customer's name together with the products and quantities he has ordered. The term *row* is used to refer to each such set of results.

There are many ways of organising a join to produce this result. On a very primitive database, with no indexes or keys, it would be necessary to pair together all possible combinations of customer and order records, and apply the WHERE clause to decide whether to return a row of results to the user or to go on to the next combination. This is described as forming a *Cartesian product*, and is obviously inefficient because it involves in principle $n * m$ record accesses, where n is the number of records of one type and m is the number of records of the other type (though this may be reduced by *look-ahead* buffering in the database). For more than two record-types, unless the database is small, the number of accesses would be astronomical.

If the customer record has an index of which CUSTOMER.ORDER_NO is the key, the situation improves. All the order records must be read, but because they contain the key of the corresponding customer record, it is possible to go straight to that record and avoid reading the records of customers which have no orders. In principle the number of records accessed will be $2 * n$, where n is the number of orders, though the actual number would vary according to the numbers of records present and the characteristics of the database. Accesses to the index itself must also be taken into account. *Hashed* access behaves in a similar way to indexed access; these two access methods are common in relational systems.

IDMS is a database of the type known as *hierarchic*. It has hashed access, and usually indexed as well, but in addition has the concept of the *ownership* of one record-type by another. A record can *own* a group (or *set*) of records of a different type, to which it is linked by a chain of pointers embedded in the records. Sometimes a local index takes the place of pointers, but the principle is the same. Each set-type has a name, which is part of the schema of the database.

In a typical IDMS system, customers and orders will be linked by a set, CUSTOMER being the owner and ORDER the member. The most efficient way of accessing these records would be to access all the CUSTOMER records in turn, and follow the set pointers to the ORDER records which belong to it. The number of accesses would in principle be m * the average number of records in the set (where m is the number of customer records). This can be very efficient, especially if advantage is taken of a facility to locate member records in the same *page* or disc block as the owner record, in which case on disc access may retrieve the whole

set. (Oracle relational databases have a similar facility, but this seems not to have been imitated by other relational vendors).

3.4 Join Optimisation

Enough has been said to show that the efficiency of joins can vary greatly according to the access method chosen, and that it is vitally important to exploit features such as keys and sets which may be present.

The main criterion in choosing a join strategy has been to minimise the number of disc transfers. The optimum balance between processor time and latency in the disc system varies with the type of disc and processor being used, but in general, with modern equipment, disc transfers are the limiting factor. The gateway therefore is oriented towards optimising the join before starting to transfer data.

Experience gained in using relational databases, some of which have very efficient optimisers, has been taken into account. Some of these depend on the availability of statistics on the ranges of field values present in the database population; this approach was ruled out because it only gives good results where the population is relatively static. The need to handle IDMS sets is a significant additional requirement, and it was determined that a method which made more use of internal storage and less use of intermediate files and sort-merge processes would be more appropriate to IDMS.

In the gateway algorithm, the order in which records are accessed and the access method are determined by the presence or absence of keys and IDMS sets. The best possible starting point is a record containing a key which is equated in the WHERE clause to a literal, because it can be accessed directly. The gateway scans the logical tree which represents the WHERE clause, picks out any such record-type, and creates a *join node* for it in a working-storage table. It then looks for record-types which can be easily accessed from this record, either because their key is contained in it or because they are connected to it by an IDMS set. If any such record is found, a join node is constructed, which is chained to the join node of the first record by forward and backward pointers. The gateway then looks for further record-types which can be accessed easily from this one, and so on, forming as long a chain as possible of join nodes.

If there is no key equated to a literal, another good starting point is a record which cannot itself be located by key but which contains the key of another record. If no especially good starting point is found a random choice is made.

When it is no longer possible to extend a chain, a new starting point is sought and the process is repeated.

The default access method, for records which cannot be accessed by key or by set, is by a complete scan of the IDMS realm(s) in which it resides.

Join nodes continue to be built, and chained where possible, until every record-type referred to in the FROM or WHERE clause has a join node.

Chains can be *forked*, for example if a record-type contains two or more separate data items which are equated in the WHERE clause to keys of other records, or if it is related by IDMS sets to more than one other record-type.

Altogether five methods of access are used by the gateway. In order of preference they are:-

1) Unique key

2) Duplicate key

3) IDMS set (owner to member)

4) IDMS set (member to owner)

5) Realm scan

Figure 2 shows join nodes for the customer/order example. Figure 3 shows a possible table of join nodes for a join of eight record-types.

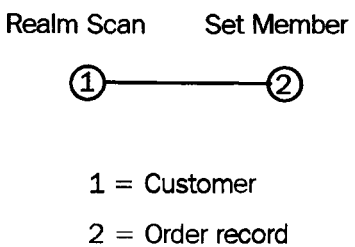


Figure 2 Join nodes for the customer/order example

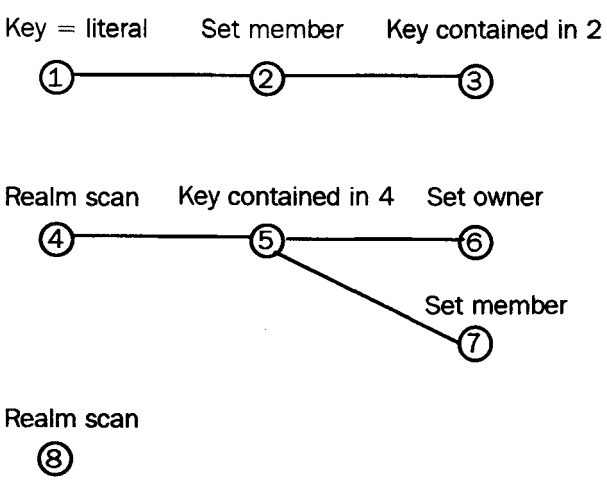


Figure 3 Possible join nodes for an 8-way join

Many SQL commands, of course, will refer only to one or two record-types, but the method is valid for higher numbers.

Analysis of SQL and join optimisation both take place when the OPEN CURSOR command is executed.

3.5 Join Execution

On receiving a FETCH command, retrieval of data begins. Initially one record of each type is read, in the order and using the access method indicated in the join nodes, thus forming a *virtual row* of data in working storage. The WHERE clause is then applied, under the control of the logical tree, and if it evaluates to TRUE the items in the SELECT clause are returned as results.

When another FETCH command is received, at least one new record must be retrieved to form a new virtual row of data. The first record-type for which retrieval is attempted is the one at the end of the last chain of join nodes. If this attempt is successful, the WHERE clause is applied again and if the result is TRUE more results are returned. If the attempted retrieval is not successful (for example because the end of a set has been reached), the backward pointer in the join node is followed and an attempt is made to retrieve a record of the previous type. Retrieval attempts can cascade backwards until the start of a chain is reached. When this happens, an attempt is made to retrieve the record-type at the end of the previous chain. If the start of the first chain is reached and no retrieval is successful, the join is finished and *end of data* is reported.

When a retrieval is successful, forward pointers are followed and forward cascading of retrievals takes place until a record of each type has been retrieved. The virtual row is then complete, and if the WHERE clause is satisfied results are returned.

If a fork in a chain is reached in backward cascading, backward cascading along that chain is halted and an attempt is made to read the record-type at the end of the previous chain. Only when all the *branches* that lead to a fork are exhausted (i.e. retrieval attempts have failed for all record-types) does backward cascading beyond the fork occur. On forward cascading, the virtual row is not considered complete until all branches have been filled.

After each successful retrieval of a record, a partial application of the WHERE clause takes place (i.e. omitting record-types for which no retrieval has taken place since the attempt to form a new virtual row began). This is to ensure that no time is wasted retrieving records whose *parents* do not satisfy the WHERE clause.

All possible combinations of records which satisfy the retrieval mechanisms and the WHERE clause are formed and tested. If there is no WHERE clause, this will result in a Cartesian product.

4. Overall Assessment

4.1 SQL Implementation

The aim of implementing the data manipulation commands of SQL has been to a large extent achieved, but there are still some gaps. For retrieval, only the cursor forms of the commands have been implemented, chiefly because they return data in chunks of more manageable size. The gateway is quite capable of retrieving large amounts of data, and does so in the implementation of summary functions such as AVG, SUM, etc. (though only the summarised value is returned to the user). To support non-cursor retrievals would be a simple enhancement.

In the case of update, only the non-cursor form of the SQL commands is supported. The inconsistency between retrieval and update is not apparent to the user because the DAIS CSQL uses the cursor form for both.

Some special features have been introduced into SQL for the handling of IDMS sets. In the customer/order example which has already been used, the natural way of representing this relationship in IDMS would be to link the CUSTOMER and ORDER records by a set (called say HAS_ORDERED), and not to rely on keys contained in the record. The SQL would then be:-

SELECT CUSTOMER_NAME, PRODUCT_NAME, QUANTITY

FROM CUSTOMER, ORDER

WHERE CUSTOMER.HAS_ORDERED = ORDER.HAS_ORDERED;

This SQL should fail, because there is no item HAS_ORDERED in either record, but the gateway recognises that this is an IDMS set and accepts the command. The DAIS/IS transparency requirement is satisfied because if the data were transferred to an equivalent relational database, an additional field HAS_ORDERED could be created as part of the relocation process.

Another feature introduced specially for IDMS is subscripting. Because the handling of items in IDMS is done by COBOL, repeated items are allowed (by the COBOL OCCURS clause), and subscripting is used to identify which occurrence is required. In gateway SQL, the character "@" is used to indicate a subscript, so that TOTAL@9 means the ninth occurrence of TOTAL. Once again the DAIS/IS transparency requirement is satisfied, because the multiple occurrences could be represented in a relational database by distinct items with names TOTAL@1, TOTAL@2 etc. "@" is a permitted SQL character.

In both cases the SQL syntax is formally preserved, although the semantics are distorted to some extent. It should be borne in mind that, within a DAIS/IS context, the user will be insulated from these features by CSQL and the DAIS/IS model [CROCKFORD & DRAHOTA, 1992]. It is hoped in due course to provide a DAIS SQL interface which is quite

independent of the SQL of the underlying databases, at a level above CSQL. This will be completely SQL-conformant.

SQL subqueries (use of SELECT within a SELECT) are not supported by the gateway because this feature is handled at the CSQL level by DAIS itself. It would not be difficult to add at the SQL level.

On the other hand, many useful SQL features which were not initially required by DAIS, such as the LIKE operator for fuzzy matching, have been incorporated.

On the whole, the SQL syntax supported by the gateway is more than adequate for its role within DAIS, but it is probable that some enhancement would be needed for use in other environments. Besides non-cursor retrieval and subqueries, which have already been mentioned, these include ORDER BY, GROUP BY, and HAVING clauses.

4.2 IDMS Implementation

All data formats commonly found in IDMS and IDMSX databases are supported. Multi-member sets, compound keys, multi-part keys, binary fields of up to 18 bytes, and character fields of up to 200 bytes (easily extendable) can be handled. Packed decimal is not currently supported in the IDMS gateway but has been used successfully in an ISAM gateway and will be added for IDMS. The fact that data access is done by a generated COBOL procedure, which can be edited by users, provides a powerful *user hook* facility which can be used to access more complex data.

Both CALC (hashed) and indexed keys are supported, though there is a restriction in the case of indexed keys in that keyed access will only be used if the operator in the WHERE clause is "=". If the operator specifies a range (e.g. ">" or "<"), the command will be correctly executed but keyed access will not be used.

4.3 Performance

In general performance has been shown to be good, particularly the ability to pick out and use keys and IDMS sets.

In the many cases where realm scans are necessary, performance could often be improved by using CAFS. The current gateway does not use CAFS, because many existing IDMS databases do not have it, but it is hoped to offer CAFS retrieval as an option in the not-too-distant future.

4.4 Reliability

The gateway has been successfully deployed on a customer site since July 1993. No faults were reported during the first year of service.

5. Conclusion

Enough experience has been gained to show that the original design objectives have been achieved and that the design approach has been fully vindicated.

It has also been shown that the implementation of a practical SQL gateway, provided that the objectives are kept within reasonable bounds, can be achieved within a reasonable timescale and cost.

The basic design described here can be used to build gateways for other databases which do not have their own SQL interface; many portable 'C' modules which ICL developed for the IDMS gateway could be used unchanged to provide a basis for such further products.

6. Reference

CROCKFORD, L E, & DRAHOTA, A. "A Support Environment for Distributed Processing", *ICL Tech. J.* Vol 8 (2), pp 284-302, Nov. 1992.

7. Biography

John Venn

John Venn has been in the computer industry since 1955. During the 1960s and 1970s he worked on various special operating systems for the KDF9 and System 4 computers, including the LACES monitor real-time system for HM Customs and Excise which was in use for many years. During the 1980s he was a consultant in government sales with ICL but reverted to a technical role in 1991 to work on the DAIS distributed system. He has been with ICL since 1972.

Asynchronous transfer mode - ATM

Frank Deignan

OPEN*framework* Architect, Networking Services, ICL, Bracknell, UK

Abstract

Asynchronous transfer mode (ATM) networking promises to deliver scalable bandwidth on demand capable of supporting a wide range of multimedia applications whilst also presenting the opportunity to rationalise the incompatible networking techniques in operation today. ATM has attracted a great deal of attention in the press and other media where its merits and weaknesses have been explored. There has also been much speculation on how ATM may change networking as we know it.

This paper briefly reviews networking technology today, looks at the emergence of ATM from traditional wide area networking techniques, provides a technical but non-rigorous explanation of how ATM works outlining some hurdles to be overcome before ATM delivers its promise, and finally analyses from various perspectives the probable impact of ATM on enterprises.

1. Introduction

In the last 25 years networking technology has improved by many orders of magnitude. Telecommunications networking delivering voice services is now globally pervasive. Satellite communications services span great distances providing entertainment, news and real-time coverage of events in remote places. Such networks, apart from presenting many new business opportunities, have contributed considerably to widening the geographical scope of enterprises and the way these do business. On a smaller geographical scale, local area networks have provided undreamed of bandwidths to enterprises for meeting their in-house business network requirements. And, of course, the combination of desktop workstations or personal computers with local area networking has changed the fundamental information systems paradigm.

Ironically global data networking has lagged behind advances in other areas of networking. Although it is feasible to set up wide area data networks covering many continents, the costs are unattractive and end-to-end performance characteristics and quality of service are sometimes indeterminate. There is a growing demand to improve services.

A combination of market pull and technology push is resulting in pressure for evolution and rationalisation of existing services and for the introduction of new services. Multimedia networking is a key driver here with its demands for interaction at will and at a low cost with video, image, sound, voice and data servers, which may be globally remote. Requirements for enhanced telecommunications services such as video telephony, video conferencing and video electronic mail are emerging while the home entertainment market for on-line videos and games on demand is growing and appears insatiable.

Most wide area networking technology already installed will not be able to meet the stringent requirements placed on the networking infrastructure by these services. Thus it is time to consider a new approach to global networking, called Asynchronous Transfer Mode. But in what way is it new and how does it meet the above needs? This paper attempts to answer these questions.

2. The Current Networking Dilemma

Networking technologies are normally classified in three categories as shown in figure 1:

- Local Area Networks (LANs)
- Metropolitan Area Networks (MANs)
- Wide Area Networks (WANs) .

Each category is associated with a geographical spread, the distance over which the technology can feasibly operate, and a range of communications bandwidths. However, the categories and the application of the technologies both overlap. LANs tend to be owned and managed by the enterprises they support and are often referred to as private networks. MANs and WANs on the other hand may be private, or may be public in the sense that they are supplied and managed by national or international networking service providers (public voice/data network operators).

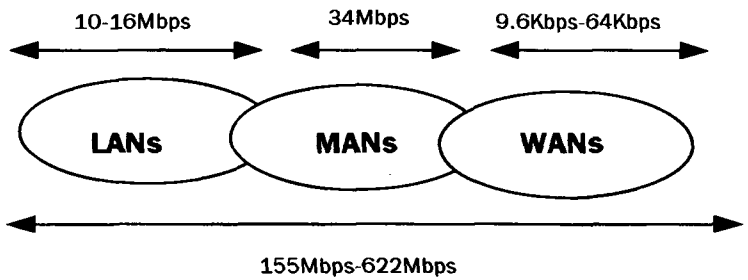


Figure 1 Positioning of networking technologies

2.1 Local area networks

LANs provide networking over distances of up to 3 kilometres delivering bandwidths, shared between the devices on the LANs, of between 4 Mbits/sec and 16 Mbits/sec. Geographical coverage can be increased by means of bridges or routers linked to backbone transmission technology such as Fibre Distributed Data Interface (FDDI) [Taylor, 1992], MANs or WANs [Flatman et al, 1992]. An important fundamental distinction between various LAN technologies is the method of accessing the transmission medium. The contention method permits a LAN attached device to attempt to transmit whenever it wishes. If no other attached device is active, transmission proceeds; however if another device is active the contender must back off and try again. This approach has the advantage of gaining instant access to the LAN when its utilisation is low and the disadvantage of being unable to access the transmission medium quickly when traffic volumes are high. In theory, a LAN device could be unable to communicate for an indeterminate period of time. An alternative approach which guarantees each LAN device a chance to use the transmission medium at predetermined time periods, addresses the indeterminacy of the contention method but has the disadvantage that a LAN attached device must wait a fixed period even if traffic volume is low or non-existent. Some improvements have been made to these two basic approaches to alleviate their weaknesses; however, careful sizing of LANs can improve availability and usability. Popular technologies in this LAN category are CSMA/CD (Ethernet) and Token Ring.

In general, the design objectives of LANs were to provide relatively high bandwidth communications for transferring information between information technology platforms or peripherals attached to a LAN. Voice messaging can also be supported as can limited image applications. However neither of the popular technologies can support real-time voice communications because of the lack of isochronous channels. Also contention access methods preclude support of real-time voice conversations. Simple image transfer can be facilitated but transfers of large image files, which are bandwidth constrained anyway, may lock out other users. Apart from the most trivial applications, multimedia cannot be feasibly networked over current LAN technologies. New developments in this area are focused on increasing the LAN bandwidth to 100 Mbits/sec and exploiting an innovative combination of LAN and switching techniques to ease contention problems and improve throughput. Clearly this will overcome some of the communication bottlenecks; however, further work will be necessary to provide a means for full multimedia support.

2.2 Metropolitan area networks

Proprietary MANs of course have been in use for some time servicing the cable television marketplace and in some cases providing telephony facilities. Standardised communications MANs were designed to cover distances of up to about 100 kilometres offering bandwidths between 2 Mbits/sec and 140 Mbits/sec. These are based on a technique called

Distributed Queue Dual Bus (DQDB) which accommodates unequivocal access to the transmission medium with low latency and is capable of supporting isochronous traffic. MAN products based on DQDB are QPSX and Switched Multi-megabit Data Service (SMDS) which provide bandwidths in the range 2 Mbits/sec to 34 Mbits/sec. Although based on the same underlying principles, these products are not compatible with each other. QPSX sustains voice, video conferencing and data communications. SMDS at present supports only data transfer.

MANs have two main roles: as private networking backbones and as high bandwidth metropolitan-wide, in terms of geographical coverage, switching facilities with access to service providers' WANs (which may in turn link multiple MANs).

2.3 Wide area network

Data WANs potentially have a global span though historically access speeds have been low, typically varying between 9.6 Kbits/sec and 64 Kbits/sec for packet switched networks. Dedicated leased lines can be obtained from service providers for private use which can deliver much higher bandwidths, for example, 2 Mbits/sec. Traditionally service providers, like private enterprises, have operated different networks to support different traffic types. With the advent and deployment of Narrowband Integrated Services Digital Network (N-ISDN) technology [Fuller, 1991] it became possible to use the same networking technology to support voice, slow-scan video conferencing and data applications. As well as updating and rationalising the service providers networks, N-ISDN provided enterprises with the opportunity to converge their networks. However, high data rates were still required. Where available, MANs offered a solution and, with the introduction of Frame Relay, a fast data packet switching technology, bit rates up to 2 Mbits/sec were achievable.

Notwithstanding the continuing need to meet the demands of its voice and data customers, service providers are moving increasingly towards satisfying the home market for on-line entertainment in direct competition with established cable and satellite suppliers. Existing networking technology is being exploited in new and different ways to address this potentially huge market.

Although a rich set of services is available in some areas or countries, the mix often varies: what is popular and dominant in one area may be unavailable or expensive in another. Thus achieving a required end-to-end quality of service on a global network connection is difficult if not impossible.

2.4 What can be done?

From the above discussion it is clear that no single technology is appropriate for delivering a full range of multimedia services that are globally accessible. Although some LANs and MANs are potentially capable of supporting suitable bandwidths, their current transmission techniques inhibit the delivery of the wide range of services essential for

comprehensive multimedia applications. On the other hand most WANs are today not capable of offering very high bandwidths to the end user and a global quality of service is not guaranteed.

So what can be done? There are two fundamental requirements. What is needed first is a globally available WAN based on high bandwidth transmission technology offering high bit rates, comparable with LANs/MANs, and which is capable of sustaining the bandwidth demands of a mixture of traffic types such as high definition video pictures, image, sound, voice and data. And second, transmission techniques which will ensure that the different traffic types receive the necessary quality of service consistently from the underlying transmission medium. Ideally this mechanism should be defined to be independent of any particular transmission technology, though mappings will need to be provided for specific technologies, to enable the mechanism to operate over the geographical areas now covered by LANs, MANs and WANs.

Broadband synchronous digital transmission is a response to the first requirement and ATM meets the second.

3. Broadband Synchronous Digital Transmission

Developments in service provider WAN transmission techniques have tended to focus on satisfying voice networking requirements [CCTA, 1994] characterised by point-to-point connections (logical or physical) and relatively low bandwidths. Although many service providers have installed high bandwidth optical fibre based transmission technologies, these were not exploited efficiently by the transmission techniques in place to handle voice networking. This, and the drive for higher bandwidths, led to work in the International Telecommunications Union - Telecommunications (ITU-T), a formal standards body, on an improved set of transmission techniques called Synchronous Digital Hierarchy (SDH). SDH is based on some seminal developments carried out in the United States on digital transmission techniques called Synchronous Optical Network (SONET) [Stevenson et al, 1991]. The SDH definition which most probably will be adopted in Europe has diverged from the SONET specification, which will be taken up in the United States and Japan; however, there remains enough compatibility to support interworking between the two sets of transmission techniques.

In essence, SDH provides very efficient digital multiplexing and demultiplexing, and fast cross-connect switching to support a hierarchy of transmission rates between 155.52 Mbits/sec and 2488.32 Mbits/sec. SONET also accommodates a rate of 51.84 Mbits/sec though implementations of both sets of techniques can handle lower rates, for example 2 Mbits/sec, via multiplexors. And as the technology improves, greater data rates will be added to the top end of the digital hierarchies. Typical maximum bit rates required by various services are 64 Kbits/sec for voice, 384 Kbits/sec per bit-mapped image, 700 Kbits/sec for hi-fi music, 70-140 Mbits/sec for domestic and studio quality video and 700

Mbits/sec plus for digital High Definition Television [CCTA, 1994]. Compression techniques will lessen these requirements.

Clearly SDH bandwidths have the capacity to sustain numerous traffic types and future multimedia applications. However the different traffic types will each need a different quality of service. For example, video and voice applications require a continuous bit rate with no delays and are not sensitive to some data loss. However, traditional data communications is bursty in nature, is not sensitive to some delay, within agreed bounds, but in general cannot tolerate data loss,while interactive multimedia, which will support various traffic types intermittently, will require bandwidth on demand. ATM was developed by the ITU-T to address these needs.

4. The Origins Of Asynchronous Transfer Mode

4.1 Time-division multiplexing

WAN service providers have continually aimed to use their transmission technology effectively. Early telephony required a dedicated communications link between the communicating parties. Growth in telephone usage led to the development of such techniques as frequency-division multiplexing (FDM) which enabled 24 voice channels (or multiples of) to be multiplexed onto a single communications link. The migration from analogue to digital transmission techniques presented the opportunity to apply a more efficient way of multiplexing multiple communications channels onto a single communications link. It is called Digital Time-Division Multiplexing (TDM) [Davies et al, 1975].

Although there are variants, TDM is essentially a multiplexing method in which a group of communications channels, typically 24 or 30, operate over a common communications link. Each channel on the link is allocated fixed dedicated time-slots in which it can transfer information. Figure 2 represents 6 time-slots supporting 6 channels, each time-slot carrying a portion of each channel's information (the time-slots are repeated).

The slots do not have addressing included so timing information is transmitted along the link to permit separation of the individual communications channels. The time-slots are present whether or not a channel is in use (active or unallocated) and in cases where the information content, in data terms, is non-existent (unused), for example silences in a voice conversation. So in many cases bandwidth was not used efficiently. There was no advantage in reallocating the spare bandwidth to active voice channels since these were sufficiently catered for by their reserved time-slots. Overall, TDM was ideal for voice traffic in which an apparent end-to-end dedicated connection was provided with a pre-determined quality of service.

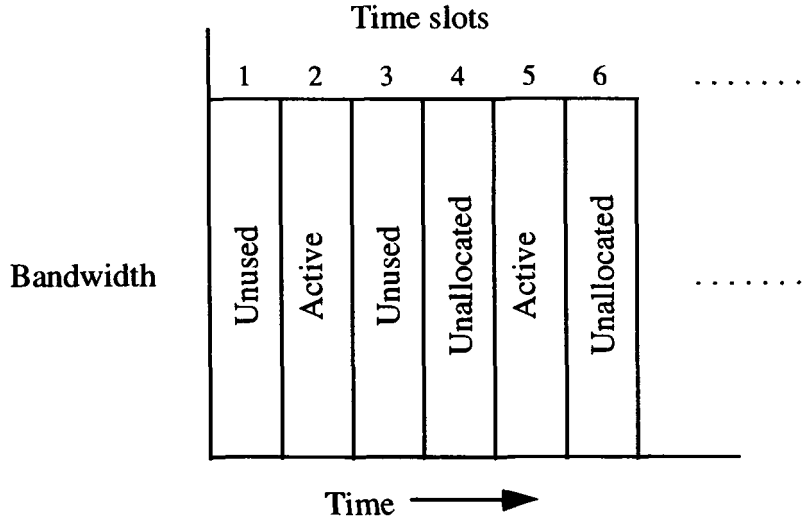


Figure 2 Time-division multiplexing

4.2 Packet switching

The characteristics of Information Technology (IT) data communications differ from those of voice. Specifically IT communication is bursty in nature requiring a large amount of bandwidth for periods which often cannot be determined beforehand. This led in the 1960s to the development of packet switching techniques which effectively superimposed a new set of communication techniques on existing voice networks, so concealing but exploiting the underlying transmission mechanisms.

The principles behind packet switching are well understood and widely applied. A stream of information which is to be communicated between a source and a destination is chopped into units called packets. Some control information is added, for instance to detect transmission errors and help reassembly of the information stream, and addressing details are also included which determine the source and destination of the information. Packets are sent over existing communication links to packet switches which use the packet addresses to deliver the information to the destination or to pass the packet to another packet switch on the way to the destination. The advantages are obvious: many packets from different sources to different destinations can use the same communications link. For example, where TDM is in operation, packets for different destinations can occupy time-slots for the same communications channel and indeed packets for the same destination may use different channel time-slots, see figure 3. Thus, assuming the right balance was struck, the overall bandwidth capacity of the communications links could be used very efficiently.

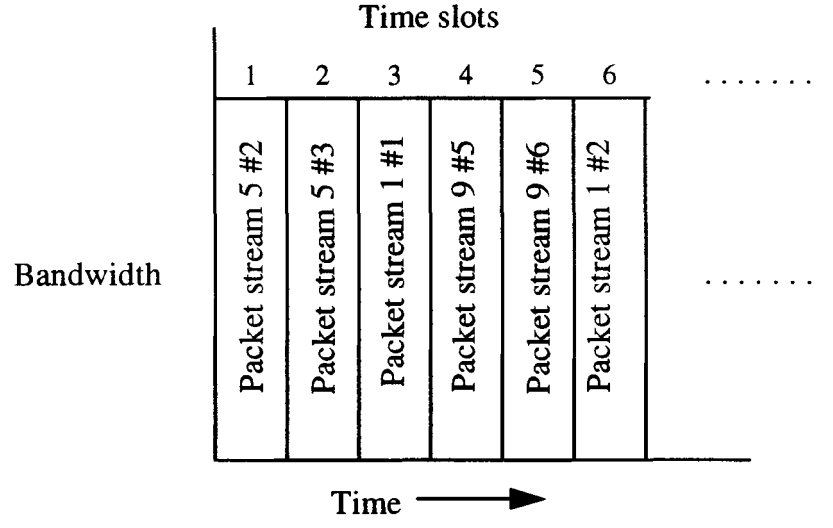
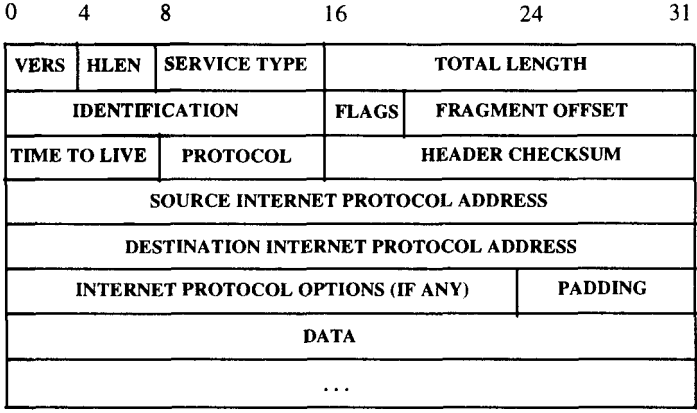


Figure 3 Packet utilisation of TDM time-slots

All IT vendors were attracted to the principles of packet switching and service providers, responding to the market, offered public packet switching facilities. In the early days of packet switching there was great debate about packet sizes. IT vendors wanted a variety of sizes which would efficiently handle variable quanta of data to support single character interactions to large file transfers where large packets proved more appropriate. There was a general preference among IT vendors for large packet sizes. Packet size was ultimately determined by the quality of existing transmission technology (for example, long packets were prone to errors) and the design considerations in producing packet switches (for example, processing overhead and buffering). A packet size of 255 characters was popular initially but improvements in the technology extended the packet size range in both directions. Many different, incompatible packet structures emerged together with distinct networking approaches. The emergence of the Open Systems Interconnection (OSI) Reference Model [Houldsworth, 1992] provided a basis for discussing and standardising packet techniques.

4.3 Connectionless and connection-mode operation

The so called connectionless-mode or datagram operation was based on packets each of which contain complete source and destination addressing, see figure 4, to enable each packet to reach its destination independently of all others and via different routes. Packets could be dropped en route and arrive out of source transmission order though there were run-time mechanisms to detect and recover from these occurrences.

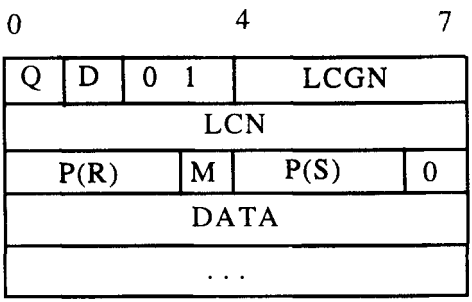


[Comer, 1991]

Figure 4 Internet protocol datagram

The main advantages of *datagram* operation are that bursty transmission could be accommodated (or at least attempted), alternative routing around broken network components was possible and there was no need to reserve substantial costly networking resources between the source and destination. Drawbacks included non-deterministic behaviour and the overhead of always carrying a full set of addresses (consider the case where the actual payload is 1 or 2 octets long). The popular internet protocol (IP) is an example of this mode of operation.

The connectionless nature of datagram operation was not acceptable to some vendors and particularly the service providers who, as a rule, offered guaranteed connections with a defined quality of service. The generic term for this approach is connection-mode and its realisation is a blend of point-to-point channel theory with packet switching principles. The *connection-mode* approach establishes a so called virtual circuit between the source and destination by means of a call set-up packet which contains full source and destination addressing. Resources are reserved along the communications path in all the intervening switches to handle the projected virtual circuit's traffic. Subsequent data packets contain a short virtual circuit identifier (see figure 5), thus reducing overhead, and are always routed to their destination along the same path through the network. A call close packet breaks the virtual circuit and releases the resources. Connection-mode provides fixed bandwidth guarantees and greater levels of security than connectionless-mode. It also makes good use of real communications links, since packets from various virtual circuits can be interleaved. Some drawbacks are that it reserves resources which may not be fully used at all times and, in general, it is not transparent to packet switch breakdown where dynamic alternative routing is needed. An example of connection-mode operation is the X.25 access protocol.



LCGN: Logical Channel Group Number
LCN: Logical Channel Number
P(R), P(S): Flow control

Figure 5 An X.25 data packet

To an extent, the emergence of packet switching was the IT industry imposing its will on the service providers whose main business was the provision of voice networking. Recent deployment of N-ISDN by the service providers, though marketed as a technology with widespread applications, effectively offered a switched circuit service which exploited and made more efficient use of underlying TDM techniques [Gould, 1994]. Although two 64 Kbits/sec channels were available, each of which could support separate voice or data channels, or a mixture of both, its main application was voice networking. Over twenty years ago when N-ISDN work began, two 64 Kbits/sec seemed more than adequate to its developers to satisfy envisaged data needs.

4.4 The move to higher bandwidth

It became clear in the 1980s that future IT and other markets would require access to very high bandwidths indeed. The existing transmission techniques, such as TDM, with their point-to-point connection philosophy and multiple sources of clocking were leading to ponderous complexity and overheads in providing networking services without potential for change to accommodate diverse bandwidth demands. SDH [Sakai, 1992] was developed to address these issues. SDH is compatible with legacy transmission techniques and uses existing transmission media. All equipment in a SDH network operates from a single network-wide clock removing the need to carry large amounts of information for delineating the time-slots. With SDH, multiplexing and demultiplexing of communications channels is considerably easier and additional management functions give service providers scope for excellent systems management to maximise availability and performance. In summary, the move to SDM by service providers for their internal network operation is driven by the need to provide the networking infrastructure to meet the foreseen market requirement for high availability, with unparalleled high bandwidths. SDH is inherently circuit based and, for example, is ideally

suitable for delivering high definition video pictures which would require a bandwidth of about 140 Mbits/sec. But the problem is how to avoid locking up precious network resources to cater for the needs of devices that require variable amounts of bandwidths, sometimes changing instantly, with lulls in between. Obviously packet switching of some form offers a solution.

4.5 The need for a new approach

From our discussion on packet switching above it is clear that packet usage of the underlying synchronous transmission infrastructure essentially imposes an asynchronous mode of operation on it, in the sense that packets can be generated at any time and use any, and many, of the different underlying channel time-slots. We have also seen that it is possible to set up virtual circuits using packet techniques which, although requiring more network resource than datagrams, offer greater utilisation than dedicated circuits. The packet switching process took place in network switches where packets were demultiplexed from communications links and placed on appropriate multiplexing queues for onward transmission. Thus delays were introduced. Further, early packet technique assumed unreliable transmission media and employed extensive mechanisms to detect packet loss and data corruption, and had built in recovery techniques. Packet checking and recovery took place in switches or receiving systems or both. The net result was that packet handling was complex, normally carried out by software, and time consuming.

The advent of highly reliable transmission media, such as optical fibre, provided the opportunity to re-evaluate packet switching principles. Most error correcting overhead was deemed unnecessary and although some loss and errors could still occur, responsibility for recovery was passed to the attached network device rather than handled in the network switches. Other aspects were also rethought, for example, accountability for data flow control was, in general, passed back to the data source. This resulted in much simplified packet protocol and switching techniques, called Fast Packet Switching, which could be implemented in chip technology.

Telephony research in the United States in the 1960s [Gould, 1994] and more recent work in CNET, France in the 1980s [Stevenson et al, 1991] adopted packet switching principles when investigating ways of using TDM more efficiently. Fixed length packets, called cells, were generated from channel activity and dropped with addressing information into TDM time-slots asynchronously. It was not necessary to use a pre-determined time-slot, any one could be used and the channels were reconstituted using the cell addressing information. The technique was originally named asynchronous time-division multiplexing. The technique, together with the Fast Packet Switching principles, was adopted by the ITU-T for its broadband ISDN (B-ISDN) standardisation work, to be built on SDH, and renamed Asynchronous Transfer Mode (ATM).

5. What is Asynchronous Transfer Mode?

5.1 The ATM building brick and what it must do

ATM may be viewed as an overlay network on SDH which provides the benefits of both packet switching and apparent point-to-point dedicated communications channels thus offering the potential to deliver a wide range of services. Fundamental to ATM is the cell.

Fixed length packets or cells are attractive from a implementation viewpoint. They enable the development of less complex chip technology and the deployment of very fast hardware switching. The ITU-T decided on a cell size of 53 octets for ATM of which 48 are payload (data) and 5 constitute the cell header. Using this simple building brick a variety of services of different qualities need to be supported by an ATM network. For example, high definition moving video requires continuous high bandwidth with minimum latency and real time transmission. On the other hand IT traffic, text or static image transfers tend to be bursty in nature requiring chunks of high bandwidth with low latency. Also for continuous media, the loss of some information may not be important so error detection and recovery may be different from that employed in recovering from errors in non-continuous transmissions. In other words, an ATM network must be capable of supporting Continuous Bit-rate Services (CBR) and Variable Bit-rate Services (VBR). How does it do this?

5.2 ATM architecture

The physical layer defines physical interfaces and framing protocols for accessing and mapping onto the underlying transmission technology, for example, SONET or SDH communication links operating at 45 Mbits/sec, 155 Mbits/sec and 622 Mbits/sec (see figure 6). Other mappings, as indeed other transmission technologies, are possible. The physical layer generates the cell Header Error Control (HEC), and detects and corrects, if possible, header errors (see below). This layer also inserts or removes *idle* cells, identified by a unique header code, which effectively adapt the rate of the data source to that of the transmission technology. Idle cells can be discarded at various parts of the switching hierarchy or used for valid data.

The ATM layer defines the cell structure and how cells flow over virtual connections in the network. It is independent of the actual services offered to the users.

The adaptation layer supports the techniques to provide the multiple quality of services required by the various traffic types. It is service dependent.

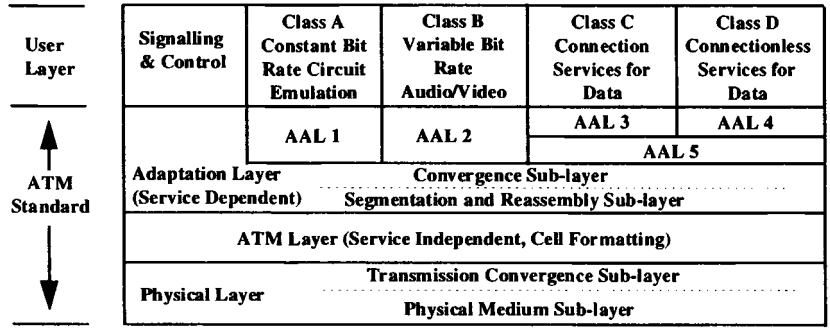


Figure 6 Outline ATM architecture

The user layer, and there are likely to be many sublayers within this layer, will deliver voice, video, audio and data services to applications.

5.3 The ATM layer

The ATM layer is independent of upper layer services offered. It makes use of and refines some of the packet virtual circuit concepts introduced earlier. See figure 7.

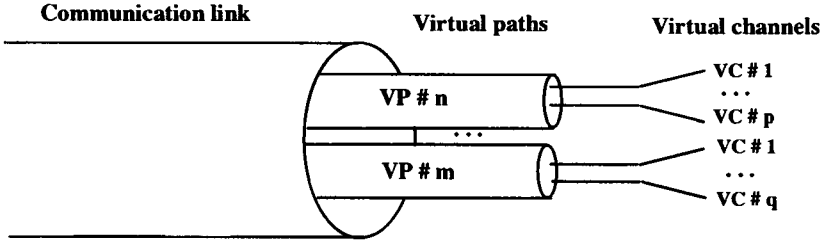


Figure 7 Virtual paths and virtual channels

Sources and destinations communicate via Virtual Channels (VCs) [Houldsworth et al, 1991]. VCs have a set-up, data transfer and close down phases. Set-up cells carry full addressing information and such information as a required quality of service and a requested amount of bandwidth (determined by the higher layers). Acceptance of the VC set-up reserves bandwidth and switch resources throughout the network thus guaranteeing the requested quality of service. If resources are unavailable the VC is rejected. Once the VC is established, data can be transferred from the source using a VC identifier to label VC traffic uniquely originating from a source and destined for a specific destination. Effectively a VC is a continuous or discontinuous stream of cells each with the same VC identifier carried in the cell header. Cells from different VCs can be interleaved randomly on communications links at the point of entry to the network and throughout the network itself. VC

set-up and close down are likely to be extremely fast thus allowing a network device rapid access to various services and bandwidths.

A virtual path (VP) is a collection of VCs which for routing purposes share the same virtual path identifier. This two level structure enables sets of virtual channels associated with a VP destined for the same switch or network device to be routed collectively without unpacking the constituent VCs. Of course in situations where VCs need to be routed in various directions the VCs are unpacked from the VP, the destinations determined and the VCs are then associated with new onward VPs. In general, virtual channel identifiers and virtual path identifiers have no end-to-end significance and values can change in the network as routing and switching occurs. However ATM switches have tables, either set up by management functions or as a result of dynamic channel set-up, which maintain relationships between all of the active VP and VC identifiers (hence the name virtual). Figure 8 shows a simple example of this in operation. In this incoming cells on VC #2, which is part of VP #1, are unpacked and routed according to the routing table to outgoing VC #1 in VP #3. Note the sequence numbers in brackets are to add clarity to the diagram and are not part of the ATM cell structure.

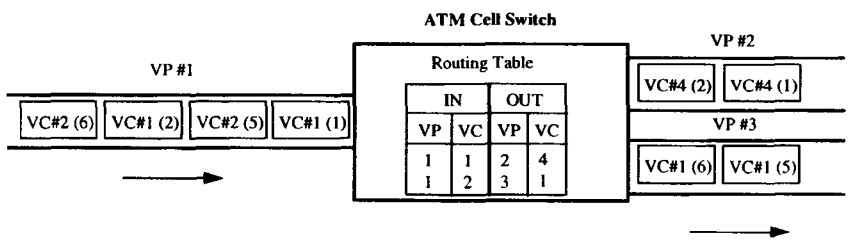
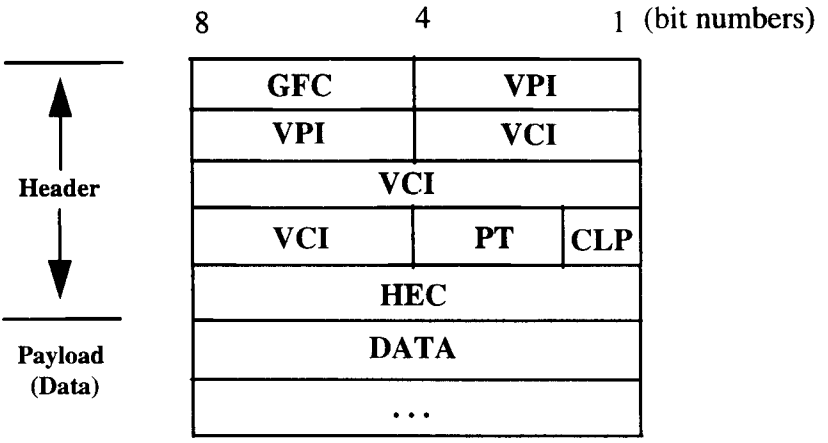


Figure 8 Simple example of cell switching

The ATM layer generates and extracts cell headers. For transmission, it takes the 48 octet payload from the higher layer and produces a cell header with VP and VC identifiers, see figure 9. Note it is not possible to mix data from different sources in a single cell payload. Generic flow control will be used to attempt to control congestion in the network although how this will be achieved is still being investigated. The payload type field indicates whether the payload is network attached device data or internal network data (ATM uses VCs for internal management purposes). The cell loss priority field indicates whether or not resources have been pre-allocated for the cell throughout the network. Cells without agreed resources allocated may be discarded if congestion occurs. It is clearly important that the integrity of cell headers is maintained throughout the network. The header error control is an 8 bit polynomial error check code, covering the first 4 octets of the header, which detects single and multiple bit corruption, and can correct single bit errors. It is generated and validated in the physical layer. Note the payload is not

error checked by ATM, this is left to the higher layers where an appropriate recovery strategy is expected to be implemented.



GFC: General Flow Control
VCI: Virtual Channel Identifier
CLP: Cell Loss Priority
VPI: Virtual Path Identifier
PT: Payload Type
HEC: Header Error Control

Figure 9 An ATM data cell

5.4 The adaptation layer

The ATM adaptation layer (AAL) provides the means to support multiple types of services with different attributes. It provides four classes of services to the user layer:

- Class A supports constant bit rate to handle, for example, voice, high definition video, hi-fi audio and graphical animation. Effectively, connection-mode switched circuits of agreed bandwidth are emulated by means of VCs.
- Class B delivers a variable bit rate for supporting such applications as compressed video and video conferencing.
- Class C provides for variable bit rate connection-mode operation to support, for example, traditional data virtual circuit connections.
- Class D supports a variable bit rate connectionless-mode operation to provide for datagram services.

These user services are achieved in the AAL by four types of adaptation: AAL 1 to 4 (AAL 5 is a more efficient variant of AAL 3/4 for high speed data and internal network use). AAL adaptation types perform convergence functions, which are mappings of the variable length higher layer data blocks or packets into and from ATM cells. This segmentation

and reassembly process, apart from taking time, also has a 1-4 octet associated overhead which reduces the octet payload by that amount.

AAL 1 is characterised [Boerjan et al, 1992] by its connection-mode service, its ability to accept and deliver data at a fixed clock rate and to flag unrecoverable data loss to the upper layer. It is also capable of transferring details of the source information structure to the destination.

AAL 2 shares characteristics with AAL1, however, it offers two data rate options: a rate for normal use and a burst rate for occasional use.

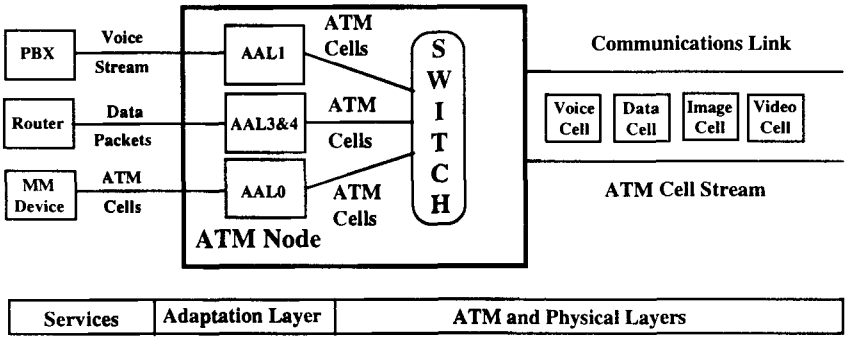
AAL 3 and 4 services have been merged into a single specification referred to as ALL 3/4 which is data oriented and not dependent on timing. Two modes of operation are defined: message and streaming. Both modes offer assured or non-assured delivery. Assured delivery guarantees the data arriving at the destination complete and intact by using flow control, loss and error detection techniques with recovery. With non-assured delivery, data can be lost or corrupted. AAL 3/4 services can be used to support existing packet transfer techniques such as HDLC, Frame Relay, OSI protocols and the internet protocol (IP).

The user layer may wish to make direct use of the ATM layer without any adaptation. This is also possible and is sometimes referred to as AAL 0.

5.5 ATM in practice

As described above, fixed length cells enable fast and deterministic switching, and any bit rate can be achieved by using the required number of cells per second [ATM Forum, 1992]. The number of VCs carrying the cells can vary depending on user needs and, of course, the bit rate of each VC can change in time in response to these needs and the service being supported. It is this flexibility that makes ATM an attractive technology.

A Private Branch Exchange (PBX) or LAN router can interface to an ATM switching node (see figure 10). The voice stream and data packets are segmented into cell payloads, which include segmentation control octets, VP and VC identifiers are included with other header control information, and cells are passed to the appropriate communications links for transmission depending on destinations (one such link is shown in the figure). Multiple voice channels can be supported by means of multiple VCs. And data packets from multiple sources can share the same virtual channel or visa versa, higher level packet addressing can determine routing to the destination. The figure also shows a multimedia device using ATM directly. In this case the adaptation is minimal or null. The adaptation layer at the receiving node performs reassembly into voice channels and packets depending upon the service and passes the information to the attached devices.



[ATM Forum]

Figure 10 An overview of ATM working

5.6 So what is the state of ATM standards?

ATM is an exciting technology, however, much standardisation effort is required before it delivers its promise. ATM standardisation takes place in a number of bodies, two of which are prominent. The ITU-T is the international standards body where formal ATM standards work is progressed. The ATM Forum is a United States based consortium composed of international members including service providers, vendors and users. It has a mission to speed up the development and deployment of ATM products and services, and it works in tandem with the ITU-T. Specifically the ATM Forum attempts to agree on ATM implementation issues which are still under discussion or which for various reasons may not have been clearly ratified by the ITU-T.

Although great energy is going into this work some of the major areas that still need addressing are:

- *Switched virtual channels*: We explained the concept of virtual channels above. VCs may be permanent or switched. A permanent VC for all intents and purposes is the same as a permanent circuit between source and destination: an agreed amount of bandwidth is available all the time. It is expensive and generally not well utilised by the end user. A switched VC is one which is set-up when required between the source and destination and closed down when not needed. ATM networks will potentially have very fast channel set-up and close down attributes which effectively can provide bandwidth on demand. Sadly progress on the standardisation of switched VCs is lagging and it will be some time before implementations of the standards will be available.

- *Congestion control*: An ATM network is fundamentally no different from traditional networks in that, given certain circumstances, the network can become congested with traffic with dramatic degradation in the expected quality of service. Congestion or flow control tries to ensure that a congested state does not occur. Although clearly a key requirement for building acceptable ATM networks, progress in the area was slow in ITU-T. The ATM Forum picked up the work and agreed congestion control techniques for constant bit rate and variable bit rate (AAL 1 and 2) for delay sensitive traffic like video and voice. However, it is having difficulty settling on controls for IT generated bursty traffic. This could take up to another year to ratify and a likely outcome is a number of incompatible techniques.
- *Management and billing*: Much more standards work needs to be done in this area. Without these, true global open ATM operation will not be achievable.
- *Service definitions*: The ATM adaptation layer requires a number of convergence functions to be defined to map the different services, such as video, to ATM. Development of standard convergence functions for some services have been slow emerging and widespread take-up of those in place is not guaranteed. This is a crucial area; without agreement, provision of interoperability between ATM supported services will be unlikely.

The fact that ratified standards do not exist for these areas will not stop vendors developing ATM technology and it being deployed. Obviously in many cases early-to-market vendors will have adopted proprietary solutions to solve the above deficiencies and thus vendor lock-in over the short to medium term is likely.

6. Analysing ATM Using the OPEN*framework* Qualities and Perspectives

Given the attention that ATM is attracting and its potential for deployment in the near future, enterprises need to understand the impact it may have on their businesses, and the opportunities it offers to improve effectiveness and respond to their market demands. One way of doing this is to use the OPEN*framework* qualities and perspectives as a structured approach to ensure all aspects of ATM are considered. The qualities:

- availability
- usability
- performance
- security
- potential for change

provide the means of looking at the attributes of ATM and putting a value on it. While the perspectives:

- enterprise management
- users
- application developers
- (enterprise) service providers

can describe the concerns and impact of ATM on various roles in an enterprise [Brunt et al, 1992] [Hutt, 1994]. In general, applying the OPEN*framework* qualities and perspectives to analyse an enterprise's business, social system and technical system is a very powerful approach for understanding and managing change. However, the approach can be used less formally, as it is here, to provide some focused summary information.

6.1 Availability

One of the attractions of ATM to enterprise management is that all existing networking services can be provided by a single networking technology. This could rationalise the current diverse set of skills employed in supporting various forms of networking. However, there will be some concern about cost, which is likely to be high, certainly in the shorter term, and about interoperability due to the state of standardisation and specific vendor strategies. Also because of the vast expense in putting optical fibre transmission technology in place and installing ATM switches, widespread public ATM networking is unlikely before the next decade.

Users will have access to guaranteed bandwidth on demand supporting a range of real-time services. ATM potentially allows instant access to a range of multimedia sources both inside and outside the enterprise which can be brought together at the users' workstations.

ATM will, in general, be reliable but what will be the impact of minimal error recovery in the network? On the whole, attachment devices like routers will provide error recovery but for multimedia application platforms, for example, which generate ATM cells, packet loss and errors will need to be handled. Vendors supplying such platforms will no doubt address this problem; however, different vendors may have different strategies of which application developers should be aware. The emergence of standardised Application Programming Interfaces (APIs) for ATM will ease development and increase portability options but little work has been done in this area.

The enterprise service provider will initially have to contend with proprietary management systems which will vary from supplier to supplier. Switched virtual channels are likely to be late coming to market so one of the key benefits of ATM will not be attainable initially. Also ATM offerings in different countries will almost certainly be diverse.

6.2 Usability

Enterprise management may find that ATM as an enabling technology together with appropriate applications, for example, multimedia kiosks, supports an *easier to do business with* enterprise strategy.

For most users it is more natural to deal with image, pictures and voice rather than just text or even graphical user interfaces. Seeing who they are dealing with, and being seen, will help users in some business contexts.

The lack of development tools (and APIs) which exploit ATM will concern application developers. Early ATM exploitation may well be a craft industry until de facto standards emerge in this area.

Enterprise service providers currently manage the in-house networks. Therefore integration of ATM management into the enterprise's management infrastructure will be of paramount importance. Initially this may prove challenging. A concern shared with application developers, will be integration with existing applications and what the migration strategy will be.

6.3 Performance

The costs/benefits analysis for ATM may prove difficult to compute. Initially benefits may be troublesome to pin-point and may, indeed, in the shorter term, be intangible. Obvious benefits can include savings on converging the enterprise's various networks.

Users will experience excellent interaction response times from application running over ATM apparently independent of the traffic and geography involved.

Application developers will want to exploit the scalable, bandwidth on demand features presented by ATM. Enterprise management will need to temper this motivation with a sound business strategy.

There will be a lack of tools to analyse ATM performance in early deployment. In some quarters there is some concern about the expected performance of large scale ATM networks in which guarantees of multiple quality of services is considered to be difficult to sustain. The lateness of arrival of congestion methods and their incompatibility with existing LAN mechanisms for data does not create confidence in the shorter term. And although, as we have seen above, cell switching has many advantages, for traditional packet switching applications the control octet overhead can be almost 20%.

6.4 Security

Although there are many questions still to be answered about ATM, it is clear that it, or at least the principles, will be important. Thus, though a cautious approach is recommended, moving towards ATM in the medium to longer term will benefit the enterprise network and support business prosperity.

End-to-end user security is aided by virtual channels and closed user groups which restrict communication to that between designated network devices.

Current application security techniques will be reusable in an ATM networking environment.

Control over access to expensive ATM bandwidth options will be required. Also policing of actual bandwidth usage is necessary to ensure agreed limits are not broken. Also, for some services, bandwidth will be negotiated at channel set-up time. If the network, or combinations of all the networks involved in supporting the virtual channel, has not the resources to meet the request, access can be denied.

6.5 Potential for change

From a networking point of view, ATM will maximise potential for change providing the vehicle for accommodating any sort of traffic that may be required to support a business.

New multimedia applications will be introduced painlessly without changing wiring, terminals or interface connectors.

For application developers, the aim will be to have a minimum number of APIs which can access the different services support facilities.

Bandwidth on demand when it arrives will ease the service providers' task of forever changing service level agreements with parties in-house and with external network providers as demands fluctuate and grow.

7. References

ATM FORUM, "ATM: The Future of Networking". McGraw-Hill Data Communications pp124-128 October 1992

BOERJAN, J., CAMPBELL, A., COULSON, G., GARCIA, F., HUTCHISON, D., LEOPOLD, H., SINGER, N. "The OSI 95 Transport Service and the New Environment". Lancaster University, ESPRIT Project 5341/Sector OBS. 1992

BRUNT, R., HUTT, A. "*OPENframework* The Systems Architecture: an introduction". Prentice Hall 1992. ISBN 0-13-560186-X.

CCTA, "Towards 2000: High Speed Networking Guide for UK Government Departments". HMSO 1994.

COMER, D.E. "Interworking with TCP/IP". Prentice-Hall International, Inc 1991. ISBN 0-13-474321-0.

DAVIES, D.W., BARBER, D.L.A. "Communication Networks for Computers". John Wiley & Sons 1975. ISBN 0 471 19874 9.

FLATMAN, A.V., LAWE, J., RUSSELL, B. "Infrastructure of Corporate Networks in the Nineties". *ICL Tech J.* Vol. 8 (2) pp198-209, 1992.

FULLER, A.R. "Future Applications of ISDN to Information Technology". *ICL Tech J.* Vol. 7 (3) pp501-511, 1991.

GOULD, J. "ATM's Long, Strange Trip to the Mainstream". McGraw-Hill Data Communications pp120-130 June 1994

HOULDSWORTH, J. "Open Networks - The Key to Global Success" *ICL Tech J.* Vol. 8 (2) pp179-197, 1992.

HOULDSWORTH, J., TAYLOR, M., CAVES, K., FLATMAN, A.V., CROOK, K. "Open System LANs and their Global Interconnection". Butterworth Heinemann 1991. ISBN 0 7506 1045 X.

HUTT, A. "*Transforming your business with information technology*" To be published in 1994.

SAKAI, T., TOKO, Y., TOKIMASA, A. "Synchronous Digital Network Systems". *FUJITSU Sci. Tech. J.* Vol. 28 (2) pp161-171, 1992

STEVENSON, I., TIMMS, S. "Broadband Communications: Market Strategies". Ovum 1991. ISBN 0 903969 63 7.

TAYLOR, M. "FDDI - The High Speed Network for the Nineties". *ICL Tech J.* Vol. 8 (2) pp225-241, 1992

8. Biography

Frank Deignan

Frank Deignan has more than twenty years experience in the IT industry. He is currently the OPEN*framework* Networking Services architect in ICL. Prior to this role he was the ICL messaging strategy manager and OSI standards manager.

He has a MSc from Manchester University, is a Chartered Engineer (CEng), a Member of the British Computer Society (MBCS) and an Associate Fellow of the Institute of Mathematics and its Applications.

The ICL search *accelerator* ™, SCAFS ™: functionality and benefits

M.W. Martin

Server Systems Division, ICL, Bracknell, UK

Abstract

This document describes the system and functional capabilities of the **search** *accelerator* technologies, on which ICL's Database **search** *accelerator* programme is built.

The performance of applications can be accelerated by shipping application functions closely associated with input/output data from the host processor into the peripheral sub-system. This can relieve the host of a very considerable processing and I/O overhead and make better use of host caches, in addition to providing faster responses.

The performance improvements possible and realised in practice are discussed, both qualitatively and quantitatively. To illustrate use of the foundation technologies, some examples of Relational Database acceleration are included.

1. Introduction

The Son of CAFS (SCAFS) programme began as an investigation into the exploitation opportunities for CAFS-like technology in the open systems market (CAFS ™ is a search engine sold on ICL mainframe systems since the early 1980s [ICL, 1985]). Attention quickly focused on accelerating Relational Database performance in the dimensions of Decision Support, Management Information, ad hoc queries, and fuzzy text access, by executing the associated searches within a SCAFS engine in the disk sub-system.

Given past reaction to proprietary technologies in the open system market, it was immediately recognized that:

- SCAFS would need to be invisible at the application level (i.e. hidden below SQL)
- the objective must be to establish SCAFS as the de-facto industry standard solution. Thus products must be designed as a low-cost, high-volume, world market solution.

Extensive market and technology research was undertaken. Views were sought from, and proposals put forward to, experts across the industry, including; ICL marketeers and technologists, key architects from the leading relational database vendors (INGRES ™, ORACLE ™, INFORMIX ®, SYBASE), other computer vendors, market research agencies (Gartner, Dataquest), and leading US and UK Venture Capitalists.

Three main conclusions arose from the investigation:

1) That accelerating relational database was the key market to exploit initially

2) Plans for the availability of the technology across a wide range of computer platforms was a prerequisite to relational vendors' collaboration

3) In the open systems marketplace, a radically different CAFS architecture was needed to truly hide CAFS effects from users. In particular, SCAFS should search the different data formats used by each of the relational database vendors in their standard unchanged forms (unlike ICL's mainframe CAFS).

The last had proved impossible in the past. However, following research work within ICL, a radically different architecture, SCAFS, was invented that satisfies the above requirement, but also has scope for much wider exploitation. Patents have been applied for.

2. Outline Description

2.1 Generic components

A system exploiting SCAFS as an application accelerator is expected to consist of five main parts (as shown in the numbered brackets in figure 1):

(1) Existing applications built upon industry standard, or de-facto standard, interfaces (e.g. SQL). Applications, and their databases remain unchanged. SCAFS is invisible to the end user other than for performance benefits.

(2) An application support environment (e.g. relational database back-end. This needs extending to provide a platform independent library interface for the function being shipped (e.g. databases searching).

(3) Host based Accelerator library software which is called via the new library interface and which in turn communicates with *personality* software running on the SCAFS unit.

(4) Code downloaded to SCAFS to personalize it for a specific application function being shipped, called a personality.

(5) Generic SCAFS system components (hardware and software, that support the operation of Accelerator software and personality code.

2.2 Component features

The enhanced RDBMS back-end, with the new (search accelerator) library interface is expected to be the standard version shipped on all platforms, regardless of whether the **search** *accelerator* is present or not, as is already the case with INFORMIX OnLine and INGRES and will be with ORACLE during 1994.

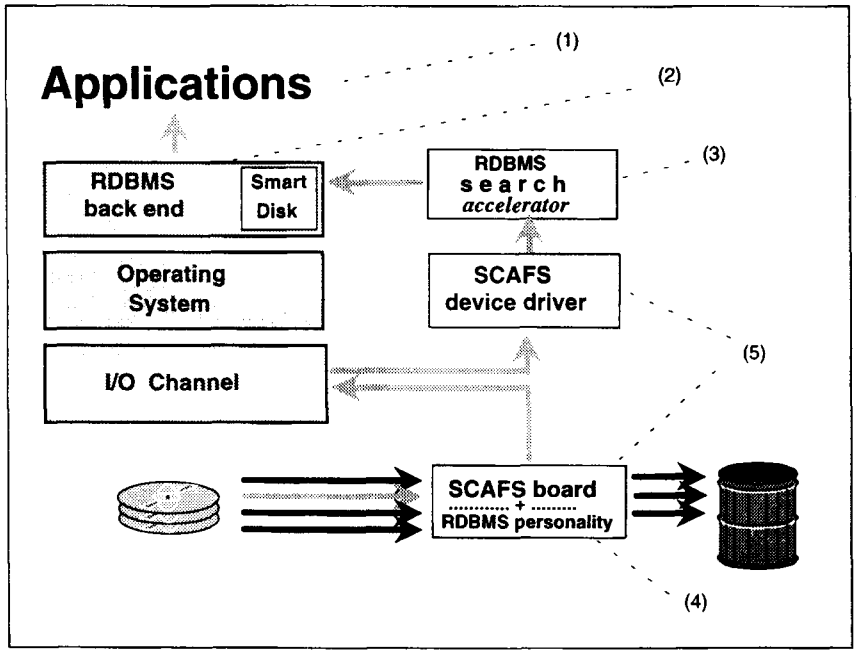


Figure 1 System Overview of RDBMS **search** *accelerator*

A call across the new library interface can determine the **search** *accelerator*'s presence plus whether it supports the function being shipped. Where support is unavailable, the database back-end executes the function in exactly the same manner as is conventional practice.

The Accelerator software and its associated personality is packaged and sold together as they are specific to the particular class of application being accelerated (e.g. Oracle, Ingres or Informix **search** *accelerator*).

Generic SCAFS system components are viewed as being integrated with the base platform, (hardware and operating system). They include a device driver(s) and plug-in hardware with associated firmware. The components are able to support different types of application function being shipped, and their concurrent operation (see figure 2).

The Generic SCAFS components contain no features particular to any database but should provide support for all of them (with appropriate accelerator software and personality).

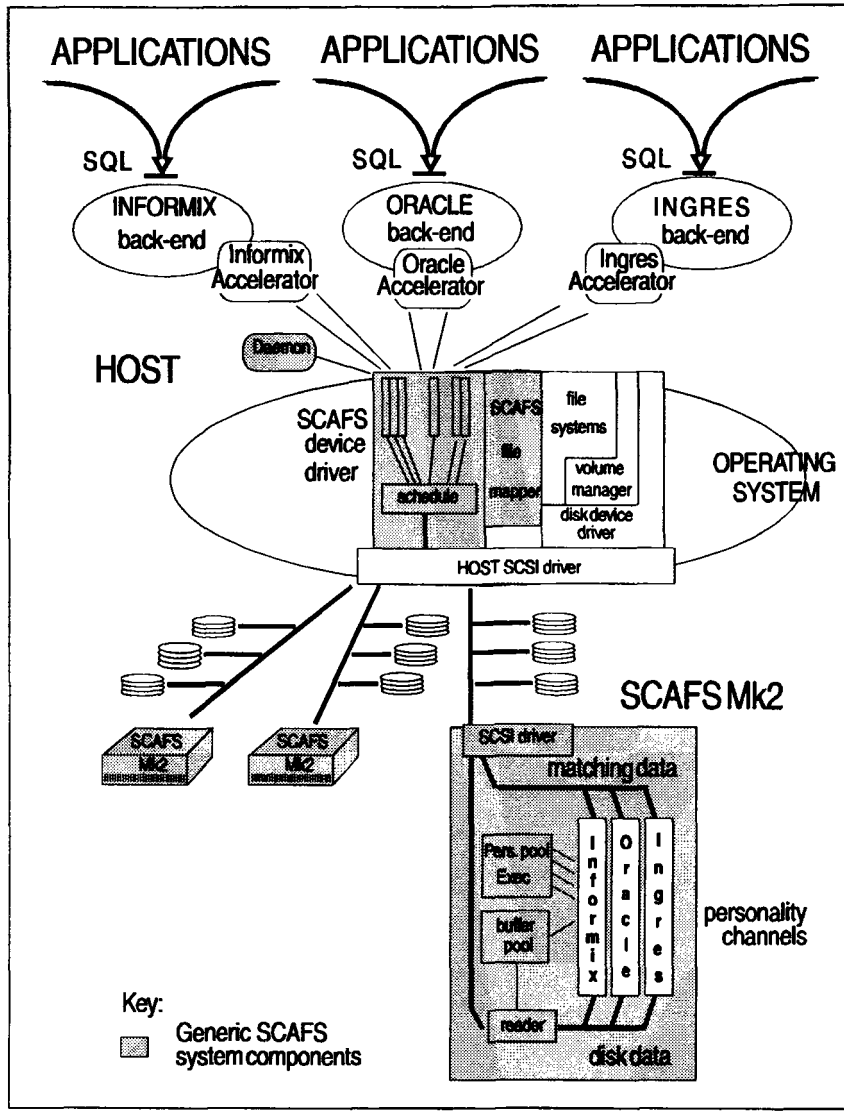


Figure 2 Example system with multiple personalities

The SCAFS device driver provides a generic interface to SCAFS for host based application accelerator software. It is also responsible for scheduling access to personalities from competing accelerator software (e.g. concurrent RDBMS queries). Where necessary, it will temporarily suspend searches that hog system resources (e.g. where the host application consumes retrieved data much slower than SCAFS delivers).

The SCAFS on-board firmware provides an operating environment for personalities and supports communication between the personality and the host accelerator software (via the SCAFS device driver).

The firmware can provide support for multiple down-loadable personalities (see figure 2); for example concurrent support for Informix, Oracle and Ingres **search** ***accelerator.***

Support for up to three active SCAFS channels is also provided within a single SCAFS hardware unit. Channels operate concurrently, and are instances of personality processes. Thus, SCAFS can support concurrent RDBMS query execution within a single SCAFS unit.

The firmware also supports autonomous reading of data from peripherals (e.g. disks) at their maximum data rate. Three independent data streams can be supported simultaneously, with an aggregate data rate up to 6.5-8Mbytes/sec on current systems (mid 1994).

Greater performance gains are possible by adding multiple SCAFS units to a system; one per string of disk drives (see figures 3 and 4). For example, a system with 6 SCAFS units can execute simultaneously, 18 different and independent databases queries, each operating on a different disk drive.

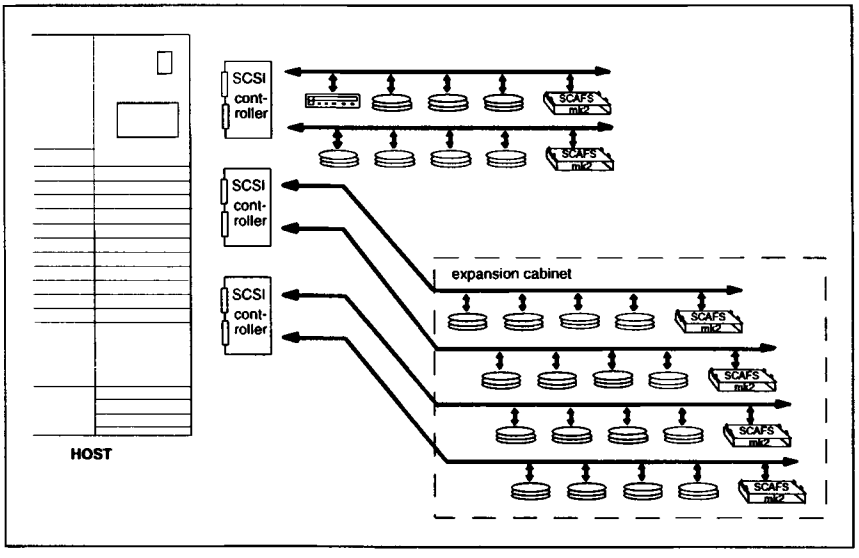


Figure 3 Example Configuration

2.3 RDBMS search *accelerator*

The area focused on for early exploitation of SCAFS technology is as a relational database search accelerator. The RDBMS vendor enhances their back-end to include search accelerator *hooks* via a platform-independent new library interface, often referred to as a *Smart Disk* interface by the RDBMS vendors.

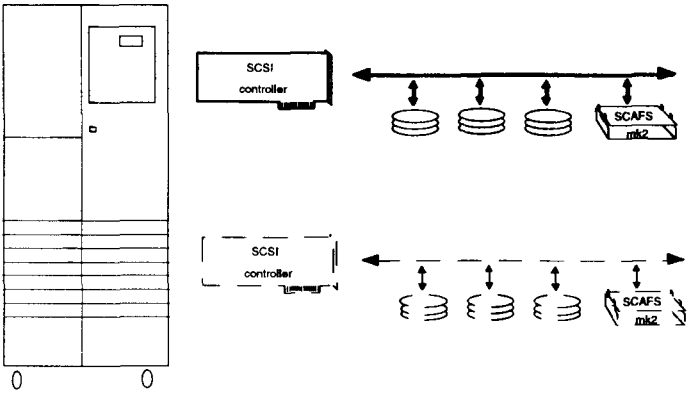


Figure 4 PC Server Configuration

In normal operation a relational database passes an SQL query to some form of query optimiser to determine the best strategy for executing the query. The optimiser decomposes the query into a sequence of related processing operations - selects, projects, sorts, joins - some of which are select operations on single tables. Where a relational database has been enhanced with a Smart Disk interface, the optimiser can choose to invite the Smart Disk to execute these single table select operations. When the Smart Disk is chosen, the select operation is executed autonomously in the disk sub system by SCAFS.

In making its decision, the optimiser first briefly communicates across the Smart Disk interface to ensure the Accelerator is available and able to support the particular select operation on the table. Once the Smart Disk is chosen, further interaction occurs to allow the host Accelerator software to construct a search specification for the select operation. The specification is passed to the RDBMS personality via the SCAFS device driver. The personality, running on the board, performs the specified search on the incoming disk data and only returns matching data back to the host Accelerator software, which in turn passes it to the RDBMS back-end.

Algorithms used within the host and personality ensure the existing RDBMS locking and consistency rules are followed. For example, **search *accelerator*** products fully support:

- all of the Informix four levels of consistency/locking
- the Oracle read consistency model
- the Oracle parallel server.

Great care is taken that the algorithms are provably correct, platform independent, efficient, and have no detrimental effect on TP performance. Novel solutions have been developed for specific RDBMSs. ICL **search**

accelerator software is developed specifically for each relational database to cope with any differences in relational vendors':

- disk storage formats
- data types
- comparison operations
- back-end architectures
- internal data structures
- users' tuning and diagnostic facilities
- data/transaction integrity requirements.

The end result is that operations performed by the **search** *accelerator* always provide an equivalent user response to that of the standard version of the RDBMS. Output may differ, as different ordering of records across the SQL interface may be allowed, though equivalent.

There is no real need for customers to know what the **search** *accelerator* technology really is. To them, adding a database accelerator should be viewed in the same light as adding extra store, or better caching; it just makes the database run faster in certain dimensions.

3. Product Characteristics

3.1 Operational constraints

3.1.1 Current products

SCAFS is available in a 3.5" form factor, or as a plug-in VMEbus or ISA/EISA board. In the plug-in board format only power connections are made to the bus. In the novel 3.5" form, called SCAFS Mk2, it fits in a 3.5" disk bay and has identical connectors and mounting as a 3.5" SCSI disk (patents have been applied for).

SCAFS interfaces to the host and input/output devices via the industry standard SCSI interface. Eight bit SCSI, compatible with SCSI 1 & 2 hosts and devices, is currently used.

Both synchronous and asynchronous SCSI communication is supported. Both single-ended SCSI and differential SCSI versions are available.

Using novel techniques, (patents have been applied for), SCAFS hardware can execute relational database table scans close to the maximum sustained data rate across the SCSI bus (e.g. up to 8Mbytes/sec).

Disks used with SCAFS must be of the type that ensure error detection/correction is performed before disk data is passed across SCSI (usually the case). Also, they must support multi-initiator access (again usually the case).

The maximum number of streams open to all SCAFS channels within a system is only an operating system limitation (e.g. 256 streams).

3.1.2 Future products

Support for additional RDBMS products, plus ports to additional computer system platforms and operating systems are kept constantly under review.

The initial generic SCAFS system components are exploitable on a range of platforms and with a range of applications. In addition, changes to the generic SCAFS system components, can be expected in order to maintain compatibility with market requirements and exploitation possibilities; decisions depend on business cases.

3.2 Portability

Minimising platform porting costs is a key element to achieving wide exploitation and setting a de-facto standard.

Host code, including device driver and application accelerator software, is portable at the source code level (i.e. C code). Device driver implementations for UNIX® V.4 and IBM AIX exist. Interfacing to SCSI includes support for USL's PDI interface for UNIX V.4 systems. The driver has been structured to support symmetric driver requirements.

Different platforms require some customisation of the device driver, but most of the platform specific code is contained in one software module. Interfacing to different filesystems and volume managers is often platform specific.

Communication between host accelerator software, and its on-board personality, is host independent, thereby giving object code portability for personality code (namely one object code for all systems). Note, that the personality can cope with both *BigEndian* and *LittleEndian* disk data.

Wherever, a VMEbus card, ISA/EISA board, or 3.5" disk can be mounted, SCAFS hardware can usually be added without change.

3.3 Security

3.3.1 User security

Features within generic SCAFS components prevent one user/query gaining access to the results or input associated with another query (maliciously or otherwise).

3.4 Standards

Initial SCAFS products interface to the host and input/output devices via the ANSI standard SCSI interface. Eight bit SCSI, compatible with SCSI 1 and SCSI 2 devices is used. Both synchronous and asynchronous communication are supported.

Initial exploitation is with the standard UNIX V.4 operating system environment. Interfacing to the SCAFS device driver is via UNIX streams.

The normal safety and EMC/RFI certificates specified by ICL, for world-wide sales, have been obtained.

3.4.1 X/Open conformance

Relational Database **search** ***accelerator*** does not affect the X/Open conformance of a system:

- If an RDBMS's SQL were X/Open conformant, it would remain so after the addition of the RDBMS **search** ***accelerator***.
- Currently, search accelerator and SCAFS interfaces are not available to application level code, only to relational database management systems; so there are no application portability issues. If the base platform is X/Open conformant it remains so after the addition of **search** ***accelerator***.

3.5 Hardware configuration and software environment

Current generic SCAFS system components require the following environment:

- accommodation for ISA/EISA bus boards, VMEbus(6U) boards, or 3.5" disks, with their standard voltage supplies
- use of up to 10 watts from the host's power supply (no special cooling requirements are needed)
- SCSI interfacing (e.g. to disks)

The data to be searched may be in a filesystem or a raw disk partition.

3.6 Reliability

SCAFS/RDBMS **search** ***accelerator*** should never provide incorrect information back to an RDBMS and therefore possibly to a user. Prior to customer use, 100% of all hardware errors are expected to be detected at time of manufacture. The target is supported by extensive manufacturing tests held in on-board PROM.

In the field, better than 99.5% of all hardware and configuration errors are expected to be automatically detected at system start-up by a combination of:

- PROM based establishment tests
- UNIX *grope*
- initiating a diagnostic check by the host SCAFS device driver
- communication with, and downloading code to, all SCAFS boards known to be configured in the system.

3.7 Resilience and diagnostics

Using simple RDBMS user commands it is possible to switch the **search** ***accelerator*** on and off on a per user basis. In addition, on/off switching for all RDBMS users is possible. First level diagnosis to identify whether

the RDBMS or **search** ***accelerator*** is at fault is then easily possible. With the Accelerator switched off the RDBMS just reverts to its standard operation and performs table scans by software.

The **search** ***accelerator*** is compatible with highly available systems. In the unlikely event of an Accelerator hardware failure, host accelerator software can detect the fact and cause the RDBMS to revert to conventional host based searching.

Where disk data is autonomously accessed via SCAFS (e.g. RDBMS data), the SCAFS device driver performs the same level of file access authorisation as is provided by the operating system.

To assist error diagnosis, including tele-diagnosis, SCAFS logs all detected errors, including personality errors. All errors contain sufficient information to identify the type of error, its location and the version numbers of associated software (e.g. a personality error should result in a log of:

- error type
- time
- date
- personality type and version
- host accelerator version
- Executive version
- location of SCAFS board and personality channel used).

To assist diagnosis further, facilities are available for a personality or a SCAFS executive to initiate selective on-board memory dumps (e.g. to a UNIX file).

Where a SCAFS system error (e.g. hardware or Executive) prevents reliable continuation, the SCAFS Executive initiates an internal reset from PROM, and signals the fact to the host SCAFS device driver. Considerable thought has gone into terminating any outstanding host SCSI command, to prevent SCSI lockup and therefore possible UNIX crashes with filestore errors. The requirement is no different from that of any other SCSI device.

4. Comparison of CAFS and SCAFS

From an overall system viewpoint CAFS and SCAFS appear to provide similar system functionality (autonomous database searching), but there are significant differences in their internal architecture. Some of the major technical differences of SCAFS are:

- searches disk data in standard RDBMS stored format
- supports a wider range of RDBMS (i.e. ORACLE and INFORMIX OnLine in addition to INGRES)
- the concepts of key channels and Search Evaluation Unit (SEU) disappear
- the restriction to 16 search terms is removed; instead an arbitrary number of boolean combination of search terms is supported
- supports a wider range of SQL 'LIKE' text searching, including variable number of don't care characters at start, within, and at end of matching string (e.g. including *contains*)
- no limit on retrieved data (i.e. dynamic rather than static buffering of retrieved data)
- supports Repeat queries with parameters and database procedures (n.b. INGRES support starts with OpenINGRES 1)
- supports inter-column comparisons (n.b. INGRES support starts with OpenINGRES 1)
- supports almost all combinations of datatypes and operators within a Select statement
- with ORACLE and INFORMIX, supports disk data searches with concurrent updates on the same table, without requiring a table lock and without compromising data/transaction integrity requirements
- hardware platform independent.

5. Performance

Relational Database **search** ***accelerator*** is an application of the SCAFS technology. The Accelerator speeds searches for data when an RDBMS determines that tables need to be searched.

It could, therefore, be expected that the Accelerator would be most efficient in speeding up queries associated with management information, decision support, investigation, text access, report generation and ad hoc access.

ICL has done extensive customer workload trials in order to demonstrate performance improvement of RDBMS systems when equipped with the Relational Database **search** ***accelerator***. In these trials the majority of applications were a mix of *production* and *decision support* workloads. The majority of systems were already highly tuned and generally no changes or conversions were applied to the database structure whatsoever. The results of the customer workload trials were as follows:

- In *mixed mode* workloads (mixed production/decision support) throughput was enhanced by a factor 1.4 to 3.6, whilst response times were on average a factor 1.4 to 20 quicker through using the Accelerator.
- In systems with a predominantly decision support workload, results were even more dramatic: response times were on average a factor 2 to 20 better, while individual transaction response times were improved up to 100 fold.
- Production systems that were optimised by indexing to reduce the need for searching tables, also benefited from the Accelerator. With the Accelerator, users could re-address the balance of adding secondary indexes or allowing table searches. Reducing the number of indexes reduces the database volume and runtime overheads associated with index maintenance, thereby significantly improving transaction performance. Thus databases can be optimised for a production workload, leaving the brute force power of the Accelerator to deal with the decision support type of queries.
- When using the Accelerator, the response time for the vast majority of RDBMS queries is independent of the type and complexity of the query unlike standard RDBMSs. For example complex fuzzy text SQL queries like:

  ```
  job_description LIKE "%MAN%DIR%"
  ```

  take no longer than simple queries like:

  ```
  dept = 105
  ```

- Boolean combinations of such terms to arbitrary levels of complexity are also supported without the normal major deterioration in response time.
- Each Accelerator unit can search multiple disks concurrently so long as the maximum data rate of the SCSI is not exceeded.
- The Accelerator does not dominate the SCSI bus while searching but allows host access to the same or other disks on the same SCSI bus. As the Accelerator is architecturally closer to the disk than the RDBMS, it is able to use disks more efficiently for the same task. The Accelerator reads data in large chunks, thus less time is spent moving heads and missing disk revolutions. When compared to having the table scan carried out by the RDBMS, the result is a reduction in the use of disk resources.

- Searches are performed faster than is possible with high performance host processors, and is as fast as data can be read from disc, e.g 4Mbytes/sec. To achieve high performance all modern host processors must make very effective use of caches (e.g. processor, database, file system). The large volume of data associated with database searches wipes out these caches, leading to their ineffective use (both for searching and other users of the system). With the **search** ***accelerator*** this no longer occurs.

6. Marketing SCAFS

The **search** ***accelerator*** products are supported via the set of hardware and software technologies, called SCAFS. SCAFS incorporates highly optimised input/output capability, and is programmable for different functions. These functions are supported by the SCAFS ability to accept downloaded code at run-time (i.e. the SCAFS personalities already mentioned).

Relational Database integration is invisible to the application writer and end-user. Customers are not prevented from moving their relational applications or data to other open-systems platforms (other than by poorer performance). In particular, SCAFS searches operate on relational data in its standard unchanged format, and the search accelerator capability is hidden below SQL.

A Relational Database can embrace the **search** ***accelerator*** without sacrificing platform portability. The database version shipped on all platforms can include the search accelerator hooks as standard, regardless of whether the Accelerator is available or present. Where the Accelerator is unavailable, the database operates exactly the same as is current practice. Thus, the Accelerator has no negative effect on RAS.

All INGRES platform ports from Version 6.4 onwards include the Accelerator hooks as standard, as does INFORMIX from Version 6.0, and ORACLE is planned to from Version 7.2. Prior to these releases, the hooks have been included in versions on ICL, and some other, platforms from INGRES 6.3, INFORMIX OnLine 5.0, and ORACLE 7.0.16 onwards.

With the introduction of within-query parallelism by the RDBMS vendors, the Accelerator's inherent parallelism allows it to exploit such features. Where a table is spread across multiple disks, a single user's query can then be answered using multiple SCAFS units each operating in parallel on a different disk string, with each SCAFS unit concurrently searching multiple disks on the same string. The very high level of parallelism is comparable to that expected from large multi-processor based host systems. The Accelerator would thereby maintain its performance advantage. Since Accelerator units can be considerably cheaper than modern host processors, substantially better cost/performance is offered.

SCAFS system and functional capabilities were designed for the world-wide open-systems server marketplace (e.g. UNIX) and at a cost level appropriate to volume shipment. Target platforms extend from high performance UNIX servers to PC based servers.

Through its own and Fujitsu sales channels, ICL has marketed the Ingres **search** ***accelerator*** since July 1991, Informix **search** ***accelerator*** since October 1992 and Oracle **search** ***accelerator*** since May 1994. IBM began selling the ICL **search** ***accelerator*** on their RISC System/6000 range in August 1994.

7. Conclusions

A sub-set of the general functional capability is: SCAFS technology supports function shipping of Relational Database searches into the disk sub-system.

The following conclusions may be drawn from the development and marketing experience with Relational Databases:

- that accelerating relational database has been the key market to exploit initially
- using novel techniques (patents applied for), a search engine can be designed that resides in the disk sub-system, and:
  - is hardware platform independent, being as easy to install as a 3.5" SCSI disk drive
  - searches market leading relational databases in their standard unchanged disk storage format
  - searches relational data as fast as it can be delivered from disks (e.g. 4Mbytes/sec)
  - achieves table search speeds largely independent of query complexity
- the full RDBMS data/transaction integrity requirements are supported while searching data autonomously from the host processor and the RDBMS buffer cache
- the ICL **search** ***accelerator*** can be integrated with the leading RDBMS products, yet be hidden below the SQL interface, thereby requiring no change in application code
- on most systems, a range of performances is possible by incremental distribution of SCAFS hardware units around the peripheral sub-system, one per disk drive string

- application of the search accelerator technology in the form of Relational Database **search** ***accelerator***, has been shown to enhance the throughput of an RDBMS in a mixed production/decision support environment up to a factor of 3.6, while speeding up response times by up to 20 fold. Response times in users' dedicated decision support systems on average were seen to improve by a factor up to 20, while individual transaction response times were enhanced up to 100 fold.

Clearly, application function shipping need not be limited to disk data searching.

Function shipping follows the sound engineering principal of putting functionality close to the data it is operating on, rather than having to ship the data to the function. Examples of successful host function shipping, from other areas, are:

- screen manipulation to graphic accelerator cards
- page layout to Postscript printers.

8. Acknowledgements

This work was only possible through committed team effort across ICL. I would particularly like to thank the original development and marketing team for their hard work, and professionalism in turning the gem of an idea into a market reality (A.P.Graham Brown, Dave Crane, Richard Downing, Robin Love, David McQuillan, Michael McKinlay, Bruce Millar, Margaret Smith, Paul Stow and Ken Watts), and senior ICL management for their encouragement and support over several years (Bill O'Riordan, Chris Phoenix, David Dace). I would also like to thank INGRES, INFORMIX and Oracle, for the help and collaboration that was essential to achieving product success.

9. Reference

The *ICL Technical Journal*, Vol. 4 No. 4, 1985 comprised 11 papers related to the history, design and applications of CAFS.

10. Trademarks

11. Biography

Mike W.Martin

Mike Martin joined ICL in 1970 with an honours degree in Pure Maths and Physics from Hull University(UK).

His early work was in computer architecture research, including multi-processors and database engines. He was one of the key designers of the original CAFS engine in the early 70s. During the 80s he worked as a systems strategist, and developed the strategy for exploiting CAFS in the open systems market, and co-invented SCAFS (with A.P.Graham Brown). He had management responsibility for implementing the SCAFS and **search** ***accelerator*** strategy, through design, development, collaborations, and product introduction. More recently he took responsible for ICL's Database Product Centre and currently acts as a Senior Systems Consultant in Server Systems, ICL Client-Server.

12. Glossary of Terms

Accelerator	short for Relational Database **search** ***accelerator***; it is an application of the SCAFS technology
BigEndian	A processor that stores a word into memory (and thus on disk) most significant byte at the lowest byte address (e.g. SPARC, 68000).
CAFS	Content Addressable File Store. A file-filtering technology available on ICL mainframe systems.
Daemon	A UNIX program with special privileges to manage a resource (e.g. SCAFS)
LittleEndian	A processor that stores a word into memory (and thus on disk) least significant byte at the lowest byte address (e.g. 386/486).
SCAFS	Son of CAFS (designed for the open systems marketplace, e.g. UNIX)
SCAFS device driver	Code built into an operating system to manage program, including Daemon, access to SCAFS,
SCSI	Small Computer System Interface, commonly used as the prime disk interface in UNIX based systems, PC servers, and workstations.
R.A.S.	Reliability, Availability, Serviceability.
RDBMS	Relational Database Management System; e.g. ORACLE, INFORMIX, INGRES, Sybase, etc.

Open Teleservice - A Framework for Service in the 1990s

Jerry Roddis

ICL Corporate Systems Division, Manchester, UK

Abstract

The exploitation of telecommunications and Information Technology is becoming increasingly significant in the delivery of service, and this topic is generally referred to as teleservice. This paper describes Open Teleservice, an architecture developed by ICL to provide a flexible and adaptable framework for service delivery. It describes the historical development of Open Teleservice, an insight into the underlying architecture, what it is currently being used for and also the potential for exploitation in the future.

1. Introduction

The IT service industry is becoming an increasingly significant area of business for IT vendors, as they see profit margins on hardware and software products continue to decline. Furthermore, the prevalence of third party service organisations and the emergence of Facilities Management are ensuring that the service industry itself is a highly competitive arena. It is, therefore, imperative that IT vendors are highly efficient, progressive, responsive and adaptable in order to compete and offer a quality service in an ever changing marketplace. The use of Information Technology (IT) in the provision of service is a key factor in realising a highly competitive organisation. Open Teleservice is a prime example of the use of IT in achieving this objective.

Teleservice is concerned with the exploitation of a telecommunication link between a service organisation and its customer base to help provide service. This link may be used to exchange information electronically in both directions, and also to permit remote systems access. It enables a service organisation to be highly efficient, both through automation and the concentration of skills into centres of excellence. It allows specialists to focus their expertise, minimising time spent on administration and travel. Furthermore, the customer organisation becomes more efficient

through having a controlled and consistent boundary between its in-house operations and procedures, and those of its service organisations.

Teleservice is now the *state-of-the-art* practice for delivering service within the IT industry, primarily for use in the traditional remedial support and maintenance service arena. There are many variations of teleservice provided by different hardware vendors and service organisations, with varying degrees of sophistication and functionality. Open Teleservice is designed to meet not only the traditional service requirements of the current customer base, but also to be adaptable and expandable to meet new service demands.

2. Background

ICL pioneered *Teleservice* in the early 1980s with its Series 39 range of mainframe systems, running the VME operating system. With complex and highly reliable systems such as Series 39, a fresh approach to the support and maintenance was necessary to provide a cost effective and competitive service. The industry could no longer afford to send large numbers of highly skilled system engineers to customer sites to resolve and correct problems. Thus, teleservice was developed with the emphasis on preventive maintenance and remote, speedy and automatic resolution of problems.

Teleservice on Series 39 uses intelligent performance monitoring and problem detection software resident on the customer systems for determining when service is required. This is linked via telecommunications to the service desk in the local ICL Support Centre, equipped with automatic resolution tools and drawing on a pool of expert diagnosticians. Typically, an incoming problem is first checked against a knowledge base of known errors. If a match is found, this will identify the appropriate corrective actions, otherwise the problem is passed on to a diagnostician to resolve. [Loach, 94] talks about a number of knowledge base tools used within ICL support centres.

Preventive maintenance of software ensures that VME systems are highly reliable because likely problems are cleared before they are ever encountered. Periodic on-site audits compare the applied fixes with the up-to-date recommended set. Any fixes which have not been applied are automatically retrieved via teleservice and made available for customer application.

In addition to support and maintenance, another service has been integrated, namely the news service. This is an information channel which gives customers access to a wide range of subject matter, including support bulletins, technical hints and tips, training, marketing and new products.

In recent years, UNIX systems have become firmly established in the commercial market place. To meet the servicing requirements of these systems, ICL has taken a fresh look at the provision of teleservice, and

this has resulted in Open Teleservice. The initial implementation was for the ICL DRS/NX platforms, with the range of platforms now expanding to meet the demands of the service business.

3. Architecture

The architecture of Open Teleservice, as developed by ICL, provides a framework for the delivery of service to customers. It allows any number of discrete services to be provided, the support and maintenance service being just one.

The Open Teleservice model is based on the concept of *Service Consumers* and *Service Providers*. A service consumer has a requirement for a particular service and a service provider is responsible for satisfying that requirement. In this context, a *service* can be viewed as any process which involves a consumer-provider dialogue. In may respects, the architecture can be viewed as an example of client-server, embodying many of the features described in [Brenner, 94].

In the support and maintenance service, a service consumer could be a customer with a service contract. The service provider would be the particular organisation(s) with the contractual responsibility for providing support and maintenance, for example, ICL or a Value Added Reseller (VAR). Indeed, the architecture is not restricted to ICL as a service provider. One of the key aspects of the *openness* is that the concepts and products are readily portable to other service communities.

Typically, each service will include an application in the consumer domain which will contain the appropriate functionality and user interfaces and will generate requests for service as required. In the provider domain there will be a complementary application which will accept the incoming requests and which will provide the necessary functionality and user interfaces to enable the provider to respond.

The model decouples the applications in both the consumer and provider domains from the message routing and communication handling. This is achieved by the Open Teleservice Interface (OTI) which defines the Application Programming Interface (API) for the service applications and defines a boundary to the communications handling function.

Figure 1 illustrates the key components and interfaces comprising the Open Teleservice architecture:

The Teleservice Electronic Data Interchange (TEDI) protocol defines the service messages which may be exchanged between consumer and provider domains. It defines a level of red tape information common across all services, and provides containers for service specific information. Each service can define which of the messages it wishes to exploit, and also the purpose for which they are being used.

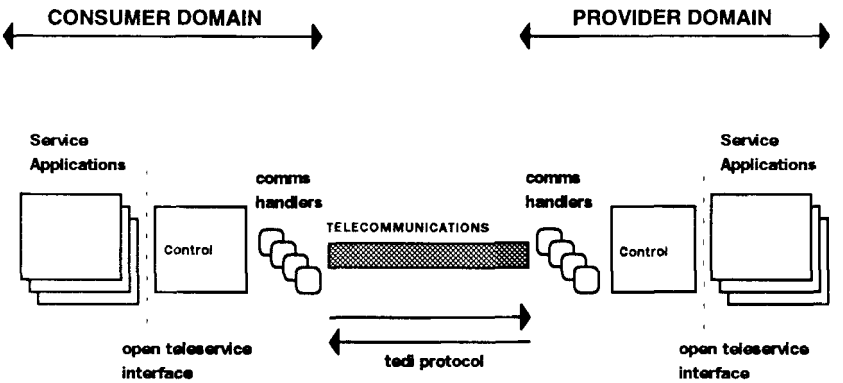


Figure 1 Open Teleservice Architectural Model

The following four message types may be generated by a consumer application:

Request for Service
This message initiates a service conversation and specifies the requirement of the consumer. By definition, this message is mandatory for all services.

Additional Information
This message allows a consumer to send additional information after the initial requests for service, either as a result of receiving a request for additional information from the provider, or as a voluntary action. Use of this message type is optional.

Request for Status
This message allows a consumer to enquire of the provider as to the progress of a particular request for service. Use of this message type is optional.

Service Complete
This message is an indication from the consumer that a particular conversation is closed. It may be used either as confirmation that a response from the provider has been accepted or, alternatively, to advise the provider that a request for service is to be withdrawn. Use of this message type is optional.

The following four message types may be generated by a provider application:

Acknowledgement
This message allows a provider to confirm receipt of a request for service. Use of this message type is optional.

Status Report
This message allows a provider to inform the consumer of the progress of a particular request, either as a result of receiving a request for status or as a voluntary action. Use of this message type is optional.

Request for Additional Information
This message allows a provider to ask the consumer to send further information concerning a particular request for service. Use of this message type is optional.

Service Response
This message is complementary to the original request for service and contains the response. The provider is obliged to send such a message, unless the consumer has previously withdrawn the request. If, on receipt of the service response, the consumer is not satisfied, then the request for service can be resent and the conversation re-opened.

The functionality in Figure 1 between the OTI in the consumer and provider domains offers a middleware function and is often referred to as the *pipework*. This supports a number of generic features designed to ease the integration of new services, and includes many which are typical of client-server architecture as described in [Brenner, 94]:

- no limit on the number of participating services
- independence from hardware platform, operating system and communications technology
- isolation of service applications from the underlying communications technology and re-use of the common middleware functionality
- flexible message set, allowing service specific definitions
- independence of consumer or provider products for a given service, e.g. different integrated service desks used by different service organisations
- no limit on the number of possible service providers for a given service
- service provider and service consumer functions may exist on the same system
- combining the consumer and provider functions into the same application allows chaining together to form a networked service.

4. Current Functionality

This section describes the functionality of the Open Teleservice product set which has been developed by ICL and is now established in the field.

4.1 Open teleservice pipework

There is the middleware component of Open Teleservice, which supports the interface to the various service applications in the consumer and provider domains, and which handles the telecommunications and message routing between applications.

When a consumer application submits a message, it is required to name the target service provider. The pipework will manage the communications path to the provider and into the target provider

application. To feed a reply back to a consumer, the provider simply has to name the original request source.

The choice of communications technology is based on what is most appropriate for the circumstances, and may be one of the following classes:

- via the X/Open Transport Interface (XTI), including OSLAN, TCP/IP and X25
- asynchronous dial-up via a modem.

If the choice is an asynchronous modem, there are a number of further options which may apply:

- point-to-point, manual or auto-dial
- dial-up to an X29 PAD (e.g. BT Dialplus), manual or auto-dial.

When using an auto-dial modem, there are a number of options for managing how and when a link is established;

- periodically, at predetermined intervals
- for requests which are sufficiently urgent to warrant an immediate connection
- manual initiation only, e.g. by the administrator.

When using a dial-up modem, messages can queue up in both the consumer and provider domains until a connection is made. On establishing a connection, all queued messages will then be exchanged. When using a transport technology, messages need not be queued but can be relayed immediately, thereby enabling a more responsive service.

4.2 Problem reporting service

The Problem Reporting Service enables problems reported within the consumer domain to be notified to the provider for resolution and correction. Figure 2 illustrates the key components of this service.

Each consumer system is equipped with a management system (referred to as the Problem Service) which is capable of detecting problems automatically and also allows users to report problems which they spot themselves. It provides facilities to the customer for viewing and progressing problems and for engaging in a dialogue with a remote service provider. Some of the end-user facilities will result in TEDI messages being submitted, for example, the act of transferring a problem will cause a *request for service* to be generated.

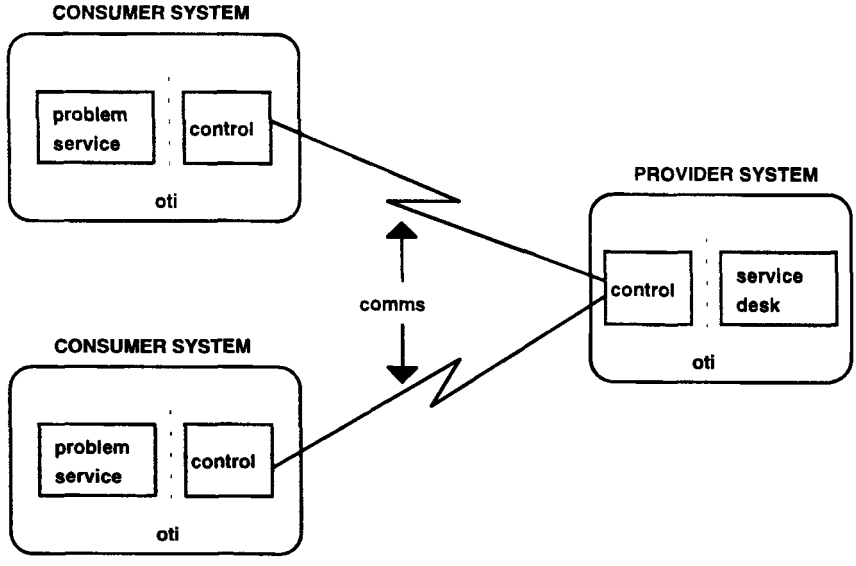


Figure 2 Overview of the Problem Reporting Service

Problems can be managed entirely locally if the resolution and correction can be performed by the consumer. Alternatively, a problem can be transferred to the remote service provider, whereupon it is automatically recorded in the service desk of the provider. This action opens up a dialogue which will typically conclude with the provider returning the solution to the problem. The full range of TEDI messages are exploited to support the following operations:

Consumer

- initial problem transfer
- supply further evidence (solicited or unsolicited)
- ask for a progress report
- re-open a problem if the solution is unacceptable
- confirm acceptance of the solution
- withdraw a problem.

Provider

- acknowledge registration of the problem in the Service Desk
- provide a progress report (solicited or unsolicited)
- ask for further evidence
- send the solution.

Figure 3 illustrates the key components of the problem service on a consumer system.

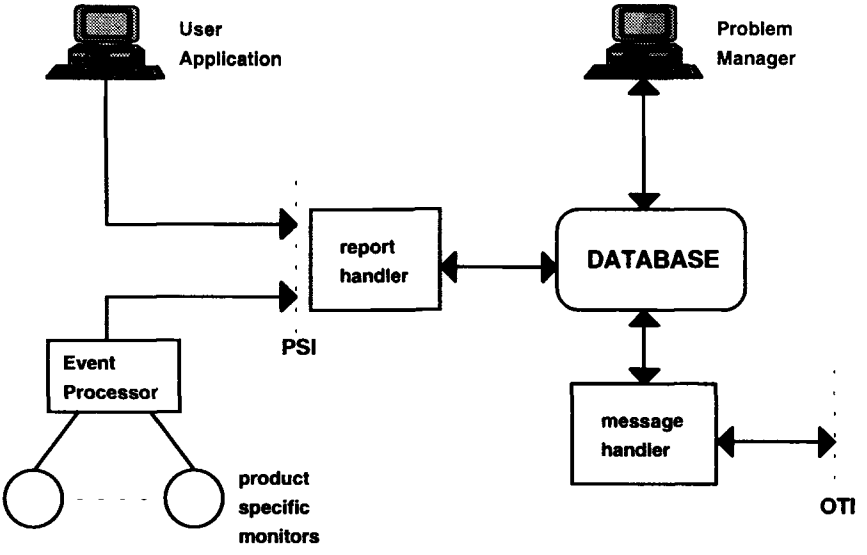


Figure 3 The Problem Service in the consumer domain

The interactive application supports all of the functions necessary to view and progress a problem. It supports a generic problem reporting function, which offers a simple means of reporting problems on any type of product using a structured sequence of forms.

Underlying this application is a relational database. This holds all the captured details for each problem, and a full audit trail of subsequent events associated with that problem, including messages sent to and received from a remote provider. The dialogue with a remote provider is dealt with by the *message handler*, which connects to the OTI to submit requests and read replies.

The Problem Service Interface (PSI) permits external applications to report problems. Problems detected automatically by the system are submitted through this interface. The event processor provides a framework into which a range of product specific monitoring programs can plug. It supports re-usable functions such as threshold checking. A further use of the PSI is to allow end-user applications to contain a problem reporting facility so that problems can be entered directly from the operational environment of the end user. In all cases, the emphasis is on providing intelligently collated summarised evidence for transmission with the problem itself, to enable rapid resolution and to minimise the need to examine the raw evidence on the consumer system.

In a distributed system environment it may be the case that the customer wishes to centralise the management of problems and have a single view

of the problem status across all systems. It may also be the case that the customer wishes to manage the point of communication with the service provider from a single point. To meet this requirement, a product called Incident Manager has been developed. This collects problems from the separate systems and allows an administrator to view and manage all problems from a single point. It reduces the administrative overheads and also presents a more informative picture of the state of the whole enterprise. Alternatively, it may be that the customer is already operating a help desk, in which case there is scope for interfacing it to Open Teleservice.

In architectural terms, Incident Manager or an integrated help desk is acting as a provider to the distributed systems and as a consumer to the remote provider.

4.3 News retrieval service

The News Retrieval Service offers an information channel between a service provider and its customer base. Customers are presented with an index of news articles from which they can select those of interest, which are then retrieved via teleservice. There are no limits to what subject matter is made available, although typically this may include support information, technical advice and guidance, sales, training, promotional and user group submissions. The service provider will generally require some form of editorial control to vet items prior to general availability.

News articles are organised within an index according to subject matter. A provider can also provide several indexes if necessary, for example, to provide foreign language translations of the articles. A single provider can even run several discrete services to give different groups of consumers access to specific indexes. This may be necessary if there are privacy or charging considerations.

Within the consumer domain there is an interactive application called the News Service. This allows the consumer to scan the index of available articles and to mark those which are required. Once retrieved, an article can be browsed interactively, printed off and copied to a file. The consumer is kept up to date with what information is available by periodically retrieving the index automatically.

The product within the provider domain is called News Manager. This comprises two components, the repository which holds the information and the process which services incoming requests. News Manager may also be used by a customer organisation to provide a local information service under the editorial control of the customer. The News Service can receive news from more than one service provider, enabling a customer system to have a view of the local news as well as from any number of remote providers.

Figure 4 illustrates the key components of the News Retrieval Service.

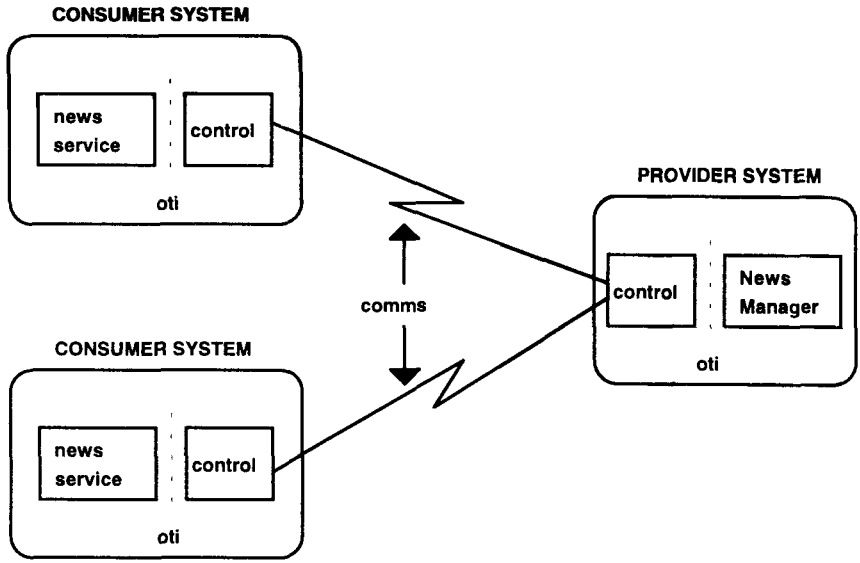


Figure 4 Overview of the News Retrieval Service

Only a subset of the TEDI message set is required by the News Retrieval Service to support the following functions:

Consumer

- request a particular item of news.

Provider

- deliver a requested item of news
- reject a request if the consumer is not entitled to the news.

Figure 5 illustrates the organisation of the news information within the News Manager repository:

The information is organised in a structured hierarchy, with the top level representing the service provider itself. Associated with the provider is an optional list of valid consumers, which is used to verify that an incoming request has originated from a recognised system.

A service provider owns one or more news indexes as required. Within each index there is provision for a blanket disclaimer statement. There is also a headlines section to contain information which is given prominent visibility in the customer News Service. Within the index, the individual articles are grouped into categories, typically based on subject matter. An index may have any number of categories, and each category may have any number of articles.

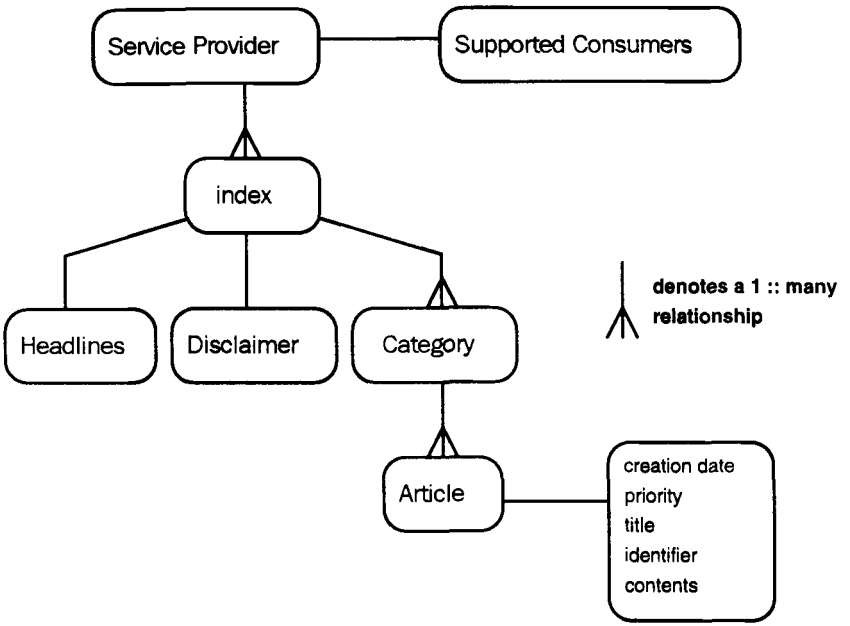


Figure 5 The News Manager Repository

4.4 Telediagnostics

The Telediagnostics facility allows the service provider to establish a remote session on a customer system. Typically, this is used to perform on-line diagnosis of a problem on a system where there is insufficient evidence defined within the problem submitted via teleservice.

There are varying degrees of security associated with this feature. It is always the customer system which establishes the connection, preventing unsolicited access. The customer specifies which username the diagnostician may log into, thereby constraining access to the system. As an additional security measure, there is a feature referred to as *Session Mirroring*, which allows the customer full visibility of a diagnostician's actions on a local terminal and vice-versa. This preserves a journal of all actions for audit purposes and also allows the customer to terminate a connection at any stage. Session mirroring is especially useful for a customer to demonstrate what was being done when a problem was encountered, and also for the customer to observe the activities of a diagnostician in resolving the problem.

During a Telediagnostic session, evidence can be moved from the customer system to one in the provider domain, for analysis at the diagnostician's convenience and also to avoid holding the link open for excessive periods. Data can be passed the other way onto the client system if required, for example, to send a work-around or a software fix.

5. Field Exploitation

Open Teleservice is now well established and in active use by ICL. It is currently operational on ICL's DRS/NX system range, covering both SPARC and Intel hardware platforms. It supports the Team and Super Servers running DRS/NX, and this will be extended to cover SCO and UNIXware. It also supports GOLDRUSH Megaserver, the massively parallel SQL server for large relational databases.

The Problem Reporting Service and News Service are delivered by ICL as part of a suitable support contract.

For DRS/NX platforms, there is automatic monitoring of SCSI devices which will report problems and also identify problem trends to enable preventive maintenance. On GOLDRUSH Megaserver there is considerable automatic monitoring of the complex hardware and software components.

If the system is running OfficePower then there is automatic monitoring of the state of OfficePower itself. Problems concerning OfficePower can also be reported interactively from within an OfficePower session.

Open Teleservice has been introduced in the UK and a number of overseas territories. This has involved establishing both the software environment and the necessary operational procedures. There are several service desk products in use throughout the ICL community, and a number of these have been interfaced to Open Teleservice. In the UK, there is a central service desk system known as CRISP. Many of the European territories use a product known as SMS, and a further desk system called PRONTO has been interfaced to meet the requirements of smaller territories.

Each territory chooses the most appropriate communications technology, depending on the possible alternatives. In the United Kingdom, the chosen technology has been asynchronous dial-up to an X29 PAD, provided by the British Telecom Dialplus service. This enables customers to connect to the most convenient local PAD to benefit from cheap call rates. Customers who prefer to use a transport protocol may do so if they wish.

6. Future Direction

6.1 Additional services

The emergence of professional services as a significant growth area will open up the possibilities for exploiting Open Teleservice. One example which has already come to fruition is the remote system administration service which ICL provides for customers who do not wish to undertake their own administration. This service now uses teleservice to deliver the state of the supported systems on a daily basis to the administration experts inside ICL. This removes the overhead of remotely accessing each

system to see if any administration action is required. It enables more responsive and proactive action than could otherwise be achieved.

The same approach for delivering information to the central service provider could be considered for a remote capacity management and planning service, to analyse the utilisation of customer systems and advise on upgrades and performance tuning.

It would be possible to offer an electronic product ordering service, whereby customers made selections from an electronic catalogue of orderable products. Teleservice could be used to relay the orders to the appropriate ordering system within ICL.

6.2 Multivendor support

Customer organisations are increasingly purchasing IT products from several vendors to avoid reliance on any particular one. However, more and more customers are appointing a single service provider to provide support for all their IT, hence the need to be able to offer multivendor support.

Open Teleservice is able to operate in a multivendor environment through a dual strategy of porting and integration.

Where the service application functionality is required on a new platform, then porting is the most likely approach. The Open Teleservice portfolio is readily portable onto other vendors' variants of UNIX, especially those which are XPG compliant.

In many cases, there are already applications in place for managing specific domains within an enterprise, such as Sun Net Manager and HP Open View, which are especially well suited to managing networked environments. Where these are in place, there is scope for integrating with Open Teleservice to provide a path for the notification of events to a remote service provider.

6.3 Exploitation by third parties

Open Teleservice could be used to good effect by service organisations other than ICL, for example, by VARs and ICL's service partners. The third party could establish its own Open Teleservice community of its supported customer base. If the third party ever needed to pass calls to ICL, or indeed, any other party, then again Open Teleservice could be used. It could integrate its own Service Desk to manage service requests or it could employ Incident Manager as an entry level desk facility. The third party could establish its own news service and also benefit from new services as they became available.

7. Conclusion

Open Teleservice is now well established as the method for providing remedial service by the ICL support community world-wide. It is helping to provide benefits to both ICL and customers alike, through helping to deliver responsive and proactive service. The occasions where ICL has to

contact customers for further information concerning a problem are drastically reduced, and the turnaround of problems is becoming increasingly faster through the delivery of evidence with the problem reports and the use of Telediagnostics sessions. The news service supplements the support service through the publication of support bulletins, as well as providing a general source of useful information.

The continued enhancement and expansion of the entire Open Teleservice capability is of paramount importance in maintaining efficient and cost effective service delivery. This must look beyond the traditional support and maintenance services to explore new markets and opportunities. The flexibility and adaptability provided by the architecture make it especially well equipped to meet the ever changing service requirements of IT community.

The possibilities of making use of the electronic connection between consumers and providers for the delivery of service are manifold. Open Teleservice will help to make these possibilities a reality.

8. Acknowledgements

The success of the Open Teleservice programme has been based on teamwork and cooperation spanning many divisional boundaries within ICL. It has combined experience and expertise from within both Corporate Systems and Client Server Systems Divisions and also from Customer Services in the UK, Europe and International.

9. References

BRENNER, J.B. "Client-Server Architecture", *ICL Tech. J*, Vol.9(1) pp 1-17, 1994

LOACH, P.J. "ICL's Problem & Resolution Information Service", *ICL Tech. J*, Vol.9(1) pp 180-190, 1994

10. Biography

Jerry Roddis

Jerry Roddis graduated from the University of Newcastle Upon Tyne with a Joint Honours degree in Electrical Engineering and Computing Science. He joined ICL in 1979, working initially on developing test software for a variety of hardware and communications products.

He has been involved in developing ICL's Teleservice capability for mainframe systems, and more recently in developing Open Teleservice.

Book Review

LEO - THE FIRST BUSINESS COMPUTER

A memoire by Frank Land

LEO was arguably the world's first general purpose business computer, started in the late 1940s on which work reached fruition in the 1950s and early 60s.

As one who was directly involved in the LEO story between 1952 and 1967 I am prompted to write the following notes by the appearance of the book by Peter Bird[1]

In the 1940s and 1950s J Lyons was one of the most successful businesses in the country with its products and establishments - Lyons Tea, Lyons Cakes, Lyons Ice Cream and the Teashops and Corner Houses - being household names. It had built its success on quality products and services sold to a mass market, and a constant striving for value-adding innovation.

Selling to a competitive mass market required tight control over costs and margins, and a sensitive response to customer preferences and market movements.

John Kay[2] in his study of what makes businesses successful suggests that *architecture* is one of the important ingredients. The distinctive architecture which Lyons had developed over the years was the way information was passed from operations - manufacturing, selling,

---

[1] LEO - The First Business Computer, by Peter Bird, Hasler Publishing, Workingham, 1994. ISBN 0-9521651-0-4.

[2] Kay, John, 1993, Foundations of Corporate Success, Oxford University Press, Oxford.

distribution, as well as the concomitant operations concerned with invoicing and payments - to the decision making senior management. Each of the many businesses (Tea, Teashops, Ice Cream, Bakeries, Kitchens, etc.) had its own groups of clerks and managers. The vast mass of transaction data stemming from these operations was summarised and compared with pre-set standards, forecasts and budgets. The resulting information was analysed by the junior manager in charge of each group, who would be responsible for explaining any important variances. The junior manager had a direct line to the senior manager, often a Lyons Director, responsible for that activity and had to explain the functioning of that activity. At the same time the senior manager could ask the junior liaison manager to undertake investigation of the *what if* type - suppose we wish to increase the production of swiss rolls by 10% and reduce the production of cup cakes by 3%: what would be the effect on gross profit? The arrangement ensured direct access by senior management to information originating at the operating level and by-passed the more usual filtering through layers of middle management.

This architecture provided the company, long before the advent of computers, with both an almost real time management information system and a decision support system of considerable sophistication. In addition the architecture and system provided senior management with a detailed picture of the week's trading on the Monday of the following week.

The management had seen the need to innovate not only in new products and services but also in *business processes* as early as the 1920s when they had engaged a senior wrangler from Cambridge to oversee the office functions which already then were seen to be the source of information for management. They established a systems research office whose function was to analyse primarily office operations in order to see how processes could be improved to provide better control and to reduce costs.

The systems research office working with line managers produced a stream of business process innovations from the time of its establishment. Examples include the notion that each sales representative, each having a customer group of many small retailers, would be responsible not only for selling to his customer group, but totally responsible for the accounting, credit and payment functions conventionally carried out at arms length by a separate accounting office. The introduction of *traveller covered credit* was a radical business process innovation which increased efficiency and the effectiveness of the representative .

Yet in many other ways the company was deeply traditional and conservative. It operated on a strictly hierarchical basis. At the top were the owners, the founding family. They ran the company with the help of a very few employee directors. Each grade of management had its own dining room, with at the pinnacle the dining room reserved for the family directors. Separate toilets divided managers from the rest. Trade Unions

were discouraged, though the family took a paternalistic interest in its staff.

In his book Peter Bird tells the story ofJ, Lyons in his early chapters and gives many more examples of innovation and enterprise in the way the business was run.

This was the company I joined in 1952 as a recent graduate in Economics. My first job was as a clerk in the statistical office - one of the major offices compiling and checking transaction data for posting to the cost accounts of the various operating units. I was responsible for keeping the cost accounts of the Provincial Bakeries for further analysis and interpretation by the junior manager Alec Kirby. I learned then that the laid down routine for almost all staff down to the lowest clerk covered only a portion of the time available. Much time was spent on tracing errors and various forms of trouble shooting. The more senior clerks and junior managers seemed to spend most of their time on that kind of activity. As in much of UK manufacturing industry *progress chasing* kept the wheels turning.

At that time Lyons had already embarked on its pioneering adventure with computers. In retrospect the move into computers is not so surprising. The systems research office had investigated the possibility of coping with the mass of transaction data by some kind of mechanisation or automation for many years. They had started to investigate the possibility of devising a document reader for transaction data before the second world war. They had researched the possible application of unit record systems based on punched cards, but rejected these as too limited, too constraining and too costly. Lyons had only one punched card installation and that had a very limited application. Instead they had installed alternative types of office mechanisation based on accounting machines and calculators. Computers, it was reasoned, had the capability of overcoming the limitations of the unit record equipment then available.

I knew nothing of the experiments going on with the LEO computer. But as the LEO group expanded the company trawled for possible recruits from its offices. It was suggested that I might like to learn about LEO to see if I was interested and fitted their requirements for computer programmers. I was put on a one week LEO *appreciation* course. The course taught us the rudiments of binary arithmetic, programming and how the computer worked. It was tough. Each evening I would go home, sometimes in despair, and together with my wife worked at the home work and to master the exercises. By the end of the week I was still in a fog, but felt that joining the LEO group would be a most exciting challenge, and certainly an improvement over the by then rather boring Provincial Bakeries.

In 1953 I was selected to join the LEO group. At that time the first LEO (LEO I) was being commissioned in its final form. Magnetic tape, though still standing around had been abandoned for the time being. Punched card and paper tape were the main input devices, though it was possible

to intervene directly from the console to change the program or data, or to single step ones way through a program when debugging. Output was also on punched cards and paper tape, but also in printed form directly on to a line printing tabulator. The computer was by present standards very unreliable, and the combination of programming errors, data errors and an unreliable machine made the log entry *passed point of previous stoppage* a particular joy for the operators and programmers. Nevertheless, a substantial amount of work was being carried out.

The first applications, primarily for demonstration purposes, had gone live three or four years earlier. Now the team was working on the various Lyons' payrolls (L1), and beginning to plan the teashop ordering job(L2), and the reserve stores allocation job(L3). In addition the machine was being utilised as a service bureau with work, principally of a scientific nature being carried out for a number of external organisations. Some of these involved programming to the customer specification by a member of the LEO team but others were programmed by the customer's own staff and LEO merely provided machine time.

Although the Lyons' applications were some of the earliest business systems on a computer, a pattern of planning and development had already been established, and for any application a kind of standard of 'good practice' had been laid down. Many of these standards are as relevant today as they were in the early 1950s.

Selecting and planning the use of LEO for business processes which might benefit from the use of computers was in the hands of the senior LEO management (T.R. Thompson and David Caminer) and the Systems Research Office, working with senior Lyons managers, often Directors and line management from the business area affected. To be selected an application had to provide clear cost savings as against conventional methods, or had to make the business process more effective as well as showing savings in costs. The planners would look for opportunities to improve the business process in a way which would not have been possible with alternative methods.

There were many examples of such improvements. The teashop ordering job (L2) provided a number of instances. The teashops were provided with food by central kitchens at Cadby Hall. The teashop ordering job made possible a major business process innovation. This provided a cheap and effective way of providing local teashop managers with some choice on what to have available for teashop customers. The manager was provided with a standard menu - a list of food items with quantities - for each day of the week. She would order the standard menu and alter some of the standard quantities and delete or add items. The Teashop senior management could override the teashop order under certain circumstances. For example if the weather forecast for the next day suggested that it would be a very hot day, the orders for the shops likely to be affected would be altered to ensure a menu suitable for hot weather. The only data required by the computer were the departures from the

standard menu plus any alterations advised by senior management. This enabled the computer to provide the kitchens with a full manufacturing order for each teashop, together with a list of raw material required and a full costing of the order. The computer would organise the teashop orders in delivery sequence for delivery by van to the teashops - another item of increased effectiveness provided by the system. Data entry, the capture of the orders from the managers was effected by the simple mechanism of each manager 'phoning in her order at a scheduled time and the telephone operator punching the variations from the standard menu directly on to a punched card for entry into the computer.

Another early application again illustrates the focus on using the computer to achieve extra value. In 1953 rationing of foodstuffs was still in force, and Lyons as manufacturers of food had to rely on a great variety of materials including substitutes for the natural raw materials. For example, a substitute for sugar used in the bakeries was sweetened fat. The substitute materials were held in so called reserve stores. An application was planned and developed for maintaining the inventory records of all the material held in the reserve stores and allocating these to the manufacturing centres, bakeries, ice cream plants and kitchens, on the basis of orders received and manufacturing schedules. The computer received manufacturing schedules as data, and consolidated and allocated material from reserve stores according to the availability of transport in order to keep distribution costs to a minimum. As with all LEO jobs the automatic procedure could always be overridden by management action. Costs were calculated for each store and charged out to the manufacturing departments. The reserve stores allocation job ran successfully only for a short time as it was abandoned once rationing was ended and material could be ordered from suppliers on something close to a just-in-time basis.

Whilst the focus of the more senior LEO management was on business processes and the selection and planning of applications those of us involved with getting the work onto the computer lived and dreamt of the technical problems of getting the programs to work. Each job presented major problems of fitting the tasks required into the small computer store and at the same time getting the job to run efficiently. The trade-off between saving instructions by tight programming and saving time in execution was a constant problem. Saving the execution of one instruction in a loop was a triumph and reported on and discussed at meal times and refreshment breaks. Programming so that a transaction could be fully computer checked within the time it took to read a punched card was good, missing the cycle by a fraction of a second could double the execution time of a job with thousands of transactions to be dealt with. Every day produced new tricks of programming and avid discussions on how the trick was done.

At the same time the standards of good practice already in force were maintained and supplemented. Senior management kept a tight discipline which ensured high quality and safe applications. No program was

allowed on the computer for debugging without it having being desk checked by a colleague. This practice helped the learning and the spread of best practice amongst the team. All applications had built in reconciliation procedures which were based on good accounting practice and which not only ensured accurate work from the computer, but helped to pin-point mistakes when they occurred, and demonstrated to the business user the integrity of the work done on the computer.

In the year I joined the LEO group the Lyons payroll and the teashop ordering jobs were rolled out and became jobs routinely carried out on a daily or weekly basis. Having routine jobs required the computer to be run by a cadre of professional operators. Hence a new operating section with initially two computer operators and a number of data preparation staff was set up.

By now what had been regarded by the outside world as a perverse experiment was reality. Knowledge about what was happening at Cadby Hall began to spread largely through the Office Management Association of which John Simmons, a Lyons employee Director, was President. A frequent occurrence was to take visitors around the computer and to talk about the applications. As a result, enquiries about the possible use of the computer for their own business began to arrive. For the LEO and Lyons management this suggested the possibility of making LEO an independent business manufacturing and selling computers and running computers as a service bureau for other companies.

With the requirement to continue to develop applications for the Lyons group and to provide for the planning, development and implementation of applications for an outside market increased the need for programming staff and a period of recruitment and expansion followed. For the group already in place it meant that the lessons learned from the successes of the Lyons' applications and the standards of good practice which had been absorbed into the way of working, had to be applied rapidly to a range of new jobs on behalf of outside clients. Of these, the biggest to be carried out on a regular basis, was the weekly Ford Motor Company payroll for over 20,000 workers. In terms of developing operational practices, providing regular secure delivery of outputs to a host of service customers provided a major challenge, which the LEO team met successfully.

But to some of us the real challenge was understanding and mastering the business need of a very diverse group of customers covering almost all sections of British, and in later years East-European business. The fact that this offspring of a food business could sell systems to a wide range of blue chip companies and Government departments - listed in appendix 6 and appendix 9 of Bird's book - is an outcome of the quality of the team recruited by the LEO management but even more of the good practices they had inherited from the Lyons pioneers. Many of those involved in those early days have risen to high ranks in industry and also in academia reflecting the important role of the LEO venture as an educator, almost as a business school.

Peter Bird has provided an excellent chronological account of the LEO project from its beginnings in the late 1940s to its absorbtion into first English Electric and subsequently ICL at the end of the 1960s. But some aspects of the story still need to be told. In particular those which relate to the development of the coherent and clear systems understanding which the LEO pioneers instilled in the team and which underlay, as much as hardware and software engineering, the achievements of the LEO venture, and which has spread out from its origins in LEO to locations all over the world.

Frank Land was, until recently, Professor of Information Management at London Business School and is now Visiting Professor at London School of Economics.

Architext

The **OPEN***framework* Systems Architecture Series from ICL

now available on CD-ROM!

The OPENframework Systems Architecture is explained in a series of books, each one concentrating on one aspect in particular. Individually each book provides an excellent study of the major technology trends and offers detailed advice and guidance for decision making. Together, they provide a comprehensive set of analysis tools for designing information systems.

Highly illustrated and using everyday language, the series is an ideal companion for those involved in planning or implementing an information systems strategy. Books are colour-coded for easy identification.

Architext delivers the books in hypertext format which brings even more powerful facilities than just the usual benefits of electronic documentation. For instance, you have complex search capabilities as well as being able to extract text and diagrams for proposals etc. However, hypertext also provides the ability to be able to annotate with your own comments; bookmark pages; print sections etc., just as with the printed form.

Take advantage of our special introductory offer ...
a FREE CD drive with your first order!

Contact the *SystemWise* Help Desk for more information

+44 (0)181 565 7993

(ICL speedcall 7993)

Index of Technical Journal Papers by Issue

Vol.1 Iss.1 - November 1978

The origins of the 2900 series
Sizing computer systems and workloads
Wind of Change
Standards for open-network operation
Distributed computing in business data processing
A general model for integrity control

Vol.1 Iss.2 - May 1979

Computers in support of agriculture in developing countries
Software and algorithms for the Distributed Array Processor
Hardware monitoring on the 2900 range
Network models of system performance
Advanced technology in printing: the laser printer
The new frontier: three essays on job control

Vol.1 Iss.3 - November 1979

Meteosat 1: Europe's first meteorological satellite
An analysis of checkpointing
Statistical and related systems
Structured programming techniques in interrupt-driven routines
The content addressable file store - CAFS
Computing in the humanities
The data dictionary system in analysis and design

Vol.2 Iss.1 - May 1980

Security and privacy of data held in computers
CADES - software engineering in practice
ME29 Initial Program Load: an exercise in defensive programming
Project Little - an experimental ultra-reliable system
Flow of instructions through a pipelined processor
Towards an 'expert' diagnostic system
Using Open System Interconnection standards

Vol.2 Iss.2 - November 1980

The ICL Information Processing Architecture, IPA
VME/B: a model for the realisation of a total system concept
Birds, Bs and CRTs
Solution of elliptic partial differential equations on the ICL Distributed Array Processor
Data routing and transpositions in processor arrays
A Bayesian approach to test modelling

Vol.2 Iss.3 - May 1981

A dynamic database for econometric modelling
Personnel on CAFS: a case study
Giving the computer a voice
Data integrity and the implications for back-up
Applications of the ICL Distributed Array Processor to econometric computations
A high-level logic design system
Measures of programming complexity

Vol.2 Iss.4 - November 1981

Architecture of the ICL System 25
Designing for the X25 telecommunications standard
Viewdata and the ICL Bulletin System
Development philosophy and fundamental processing concepts of the ICL Rapid Application Development System RADS
A moving-mesh plasma equilibrium problem on the ICL Distributed Array Processor

Vol.3 Iss.1 - May 1982

Information Technology Year 1982
Software of the ICL System 25
Security in a large general-purpose operating system: ICL's approach in VME/2900
Systems evolution dynamics of VME/B
Software aspects of the Exeter Community Health Services Computer Project
Associative data management system
Evaluating manufacturing testing strategies

Vol.3 Iss.2 - November 1982

The advance of Information Technology
Computing for the needs of development in the smallholder sector
The PERQ workstation and the distributed computing environment
Some techniques for handling encipherment keys
The use of COBOL for scientific data processing
Recognition of hand-written characters using the DAP
Hardware design faults: a classification and some measurements

Vol.3 Iss.3 - May 1983

IPA networking architecture
IPA data interchange and networking facilities
The IPA telecommunications function
IPA community management
MACROLAN: a high-performance network
Specification in CSP language of the ECMA-72 Class 4 transport protocol
Evolution of switched telecommunication networks
DAP in action

Vol.3 Iss.4 - November 1983

Expert system in heavy industry: an application of ICLX in a British Steel Corporation works
Dragon: the development of an expert sizing system
The logic language PROLOG-M in database technology and intelligent knowledge-based systems
QPROC: a natural language database enquiry system implemented in PROLOG
Modelling software support

Vol.4 Iss.1 - May 1984

The ICL University Research Council
The Atlas 10 computer
Towards better specifications
Solution of the global element equations on the ICL DAP
Quality model of system design and integration
Software cost models
Program history records: a system of software data collection and analysis

Vol.4 Iss.2 - November 1984

Modelling a multi-processor designed for telecommunication systems control
Tracking of LSI chips and printed circuit boards using the ICL Distributed Array Processor
Sorting on DAP
User functions for the generation and distribution of encipherment keys
Analysis of software failure data(1): adaptation of the Littlewood stochastic reliability growth model for coarse data
Towards a formal specification of the ICL Data Dictionary

Vol.4 Iss.3 - May 1985

Overview of the ICL Series 39 Level 30 system
VME nodal architecture: a model for the realisation of a distributed system concept
Processing node of the ICL Series 39 Level 30 system
Input/output controller and local area networks of the ICL Series 39 Level 30 system
The store of the ICL Series 39 Level 30 system
The high-speed peripheral controller for the Series 39 system
Development of 8000-gate CMOS gate arrays for the ICL Level 30 system
Development route for the C8K 8000-gate CMOS array
Design automation tools used in the development of the ICL Series 39 Level 30 system
Design and manufacture of the cabinet for the ICL Series 39 Level 30 system
Manufacturing the level 30 system I Mercury: an advanced production line
Manufacturing the Level 30 system II Merlin: an advanced printed circuit board manufacturing system
Manufacturing the Level 30 system III The test system
Patent applications arising out of the Level 30 project

Vol.4 Iss.4 - November 1985

History of the ICL content-addressable file store, (CAFS)
History of the CAFS relational software
The CAFS system today and tomorrow

Development of the CAFS-ISP controller product for Series 29 and 39 systems
CAFS-ISP: issues for the applications designer
Using secondary indexes for large CAFS databases
Creating an end-user CAFS service
Textmaster - a document retrieval system using CAFS-ISP
CAFS and text: the view from academia
Secrets of the sky: the IRAS data at Queen Mary College
CAFS file-correlation unit

Vol.5 Iss.1 - May 1986

ICL company research and development, 1904-1959
Innovation in computational architecture and design
REMIT: a natural language paraphraser for relational query expressions
Natural language database enquiry
The *me too* method of software design
Formal specification - a simple example
The effects of inspections on software quality and productivity
Recent developments in image data compression for digital facsimile
Message structure as a determinant of message processing system structure
Suggested extension of ICL DAP parallelism

Vol.5 Iss.2 - November 1986

The Management into the 1990s Research Programme
Managing strategic ideas: the role of the computer
A study of interactive computing at top management levels
A management support environment
Managing change and gaining corporate commitment
An approach to information technology planning
Preparing and organising for IPSE
Global Language for Distributed Data Integration
The design of distributed secure logical machines
Mathematical logic in the large practical world
The ICL DRS300 management graphics system
Performance of OSLAN local area network
Experience with programming parallel signal-processing algorithms in Fortran 8X

Vol.5 Iss.3 - May 1987

What is Fifth Generation? - the scope of the ICL programme
The Alvey DHSS Large Demonstrator Project
PARAMEDICL: a computer-aided medical diagnosis system for parallel architectures
S39XC - a configurer for Series 39 mainframe systems
The application of knowledge-based systems to computer capacity management
On knowledge bases at ECRC
Logic languages and relational databases: the design and implementation of Educe
The semantic aspects of MMI
Language overview
PISA - a Persistent Information Space Architecture
Software development using functional programming languages

Dactl: a computational model and compiler target language based on graph reduction
Designing system software for parallel declarative systems
Flagship computational models and machine architecture
Flagship hardware and implementation
GRIP: a parallel graph-reduction machine

Vol.5 Iss.4 - November 1987

Open Distributed Processing
The Advanced Network Systems Architecture project
Community management for the ICL networked production line
The X/OPEN Group and the Common Applications Environment
Security in distributed information systems: needs, problems and solutions
Cryptographic file storage
Standards and office information
Introducing ODA
The Technical and Office Protocols - TOP
X400 - international information distribution
A general purpose natural language interface: design and application as a database front end
DAP-Ada: Ada facilities for SIMD architectures
Quick language implementation

Vol.6 Iss.1 - May 1988

ICL Series 39 support process
The ICL systems support centre organisation
ICL Services Product Centre
Knowledge engineering as an aid to the system service desks
Logic analysers for system problem solving
Repair - past and future
OSI migration
A Network to Support Application Software Development
Universal Communications Cabling: A Building Utility
Collecting and generalising knowledge descriptions from task analysis data
The architecture of an automated Quality Management System
ICL Company Research and Development Part 2: Mergers and Mainframes, 1959-1968

Vol.6 Iss.2 - November 1988

Manufacturing at ICL's Ashton plant
Knowledge based systems in computer based manufacturing
Open systems architecture for CIM
MAES - An expert system applied to the planning of material supply in computer manufacturing
JIT and IT
Computer Aided Process Planning (CAPP): Experience at Dowty Fuel Systems
Use of integrated electronic mail within databases to control processes
Value engineering - a tool for product cost reduction
ASP: Artwork specifications in Prolog
Elastomer technology for probing high-density printed circuit boards
The effects of back-driving surface mounted digital integrated circuits

Reliability of surface-mounted component soldered joints produced by vapour phase, infrared soldering techniques
Materials evaluation
On the human side of technology

Vol.6 Iss.3 - May 1989

Tools, Methods and Theories: a personal view of progress towards Systems Engineering
Systems Integration
An architectural framework for systems
Twenty Years with Support Environments
An Introduction to the IPSE 2.5 Project
The case for CASE
The UK Inland Revenue operational systems
La solution ICL chez Carrefour a Orleans
A Formally-Specified In-Store System for the Retail Sector towards a Geographic Information System
Ingres Physical Design Adviser: a prototype system for advising on the physical design of an Ingres relational database
KANT - a Knowledge Analysis Tool
Pure Logic Language
The 'Design to Product' Alvey Demonstrator

Vol.6 Iss.4 - November 1989

Time to Market in new product development
Time to Market in manufacturing
The VME High Security Option
Security aspects of the fundamental association model
An introduction to public key systems and digital signatures
Security classes and access rights in a distributed system
Building a marketeer's workbench: an expert system applied to the marketing planning process
The Knowledge Crunching Machine at ECRC: a joint R&D project of a high speed Prolog system
Aspects of protection on the Flagship machine: binding, context and environment
ICL Company Research and Development Part 3: The New Range and other developments

Vol.7 Iss.1 - May 1990

Architecture of the DRS6000 (UNICORN) Hardware
DRS6000 (UNICORN) software: an overview
Electromechanical Design of DRS6000 (UNICORN)
The User-System Interface - a challenge for application users and application developers?
The emergence of the separable user interface
SMIS - A Knowledge-Based Interface to Marketing Data
A Conversational Interface to a Constraint-Satisfaction System
SODA: The ICL interface for ODA document access
Human - Human co-operation and the design of co-operative mechanisms
Regulatory Requirements for Security - User Access Control
Standards for secure interfaces to distributed applications
How to Use Colour in Displays - 1. Physiology Physics & Perception

Vol.7 Iss.2 - November 1990

The SX Node Architecture
SX Design Process
Physical Design Concepts of the SX Mainframe
The Development of Marketing to Design: The Incorporation of Human Factors into Specification and Design
Advances in the Processing and Management of Multimedia Information
An Overview of Multiworks
RICHE-Rèseau d'Information et de Communication Hospitalier Européen (Healthcare Information and Communication Network for Europe)
E.S.F - A European Programme for Evolutionary Introduction of Software Factories
A Spreadsheet with Visible Logic
Intelligent Help - The Results of the EUROHELP Project
How to use Colour in Displays - Coding, Cognition and Comprehension
Eye Movements for A Bidirectional Human Interface
Government IT Infrastructure for the Nineties (GIN): An Introduction to the Programme

Vol.7 Iss.3 - May 1991

Introduction to the technical characteristics of ISDN
ISDN in France: Numéris and its market
The Telecoms Scene in Spain
Future Applications of ISDN to Information Technology
A Geographical Information System for Managing the Assets of a Water Company
Using Constraint Logic Programming Techniques in Container Port Planning
Locator - An Application of Knowledge Engineering to ICL's Customer Service
Designing the HCI for a Graphical Knowledge Tree Editor: A Case Study in User-Centred Design
X/OPEN - From Strength to Strength
Architectures of Database Machines
Computer Simulation for the Efficient Development of Silicon Technologies
The use of Ward and Mellor Structured Methodology for the Design of a Complex Real Time System

Vol.7 Iss.4 - November 1991

Systems Management: A Challenge for the Nineties - Why now?
The Evolution within ICL of an Architecture for Systems Management
Manageability of a Distributed System
Distribution Management - ICL's Open Approach
Experience of Managing Data Flows in Distributed Computing in Retail Businesses
Generation of Configurations - a Collaborative Venture
Operations Management
OSMC: The Operations Control Manager
The Network Management Domain
An Overview of the Raleigh Object-Oriented Database System
Making a Secure Office System
Architectures of Knowledge Base Machines
The Origins of PERICLES - A common on-line Interface

Vol.8 Iss.1 - May 1992

Defining CASE Requirements
ICL's ICASE Products
The Engineering Database
CASE Data Integration: The Emerging International Standards
Building Maintainable Knowledge Based Systems
The Architecture of an Open Dictionary
The Use of a Persistent Language in the Implementation of a Process Support System
ALF: A Third Generation Environment for Systems Engineering
MASP/DL: The ALF Language for Process Modelling
The ALF User Interface Management System
A New Notation for Dataflow Specifications

Vol.8 Iss.2 - November 1992

Open Networks - The Key to Global Success
Infrastructure of Corporate Networks in the Nineties
Broadband Networking
FDDI - The High Speed Network of the Nineties
The Evolution of Wireless Networks
Communications Technology for the Retail Environment
RIBA - A Support Environment for Distributed Processing
Information Technology: Support for Law Enforcement Investigations and Intelligence
Standard for Keyboard Layouts - The Origins and Scope of ISO/TEC 9995
ESS - A Solid State Disc System for ICL System for ICL Series 39 Mainframes

Vol.8 Iss.3 - May 1993

An Introduction to OPEN*framework*
The Evolution of the OPEN*framework* Systems Architecture
Creating Potential-for-Change
OPEN*framework* in Action at DEVETIR
Strategic Information Systems planning: A Process to Integrate IT and Business Systems
Describing Systems in the OPEN*framework* Integration Knowledge Base
Multimedia and Standards for Open Information
VME-X: Making VME Open
A New Approach to Cryptographic Facility Design
CHISLE: An Engineer's Tool for Hardware System Design
Distributed Detection of Deadlock

Vol.8 Iss.4 - November 1993

Toward the 4th Generation Office: A Study in Office Systems Evolution
IPCS - Integrated Product Configuring Service
CGS - The ICL Configurer Graphics Service
Location Transparency in Heterogeneous Networks
Future Office Interconnection Architectures for LAN and Wide Area Access
Parallel Lisp and the Text Translation System METAL on the European Declarative System
Detecting Latent Sector Faults in SCSI Disks

Vol.9 Iss.1 - May 1994

Client-server architecture
How ICL Corporate Systems support Client-server: an ArchitecturalOverview
Exploiting Client-server Computing to meet the needs of Retail Banking Organisations
A practical example of Client-server Integration
From a Frog to a Handsome Prince: Enhancing existing character based mainframe applications
Legacy systems in client-server networks: A gateway employing scripted terminal emulation
The Management of Client-server Systems
Dialogue Manager: Integrating disparate services in client-server environments
Distributed Printing in a Heterogeneous World
Systems Management: an example of a successful Client-server Architecture
PARIS - ICL's Problem & Resolution Information System

To order back issues

Contact

ICL *SystemWise* Help Desk
D2D House, Manchester Road, Ashton-u-Lyne, Lancs OL7 OES, UK
Telephone +44 (0)181 565 7993 or +44 (0)161 371 0208 x2772
Fax +44 (0)161 371 9164
(ICL speedcall 7993)

Ingenuity

The ICL Technical Journal

Guidance for Authors

1. Content

Ingenuity, the ICL Technical Journal, has an international circulation. It publishes high standard papers that have some relevance to ICL's business. It is aimed at the general technical community and in particular at ICL's users and customers. It is intended for readers who have an interest in the information technology field in general but who may not be informed on the aspect covered by a particular paper. To be acceptable, papers on more specialised aspects of design or application must include some suitable introductory material or reference.

Ingenuity will usually not reprint papers already published but this does not necessarily exclude papers presented at conferences. It is not necessary for the material to be entirely new or original. Papers will not reveal matter relating to unannounced products of any of the ICL Group companies.

Letters to the Editor and book reviews may also be published.

2. Authors

Within the framework defined in paragraph 1, the Editor will be happy to consider a paper by any author or group of authors, whether or not an employee of a company in the ICL Group. All papers are judged on their merit, irrespective of origin.

3. Length

There is no fixed upper or lower limit, but a useful working range is 4000-6000 words; it may be difficult to accommodate a long paper in a particular issue. Authors should always keep brevity in mind but should not sacrifice necessary fullness of explanation to this.

4. Abstract

All papers should have an Abstract of not more than 200 words, suitable for the various abstracting journals to use without alteration.

5. Presentation

5.1 Printed (typed) copy

Two copies of the manuscript, typed 1½/2 line spacing on one side only of A4 paper, with right and left margins of at least 2.5cms, and the pages numbered in sequence, should be sent to the Editor. Particular care should be taken to ensure that mathematical symbols and expressions, and any special characters such as Greek letters, are clear. Any detailed mathematical treatment should be put in an Appendix so that only essential results need be referred to in the text.

5.2 Disk version

Authors are requested to submit a magnetic disk version of their copy in addition to the manuscript. All artwork and diagrams to be supplied in their original source format. The Editor will be glad to provide detailed advice on the format of the text on the disk.

5.3 Diagrams

Line diagrams will, if necessary, be redrawn and professionally lettered for publication, so it is essential that they are clear. Axes of graphs should be labelled with the relevant variables and, where this is desirable, marked off with their values. All diagrams should have a caption and be numbered for reference in the text and the text marked to show where each should be placed - e.g. "Figure 5 here". Authors should check that all diagrams are actually referred to in the text and that all diagrams referred to are supplied. Since diagrams are always separated from their text in the production process, these should be presented each on a separate sheet and, *most important*, each sheet must carry the author's name and the title of the paper. The diagram captions and numbers should be listed on a separate sheet which also should give the author's name and the title of the paper.

5.4 Tables

As with diagrams, these should all be given captions and reference numbers; adequate row and column headings should be given, also the relevant units for all the quantities tabulated.

5.5 References

Authors are asked to use the Author/Date system, in which the author(s) and the date of the publication are given in the text, and all the references are listed in alphabetical order of author at the end.

> e.g. in the text: "...further details are given in [Henderson, 1986]"

with the corresponding entry in the reference list:

> HENDERSON, P. Functional Programming Formal Specification and Rapid Prototyping. IEEE Trans. on Software Engineering SE*12, 2, 241-250, 1986.

Where there are more than two authors it is usual to give the text reference as "[X et al ...]".

Authors should check that all text references are listed; references to works not quoted in the text should be listed under a heading such as *Bibliography* or *Further reading*.

5.6 Style

A note is available from the Editor summarising the main points of style - punctuation, spelling, use of initials and acronyms etc. - preferred for Journal papers.

6. Referees

The Editor may refer papers to independent referees for comment. If the referee recommends revisions to the draft, the author will be asked to make those revisions. Referees are anonymous. Minor editorial corrections, as for example to conform to the **Ingenuity** general style for spelling or notation, will be made by the Editor.

7. Proofs, Offprints

Printed proofs are sent to authors for correction before publication. Authors receive either a hard copy master of their paper or a Word for Windows version in either V2 or V6 on magnetic media.

8. Copyright

Copyright of papers published in **Ingenuity** rests with ICL unless specifically agreed otherwise before publication. Publications may be reproduced with the Editor's permission, which will normally be granted, and with due acknowledgement.

Ingenuity November 1994