Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

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

ANSA Phase III

The compatibility of objects in distributed systems

Yigal Hoffner, Rob van der Linden & Mike Beasley

Paper for ACM SIGPLAN workshop on IDLs

Abstract

The paper sets the context for dealing with the issue of compatibility of objects wishing to co-operate with each other. Limitations of current notions of compatibility and their manifestation in existing IDLs will be exposed and discussed. Issues which are not dealt with by current approaches will be outlined followed by some suggestions concerning future work.

APM.1066.00.03 **Draft** 23 October 1993

External Paper

Distribution:
Supersedes:
Superseded by:

The compatibility of objects in distributed systems

External Paper

The compatibility of objects in distributed systems

Paper for ACM SIGPLAN workshop on IDLs

APM.1066.00.03

23 October 1993

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

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

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

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

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

Contents

1 Introduction

1.1 Summary and background

The paper sets the context for dealing with the issue of compatibility of objects wishing to co-operate with each other. Limitations of current notions of compatibility and their manifestation in existing IDLs are exposed and discussed.

The notion of compatibility is extended to include all the information client and servers have to know about each other in order to co-operate successfully. Compatibility is also generalized to include notions of contracts, accounting and billing and management.

The study is based on a comparison between currently available distributed system platforms [APM.1042.1.93][APM.1042.1.93] such as DCE [OSF 92}, CORBA [OMG 92] variants such as ORBIX[IONA 93a] [IONA 93b], ACAS [DEC 93a] [DEC 93b] and DOME [OOT 93], and ANSAware [ARM 92].

1.2 Audience

The paper will benefit both designers of distributed system platforms and applications looking at object specification issues which are likely to impact distributed systems in the near future.

1.3 The distributed development process

We define the development process to span the specification, design, implementation, creation, run-time, maintenance and destruction of a program.

Early development processes were centralized by nature: the process was driven by a single authority and usually carried out by in a homogeneous environment. The design of software systems reflected this centralized structure: a single main program invoking a variety of sub-routines, all stored on the same computer.

The extension of the subroutine concept by the remote procedure call (RPC) [BIRRELL 84] allowed programs to invoke sub-routines residing on remote computers, thus paving the way to distributed processing.

Essential support for RPC usually includes an interface definition language (IDL) to define the interface between the client and server [OSF 92}, [OMG 92], [IONA 93a], [IONA 93b], [DEC 93a], [DEC 93b], [OOT 93] and [ARM 92]. The ability to define interfaces using IDLs facilitated the implementation of software modules separately from each other. This is an additional advantage of the distributed approach.

Distribution entails not only physically separate run-time components but also the likely de-centralization of the development process. Thus a client and a server may be independently developed by different designers from an agreed well-defined interface, at different sites and times, using different design methodologies, tools and technologies. Furthermore, clients and servers may be developed independently without the starting point of an agreed common interface definition, thus depending on the availability of mechanisms and information to find compatible objects at run-time.

An essential starting point to achieving co-operation between separately implemented objects is being able to compute the relationship between clients and servers in order to determine their compatibility for co-operation. For this it is necessary to have a specification of the objects in a form which can be used for such computation.

1.4 Testing for the compatibility of objects

There are two major aspects of testing for the compatibility of objects:

- WHAT to specify when describing objects. Current models used to describe clients and servers (the basis of the compatibility test) are incomplete.
- HOW to specify what is necessary for describing objects. The languages used to describe the objects may be different, adding another level of complexity to the problem of specification comparison.

Existing approaches to specification of objects omit important aspects of object characteristics. Also no agreement on the form of the specification exists as can be seen from the diversity of IDLs currently available [OSF 92}, [OMG 92], [IONA 93a], [DEC 93a] and [ARM 92]. Achieving at least partial agreement or a mapping between them is needed in order to be able to extend the compatibility test beyond the local context into a heterogeneous distributed environment.

1.4.1 Common assumptions concerning compatibility testing

Most approaches to solving the compatibility problem make one or several of the following assumptions:

- some degree of homogeneity exists in the system. This could manifest itself in contractual convention, compatible accounting/billing or the decision not to have any, common RPC/comms protocols, etc.
- there is agreement either on a single specification language or on a restricted set of specification languages and the mappings between them.
- the specification of the functional aspects of objects is assumed sufficient for compatibility testing.

1.4.2 Compatibility checking today

In current systems such as DCE [OSF 92},CORBA [OMG 92] and ANSAware [ARM 92], the set up for inter-working is based on using two different categories of information:

- functional specification of objects: type signature for type matching.
- comms protocol specification: for selecting the appropriate comms protocol and setting up the binding between the objects.

1.5 Trading, binding and type checking

In ANSA [IONA 93a] and ANSAware [ARM 92] the distinction between the two categories mentioned above is reflected in the difference between the processes of trading and binding. Trading is the process of searching and match making service offers from servers to service requests from clients. A trader is an object which provides the trading process to clients and servers. Binding is the process of allocating resources necessary for the client and server to interact with each other.

Trading involves a server exporting a service offer to the trader. The information passed to the trader includes an interface reference (Ifref), type information, and some properties particular to the service offer. A client imports an interface from the trader by providing the trader with type information of the service required and some properties it wishes the service to have. The trader matches the request with the available offers. A successful match returns an Ifref of the service to the client. An ANSAware Ifref will contains information about all protocols supported by the server infrastructure on behalf of the server.

The binding process entails the client infrastructure looking inside the interface reference, and making a decision as to which protocol to use. There is therefore a clear distinction between trading (done explicitly by the client code and the trader) and binding (carried out transparently by the infrastructure).

The distinction between trading and binding is not as clear in DCE or in the available CORBA compliant platforms such as ORBIX [IONA 93a] and DEC's ACA service [DEC 93a].

DCE [OSF 92}, CORBA [OMG 92] and ANSAware [ARM 92] do not perform type checking on the basis of the type signature. ANSAware simplifies the type information by providing a type name. The trader type checker is based on a programmer asserted type graph which describes the sub-typing relationship between types. This means that trading can only be carried out within system where there is prior agreement on the relation between type names and descriptions. Neither DCE nor CORBA have any facilities for type checking and do not provide the check based on the type relationship which ANSAware provides.

2 The limitations of the client-server approach

Most current systems either ignore or make certain assumptions about their environment. Making those implicit assumptions explicit is an important part of recording design decisions. Often, in order to discover the hidden or implicit assumptions it is necessary to ask the question: “What happens when everything that can be different will be different?”. Any generalization of the client-server model should be able to address the explosion of heterogeneity and also be capable of being simplified where optimization is possible [APM.1020.1 93].

The information exchanged between client and server before co-operation can take place requires extensions in order to make it sufficiently general to be appropriate in a large scale distributed system. The extensions necessary concern:

- contractual compatibility
- incomplete object descriptions
- management set-up compatibility
- accounting and billing compatibility
- peer to peer interactions.

2.1 Contractual compatibility

The basis for any co-operation in a distributed system will assume either an implicit or explicit notion of contract between the authorities of the co-operating objects.

Thus, a contract will have to contain all the aspects of object specification which require agreement: functional, and technical specification, as well as management, accounting and billing specification.

2.1.1 Contracts and object specifications

In large scale distributed systems, the specification of an object has to be anchored in some *legal* context which describes what undertaking exists to ensure that the object actually delivers its promise, as well as what happens if it does not. The legal context is expected to express issues of accountability, liability and responsibility.

The legal context in which services are offered and consumed can influence confidence and set expectations of service producers and consumers.

The technical and administrative specification of an object says what an object can do. Such a specification leaves out a number of important issues:

- what an object can do is not necessarily what it will do, for example:
  - management may interfere

  - specification may be inaccurate
  - failure may occur
  - dependency on other objects and resource allocation constraints may cause problems
- what happens if the object fails to provide the promised service. This usually includes some penalty clauses
- cost of using a service: the way in which accounting and payment is expected to be made.

The information concerning these issues has to be made available to allow objects to determine whether they can co-operate with other objects. As stated specified above, a functional point of view is insufficient.

2.1.2 Contracts

Contracts provide an implicit or explicit legal relationship between two entities.

The legal context can take many forms:

- statement of abdication of liability: see freeware declaration for example
- agreement by convention (common law) or prior understanding: within areas of agreement, it is not necessary to explicitly enter a contract by, for example, signing an agreement or resorting to lawyers
- proper contract where all aspects of co-operation are stated.

The contract includes on both sides a statement of:

- what is to be provided - a description of the object
  - this may be incomplete if it depends on the run-time environment, for example. In such cases the contract should specify whether it is complete or partial and which parts are left open
  - on the behaviour of the co-operating objects
- what is the liability in case the guarantees are not met
- what is required of co-operating object.

2.1.3 Contracts and the distributed development process

Contractual agreement between objects wishing to co-operate can be reached at different times in the life of an object:

- prior to the development process
- during the development process
- at the pre-processing and compilation process
- at instantiation time
- at run-time.

There are two approaches to achieving compatibility between co-operating objects:

- develop objects specified to co-operate with each other by ensuring co-operation between the design teams during the development processes. This can be achieved:
  - by operating in a closed environment

— by reaching agreement before the design and implementation stages.

- discover a compatible object (either existing or that can be created on demand) by going through a match making process.

To-date, most developments in distributed systems operate under closed world assumptions. This also means that re-use is limited to the islands of homogeneity because of:

— technical incompatibility

— inaccessibility to information on objects

— lack of essential information as these assumptions are usually not stated explicitly.

This leads to an extensive duplication of effort. Ultimately, developing objects specially from scratch is likely to become too expensive to be widely used except where absolutely necessary. This approach will probably become limited to situations where some quality of service requirement is critical and special optimization relying on closed world assumptions are necessary. In such a case it may be necessary to ensure the compatibility between them by constraining their development process. This can be done by reaching a contract before the development process of one or the other starts. The contract will then be about creating a compatible object, rather than trying to find a matching object at run-time.

Objects in a large scale distributed system will usually attempt to find other objects and check for their compatibility at run time. Searching for a compatible object at the latest possible opportunity so as to be able to get the best match provides applications with considerable flexibility.

2.1.4 Contractual match making

Where the legal framework is of importance, match making between objects who wish to co-operate has thus to take place not just at the level of technical and administrative specification, but also to ensure that the legal aspects also match in terms of guarantees, responsibilities and liabilities.

2.2 Incomplete object descriptions

Available systems concentrate on the specification of objects using IDL. Object compatibility is thus at best reduced to comparing interface type signatures. This approach is insufficient when wishing to achieve compatibility with different degrees of co-operation.

2.2.1 Degrees of co-operation

There are different degrees of co-operation between objects (Figure 2.1):

- Inter-connection: data transfer in terms of comms protocols
- Inter-working: remote procedure call (RPC) includes location and access transparencies
- Inter-operability: additional transparencies and quality of service guarantees such as dependability and performance
- Integration: application semantics compatibility.

Figure 2.1: Degrees of co-operation between objects

Application	**Integration**	**Application**
Transparency	**Inter-operability**	**Transparency**
RPC	**Inter-working**	**RPC**
Comms	**Inter-connection**	**Comms**

2.2.2 Classification of information necessary for co-operation

The information required to establish successful co-operation can be classified into 6 categories (Figure 1):

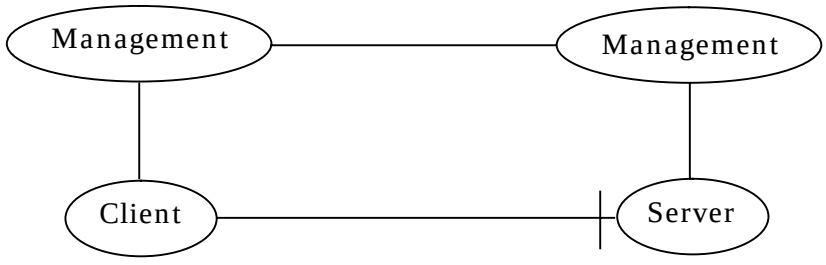
- Semantics: a description of the application in enterprise terms
- Type (signature): usually using IDL's to specify interface signature
- Quality of service (QOS): specification of the guarantees required of the client and server in terms of dependability, security, performance, etc.
- Transparency mechanisms: the mechanisms and procedures used in order to provide the QOS guarantees
- Comms protocols: the specification of the information and its structure necessary for the transparency mechanisms to achieve their objective and also provide RPC capability
- Address: the destination address of the communication link.

2.3 Management set-up compatibility

By management we refer to the procedures and structures necessary to ensure guarantees of service such as: dependability, security and performance. Other examples are management structures and procedures necessary for monitoring and debugging across system boundaries.

Different aspects of the co-operation between objects require management structures to be set up and maintained before, during and possibly after the managed entities co-operate (Figure 2.2). The interactions between the management of the co-operating objects can take place in different epochs and does not have to be constrained to the run-time epoch.

Figure 2.2: The relationship between management and co-operating objects

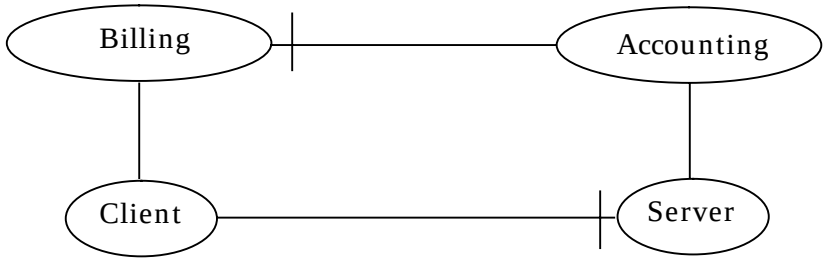


2.4 Accounting and billing compatibility

Different system may support different models of accounting and may have different notions of payment. The definition, structures and procedures which concern accounting and billing in co-operating objects (Figure 2.3) will have to be checked to be compatible.

Accounting and billing activities may actually precede the run-time epoch of the co-operating objects. For example payment for services may take place before the service can be consumed.

Figure 2.3: A general model of client-server accounting and billing



The need to include accounting and billing in the trading process requires agreement on the accounting and billing requirements in terms of procedures. For example, in what form how and when is the bill sent to the client, when and how is the client supposed to pay the measurement of work and payment per unit of work compatible billing interfaces.

2.5 Peer to peer interactions

One of the limitations of the client-server model is the asymmetric and synchronous nature of the interactions between the objects. This was the result of the RPC being developed as an extension to idea of the subroutine, where the calling program initiated the branching and was suspended until the subroutine finished. Many application in the telecomms world, for example, are based on a different model where the relationship is symmetric and asynchronous. This topic is outside the scope of this paper.

3 Summary and future directions

Large scale distributed systems will inevitably require different development teams to interact with each other. This will be done by using repositories which will contain specifications of objects developed independently of each other.

The development environments will have to support and accommodate:

- communication between different development teams/methods
- different development methods:
  - different notions of object specification
  - multiple specification languages
  - different uses of the specifications
- support for the use of extended specifications at different development epochs
- different notions of trading and relation to object specification repositories.

The process of trading will have to become more general involving more than just functional and technical information, carried out in different ways, at different times by different people and processes.

References

[ANSA 91]

ANSA: A Systems Designer's Introduction to the Architecture, APM Ltd., Cambridge U.K., April 1991.

[BIRRELL 84]

Birrell, A. & Nelson, B., "Implementing Remote Procedure Calls", ACM Transactions on Computer Systems, 2(1), 39-59, February 1984.

[DEC 93a]

'DEC ACA Services: System Integrator and Programmer Guide', DEC, Maynard, Massachusetts, USA, 1992.

[DEC 93b]

'DEC ACA Services: Reference Manual', DEC, Maynard, Massachusetts, USA, 1992.

[IONA 93a]

'Orbix: Programmer's Guide', Iona Technologies Ltd., Dublin, Republic of Ireland, 1993.

[IONA 93b]

'Orbix: Advanced Programmer's Guide', Iona Technologies Ltd., Dublin, Republic of Ireland, 1993.

[OOT 93]

'DOME: Introduction and Reference Manual', Object-Oriented Technologies, Leamington Spa, 1993.

[APM.1020.1 93]

Otway, D.J., "Abstract and Automate", Architecture Projects Management, Cambridge, 1993.

[APM.1021.1 93]

Herbert, A.J. et al., "ORB Interoperability", Architecture Projects Management, Cambridge, 1993.

[ARM 92]

ANSAware 4.0 Reference Manual, Architecture Projects Management, Cambridge, 1992.

[OMG 92]

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

[OSF 92}

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

[APM.1042.1.93]

Hoffner, Y. and Beasley M. D. R., “Type Conformance and IDLs”, Architecture Projects Management, Cambridge, 1993.

[APM.1048.1.93]

Beasley M. D. R., “Comparison of CORBA Compliant Platforms”, Architecture Projects Management, Cambridge, 1993.