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

Advanced Trading

Mike Beasley

Abstract

This paper takes the Trading Service as a starting point, and adds, layer by layer, resource management, type checking, CORBA repositories and interception on top.

APM.1091.00.01 **Draft** 8 December 1993

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

Distribution:
Supersedes:
Superseded by:

Copyright © 1993 Architecture Projects Management Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.

Advanced Trading

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

Advanced Trading

Mike Beasley

APM.1091.00.01

8 December 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

Copyright © 1993 Architecture Projects Management Limited
The copyright is held on behalf of the sponsors for the time being of the ANSA Workprogramme.

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

Contents

1 Introduction

This paper came into existence as a result of discussing how to bring together trading, object management, type checking, repositories and interceptors. The motivation for this work is to be found in [APM.1069.0 93].

It attempts to bring together:

- architectural reports, notably [APM.1005.1 93]
- the ANSAware implementation of trading, node managers and factories - see [ARM]
- BNR's 'Integrated Trading and Object Management', which is in their ODS platform and the ICL DAIS product
- the CORBA products Orbix - [Iona 93a] and [Iona 93b]
- various input to OMG - JOSS ([OMG 93a] and [OMG 93b]) and a pre-JOSS proposal from ICL and BNR [ICL 93]

The intention is that it will form the basis of the High Level Design of some of the experimental prototypes that will be produced.

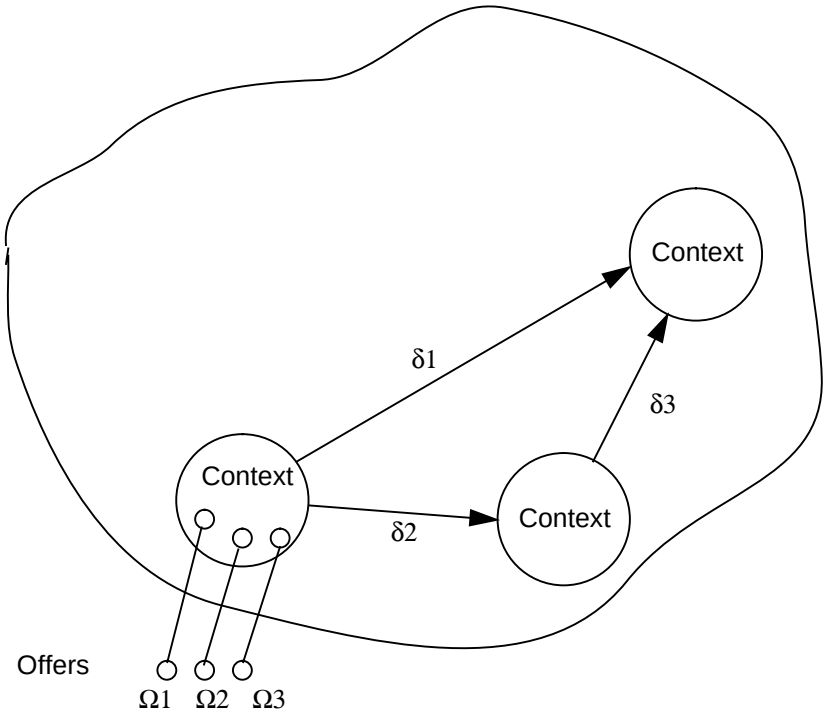
2 Trader

2.1 Trading in ANSA/ANSAware

The starting point is the trader - see [APM.1005.1 93] and [ARM]

Figure 2.1 below shows a trader and its contexts, offers and properties.

Figure 2.1: Trader with Contexts, Offers and Properties



Examples of properties might be: distance in the context graph (δn in the figure) or owner of an offer (Ωn in the figure). You can perform an import operation, specifying (for example) a maximum distance in the context graph and a specific owner. However, the trader as shown is only able to tell prospective clients about servers that are already running.

Some people take the view that contexts are 'just another property'.

CORBA-compliant products, such as Orbix ([Iona 93a] and [Iona 93b]) and ACAS ([DEC 93a] and [DEC 93b]), are generally deficient in terms of trading. Typically they have a daemon (control server) which allows you to find a server on the same node, but no contexts or properties. Type checking is based on an exact match of the name. This use of daemons lacks the scalability of the ANSAware trader; typically you have to search a circular list of nodes, rather than going straight to the trader.

One attractive feature of such products, however, is the ability to start a server dynamically. We want the best of both worlds; the advantages of the ANSAware trader, combined with dynamic starting of servers. The next chapter will investigate how such a desirable state of affairs can be arrived at.

2.2 Trading in ITOM

BNR's Integrated Trading and Object Management (ITOM) forms the basis of the joint submission to OMG with ICL [ICL 93].

The distinctive features of the BNR trader are:

- computationally, each trading context is a separate trader object (though in the engineering projection they may not be separate; indeed one would expect them to be implemented as the same code in the same capsule but with different state).
- trader contexts form a directed cyclic graph. Cycles are allowed within an (engineering) trader and between such traders. There is no cycle detection algorithm yet.
  Note that the ANSAware documentation does not mention whether cycles are allowed; it looks as if they are not checked for and would cause chaos if any were set up.

Operations supported by the trader interface are:

- 'resolve': takes the name of another trader (context) and returns an interface reference to it.
- 'register/export': similar to ANSAware; takes a type, properties and context name, and returns an ID which can be used to delete.
- 'lookup/import': takes a context name, type, constraints and preferences and returns a set of offers. Constraints cannot include superlatives; anything involving a superlative must be a preference. Preferences modify the trader policy (if allowed to); traders incorporate their policies into the preferences which they pass to each other.
- 'delete': withdraws an offer, using the ID which came back from the 'register' ('export') call.

3 Monitors and Resource Management

3.1 Introduction

At the end of the previous chapter we observed that products such as Orbix and ACAS allow servers to be started automatically, and asked how we might combine this with ANSA trading.

Work in this area has been done in various places:

- APM (section in [APM.1005.1 93] entitled 'Dynamic Services'.
- BNR, in a joint submission to OMG with ICL (see [ICL 93]).
- The University of Kent, who used the same approach to construct a trader monitor which drew pictures.

The idea is that there is a special kind of offer - called a monitored offer or a proxy offer. An attempt to import it results in the invocation of a monitor, which can then return an interface reference to the trader, which returns it to the client.

This is shown in diagrammatic form in figure 3.1.

It has been suggested that there might be an ordinary offer associated with the proxy offer, and that the monitor could say 'no', 'no but I can get you one' or 'yes'. However, this approach does not seem to have been thought through in adequate detail yet. The questions still remain:

- how should the trader handle the ordinary offer ? (perhaps it could always invoke the proxy mechanism if proxy offers are present)
- how is the association between the proxy offer and the ordinary offer established and maintained ?

It does have the desirable feature, however, that servers could work in the same way whether they are created dynamically or started explicitly.

Note: This needs further discussion.

3.2 Existing Implementations

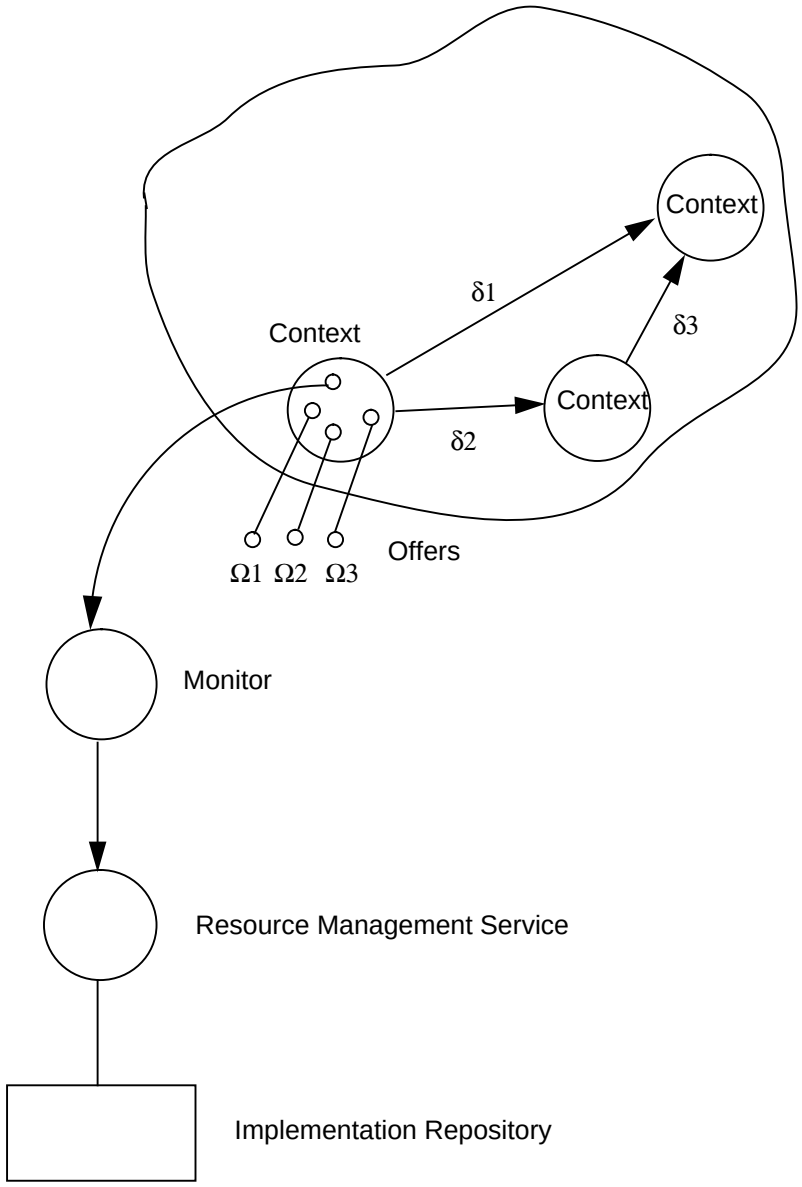
3.2.1 ANSAware: Node Manager and Factory

The ANSAware node manager (see [ARM]) supports dynamic service creation, by using the trader's proxy offer mechanism (the same interface as is used by federated traders).

The whole story goes something like this (see also Figure 3.2):

- the node manager makes a proxy offer to the trader, using the same trader interface and operation as federated traders use, and it supplies its own interface reference to be used when the offer is resolved.
- a client attempts an import operation.

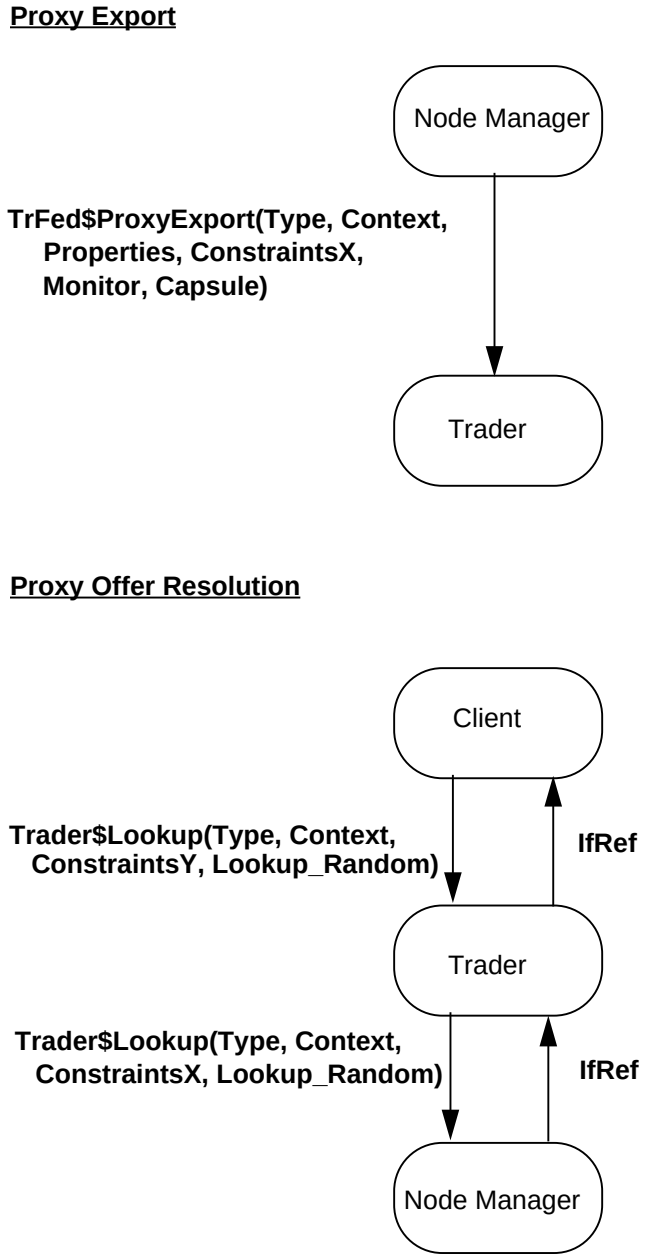
Figure 3.1: Monitor and Resource Management



- the trader forwards the offer to the node manager. The node manager has an operation 'Lookup' which has the same signature of the corresponding operation in the trader, and this is the operation which the trader invokes to forward the offer.
- the node manager creates a server capsule (if necessary) using the factory, and instantiates an object within the new capsule.
- the node manager then returns the interface reference to the trader.
- the trader passes this interface reference to the original client.

Note that the 'constraints' argument passed to the proxy is that which it originally supplied to the trader when making the proxy offer. It can use it how it likes to find the interface reference within its own data structures; it is

Figure 3.2: Node Manager implementation



a string which is only ever touched by the node manager. This is quite a clever trick, but unfortunately not documented in the comments in the code !

However, it is not ideal, from the point of view of implementing monitors, to have to use a 'Trader Federation' interface; the monitor has to remember what information it put in the proxy offer, and reproduce it in response to a 'Lookup' call. It would be much easier if there was a 'resolve proxy offer' operation which just took some sort of identification of the offer (a numerical id would be perfectly adequate) and returned an interface/object reference. The offer identification would then replace the 'constraints' string in the arguments to the 'ProxyExport' operation. An alternative to the use of offer identifications is to make offers into objects in their own right; it would then be possible to invoke operations such as 'withdraw', 'modify' and 'resolve proxy' on them.

An alternative approach that has been suggested that monitoring could be implemented a bit like relocation is in ANSAware; the monitor could export an ordinary offer to the trader, with a null interface reference, and the monitor as a relocator. When the interface reference is first used, the relocation mechanism is invoked, and a new interface reference comes back without the 'null'. This approach is quite attractive, and could lead to a unified treatment of passivation, migration and dynamic service creation via the relocator mechanism. Unfortunately the approach is somewhat dependent on the structure of ANSAware interface references; it is not easy to apply it to CORBA-compliant products. An additional objection is that the proxy has no access to the constraints, which prevents its use for federation of traders.

3.2.2 BNR ITOM

ITOM manages without posting proxy offers. If an import fails, intelligent proxy code in the client can optionally invoke the Object Manager (whether it actually does so or not is a configuration issue). The 'create' call has similar arguments to the 'import', with the addition of an 'initialisation list'.

There is a directed acyclic graph of Object Managers (allowing cycles would lead to too much confusion). These include Node Managers, which control Capsule Managers. A Node Manager can create a new capsule.

A trader can be federated to an Object Manager. One of the properties on the link is a list of the types that the Object Manager can handle. This is implemented in a 'flexible static' way; the addition of capsule templates will ripple up from the Node Manager through the system.

When an object is created, no automatic trader export is done, in accordance with the principles that objects manage themselves and that the trader is not (very) special.

3.2.3 Orbix Daemon and ACAS Control Server

The daemon (control server) is the nearest thing to a trader that there is, though there is one per node (so it includes some node manager/factory functionality too), and there are no contexts or constraints, neither is there any concept of a conformant type being good enough (no type manager). Because of these simplifications (and, in Orbix at least, the restriction that only one instance of any server one server may run on any host), the daemon is able to combine the functionality of the trader, node manager and factory in ANSAware (which includes the monitor as described above). See the Orbix and ACAS documentation ([Iona 93a], [Iona 93b], [DEC 93a] and [DEC 93b]) for more detail.

3.3 More on Resource Management

There are two interesting papers about Resource Management in the Cambridge Ring - [Herbert] and [Craft 85].

Note: The first of these is not yet proved to be interesting, as I (mdrb) have not yet seen it !

Resource management raises all sorts of questions; suppose you have a number of different hosts capable of supporting a specific service, and each server is capable of supporting some number of connections:

- do we start a new server for any particular request ?

- if so, which host do we start it on ?
- if not, which running server do we connect the request to ?
- can we activate a passive server ?
- should we move an active server from one node to another ?

The resource management must contain a centralised element and a per-host element. We could have a threshold beyond which a server, though able to handle further requests if necessary, would prefer not to do so; this threshold may differ between hosts. The threshold may be more dynamic (a server decides that it's full based on consumption of some resource rather than on number of connections).

The whole discussion above assumes the concept of a connection is well-defined; currently this is not the case. However:

- the concept of 'path expressions' (but see [APM.1010.0 93]) could be developed into connections;
- the ANSA transactions work (see [APM.1004.0 93]) could lead to a concept of 'connection';
- the life of an interface instance could be regarded as a connection.

The monitor interface talks to a resource management service, which is an active implementation repository.

Note: There are some interesting old papers in this sort of area by Mike Olsen, Dennis Nyong, rvdl and others. These should be analysed.

3.4 The JOSS Proposals

How does this all relate to OMG's 'object services' (see [OMG 93a] and [OMG 93b]) ?

JOSS has Factory Finders, which can use a Name Server or a Trader to find Factories. Factories can be either generic or object-specific.

Factories are similar to ANSAware factories (see [ARM]), but they do the whole job of creating an object, some of which is done in ANSAware by the Capsule interface and the Object interface. In JOSS, you just need to invoke the 'create_object' operation on a factory.

Note: According to ajw, JOSS ignores the interesting bits. This statement should be made a bit more formal !

4 Types

We want to bring types into this picture, starting by asking questions like ‘is this type compatible with that one ?’.

4.1 What is a Type ?

IDLs have two different kinds of type:

- a *data type*, which can be a *basic type* (such as `integer`), or a type constructed from basic types using such constructors as `array`, `record`, `sequence` and so on.
  These types have no explicit list of the operations that can be performed on them, and are passed around by value rather than by reference.
- an *interface type*, which has an IDL definition in terms of the operations that can be performed on it.

Most of the following discussion is concerned with interface types.

4.2 Type Repositories

The *ODP Type Repository* (see [ISO 93]) has an interface called ‘type’ with two operations:

- ‘is this type compatible with that one ?
- ‘getschema’ - return the IDL or equivalent. (This operation can be thought of as a private operation, performed by one type on another type as part of the compatibility check; it is also useful in specialised clients such as browsers).

As an optimization (to avoid calculating everything at run time) we have a *type library*, which is a lattice of pre-computed and asserted type compatibility relationships. Asserted relationships should be checked, and rejected if they fail a conformance check.

But what happens when you attempt to federate two such type libraries together ?

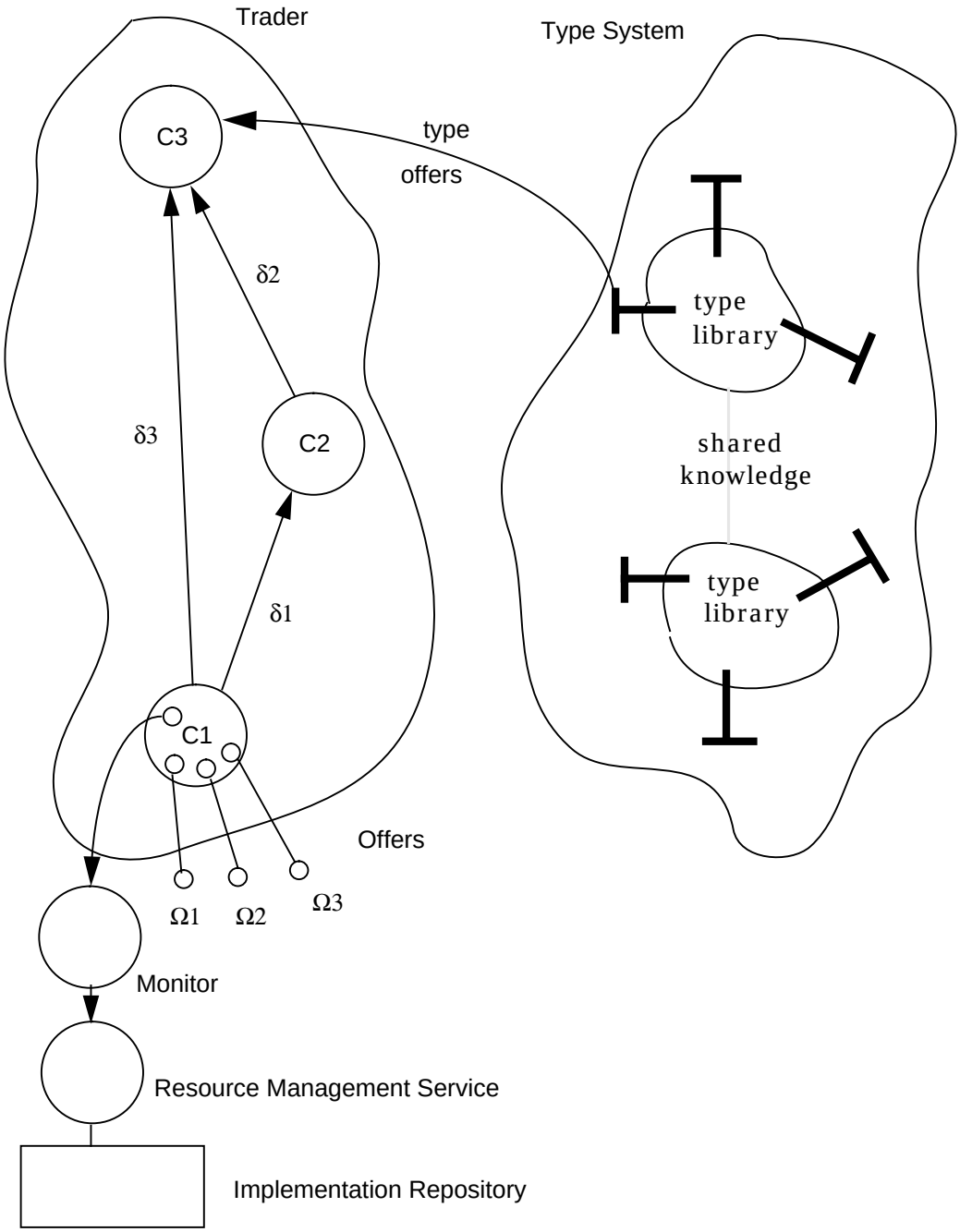
Again, if you do not want (for the sake of efficiency) to calculate the type relationships at run time, you can introduce a librarian, which allows you to introduce foreign types and assert various compatibility relationships. This builds up shared knowledge.

Figure 4.1 below shows this whole picture.

A type consists of three parts, which can be thought of (see [RC.258.03 92]) in terms of projections:

- signature (computational view).
- behaviour/semantics (information view).

Figure 4.1: Trading and Types



- environmental constraints (enterprise view ? engineering view ?).

Checking compatibility of signatures is fairly well understood; behaviour is more difficult, though the use of pre- and post- conditions can often be useful; environmental constraints are/include QoS attributes.

It is not in general possible to prove two specifications to be equivalent (the halting problem). A consequence of this is that there is a line to be drawn somewhere between what can be checked and what must be asserted.

When someone asks for a type with a given signature, behaviour and environmental constraints, then everything which was explicitly asked for

must match. Anything that was not explicitly asked for is irrelevant to the matching process.

Our Type Repository is close to the CORBA Interface Repository (see [OMG 92]), though CORBA is computational only, and incomplete in that various things need to be built on top of it, e.g. a conformance checker, to get the full computational type system. Note that the CORBA Interface Repository includes data types, which are defined in the IDL using `typedef`.

Note: We should say more about data types.

We can introduce a context into the trader with offers of type 'type', pointing at the type library, with various interesting properties.

People want to ask questions like 'I want to do banking. What kinds of interface can I use ?' (trading for types) and we're heading the right way to getting a system in which questions like this can be asked and answered (discovery).

4.3 Types in APM.1005

[APM.1005.1 93] introduces two services relating to types: a Type Description service which contains two operations 'getschema' and 'checkconformance' as above, and a Type Conformance service which does conformance checking independently of a Type Description service.

It also introduces the idea of a type being an interface (with the two operations 'getschema' and 'checkconformance', which are called 'GetDescription' and 'ConformsTo'). The provision of a type library is done by exporting the interface references for types to the trader, and putting them all in the same context.

When exporting to the trader, a type is represented by an interface reference to an AbstractType, rather than merely by a name.

4.4 Existing Implementations

4.4.1 ANSAware

In ANSAware (see [ARM], all type relationships are asserted (except that all types are implicitly compatible with the empty type 'ansa' and the type 'Management'). Types form a Directed Acyclic Graph. The trader has an interface 'TrType' with operations 'Add', 'Mask', 'Unmask', 'Del' and 'List', and there are clients 'typecl' (command line) and 'typeMgr' (graphical browser) to allow the type space to be inspected and updated.

ANSAware IDL also has a distinction between two kinds of type: on the one hand, built-in types such as integer together with constructed types such as structures, unions, arrays; on the other, interface types which are defined explicitly in terms of their operations.

When exporting to the trader, a type is represented by its name.

The problems with the ANSAware approach are well-known:

- The type relationships in the trader's type space have to be maintained in a consistent state with the IDL ('is compatible with' and actual compatibility of signatures).

- It all depends on type names, and thus does not federate across different type spaces.

4.4.2 DPL

DPL, a language which has been used for research purposes at APM (see [APM.1014.0 93], [APM.1015.0 93] and [RC.339.02 92]) has proper conformance-based type checking, and is therefore closer to the architectural principles than is the ANSAware implementation.

The problems with the DPL type system are addressed in [RC.339.02 92] and [RC.258.03 92]:

- there are performance problems because *all* types (including 'integer') are defined in terms of their operations.
- there are problems with generic factories and with trading (defining the type which is returned by a factory 'create' or a trader 'import').
- the definition of conformance needs to be modified in cases of self-reference; it is too strict to be useful (for example, you want to do a conformance test on the basis that a type 'integer' has an operation 'add' with parameters of type 'itself', rather than of type 'integer').
- it does not address non-computational issues such as semantics.

4.4.3 BNR ITOM

There is a separate type manager object which is shared by the trader and the Object Manager (LifeCycle service). Types have not yet been implemented as interfaces, though BNR have no opposition to the principle. A simple default type manager is built in to the trader, for use if no external type manager exists: it does a straightforward string comparison.

4.5 JOSS

The ICL/BNR proposal [ICL 93] in response to OMG's RFP 1 also includes a proposal for the Object Trader Service (though this was not actually requested by the RFP).

The trader interface has operations 'register_offer' and 'lookup' in which the types are represented as object references rather than names. It suggests that there should be a Type Manager which performs conformance-based type checking using information from the Interface Repository.

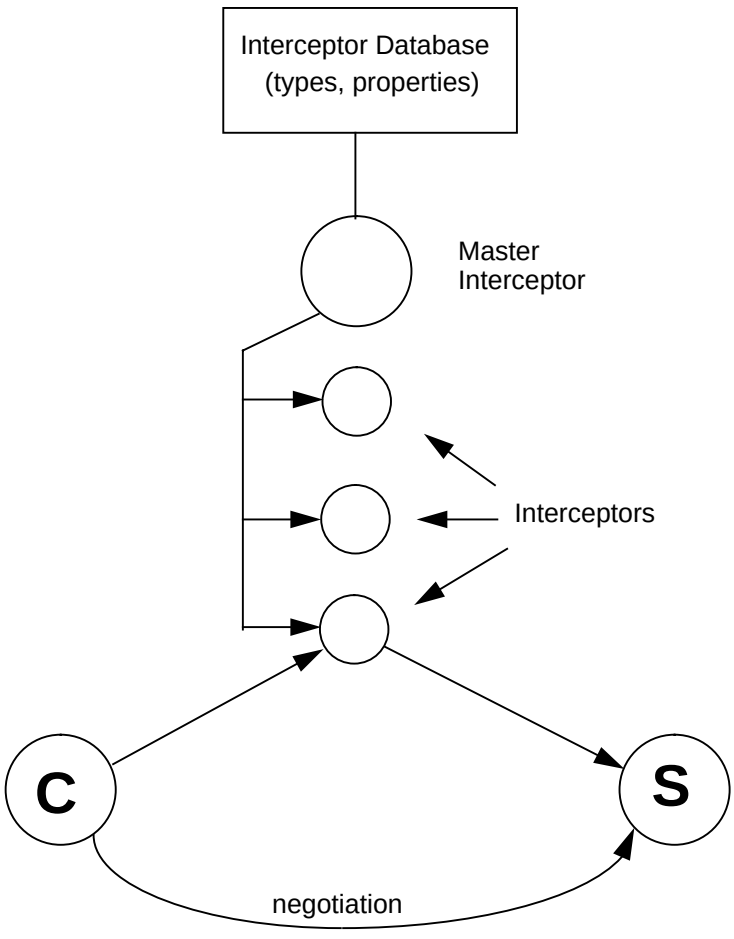
5 Interceptors

We now want to introduce interceptors for situations in which operation/ termination names, property names, type descriptions and other things need to be mapped.

A good example with property names is where the property names ‘local’ and ‘remote’ are context relative, and they have to be mapped one to the other on crossing a boundary.

If we have a ‘master interceptor’ which stands in front of the trader, this can instantiate gateways as required for particular conversions to be done. We can remove the ‘introduction’ function from the type library, and use interceptors instead. The figure below shows this.

Figure 5.1: Interceptors



6 Implementation of the above

Any implementation would need to be in terms of current technology. For example, the use of Orbix and C++ would get us off to a good start with the type checker, building on the existing interface repository in Orbix.

6.1 A Re-Engineered Trader

The trader could be re-engineered in a more modular way.

This would enable different versions to be built quite easily, for example an Orbix trader using an SQL database for its persistent storage. (Also X.500 directories, etc.)

6.1.1 Why ?

We are talking about bringing together trading, resource management, type management, CORBA repositories and interception. We are also talking about using 'real' (SQL) databases. Combining the interface repositories from Orbix with the ANSAware trader suggests that we ought to have an Orbix trader with interfaces to it defined in CORBA IDL.

6.1.2 How ?

The trader would need to be more modular to meet these varied requirements. For example, it could be separated into:

- RPC code (including everything that's IDL-dependent)
- database code
- context name management
- offer management
- property and constraint management
- federation of traders
- thread management (including mutual exclusion and all that sort of thing)

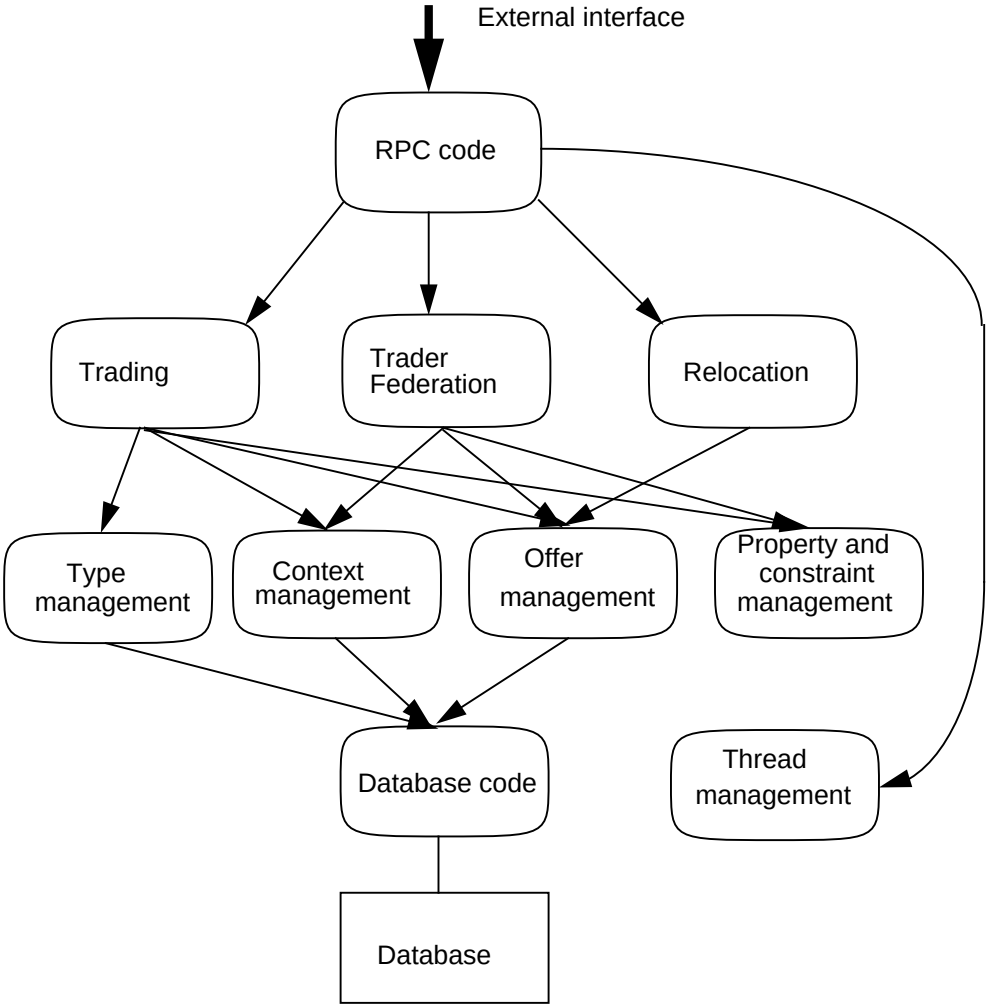
Two other components, which are computationally separate but co-located from an engineering point of view in the current ANSAware trader:

- type management (basic)
- relocation

The above modularisation is shown in figure 6.1 below.

The idea of implementing any new, or substantially changed, code in C++ seems attractive; it should help us to get the modularity we need.

Figure 6.1: Modularised Trader



6.1.3 What are the problems ? How do we solve them ?

1. The difference between CORBA object references and ANSA interface references.

   Not only are the internals different, but the information is at a different level. An Orbix object reference contains host name, interface name and server name, and is at a higher level than an ANSAware interface reference which contains protocol addressing information. This higher level is possible because of the existence of the Orbix daemon.

   An additional important difference is the uniqueness of ANSAware interface references provided by the ‘nonce’; Orbix object references have no equivalent mechanism, and tend to be predictable.

   Object/Interface references must be treated as opaque in most of the code, and the internals should only be manipulated (if at all) in the IDL-dependent part.

   Any one trader should only expect to handle the kind of interface/object reference that it is built for.

An alternative approach is to use ANSAware interface references throughout, with other types of references buried within them.

2. ANSAware dependencies in the trader interfaces.

   For example, the ANSAware trader interface involves the Capsule and Notify interfaces; mapping to CORBA is not entirely obvious.

   Define interfaces in the most natural way for the IDL concerned, and put code in the IDL-dependent part when necessary.

3. Differences in threading models (synchronous/asynchronous, pre-emptive or not).

   Needs further study, but the problem has been successfully tackled when making ANSAware work with DCE threads - see [TR.037.00 93].

4. The separation of the trader and the type manager may lead to a problem if they become physically separated: the trader starts up, and has no type manager to use when registering its own internal interfaces; the type manager cannot register itself until this has been done. This is a bootstrapping problem.

   The solution may be to incorporate a small internal type manager into the trader, and use that until the external one registers itself. This is the approach adopted by BNR in ITOM; their 'basic' internal type manager is based on string equality (it makes no use of type graphs).

5. Implementation in C++ may be a problem if we want to bring this work back into ANSAware.

   We don't necessarily have to bring the work back into ANSAware; alternatively we could continue to issue the existing trader for those people who don't have C++.

6. Orbix 1.1 has some problems at present with passing around an object reference to an unspecified type of object:

   — IDL doesn't know about the type 'Object' at present

   — passing an 'any' which is an object reference doesn't work

   — if an object reference is passed to a process which is not actually going to use it, and the client code for it is not built in to that process, Orbix treats this as an error (even if the client code for a derived class is included, and even if we are only intending to use the facilities of the derived class).

   We (and Iona) hope that Orbix 1.2 will be better.

6.2 Monitors and Resource Management

Note: To be supplied.

6.3 Type Checking

Note: To be supplied (ajw ?).

6.4 Interception

Note: To be supplied.

References

[APM.1004.0 93]

Warne, J.P., Rees, R.T.O., “ANSA Atomic Activity Model and Infrastructure”, Architecture Projects Management, Cambridge, 1993.

[APM.1005.1 93]

Deschrevel, J-P., “The ANSA Model for Trading and Federation”, Architecture Projects Management, Cambridge, 1993.

[APM.1010.0 93]

Rees, R.T.O., “Using Path Expressions as Concurrency Guards”, Architecture Projects Management, Cambridge, 1993.

[APM.1014.0 93]

Howarth, N.J., Watson, A.J., Rees, R.T.O., Otway, D.J., “DPL Programmers' Manual”, Architecture Projects Management, Cambridge, 1993.

[APM.1015.0 93]

Otway, D.J., “DPL Reference Manual”, Architecture Projects Management, Cambridge, 1993.

[APM.1069.0 93]

van der Linden, R.J., “Service concepts in evolving systems”, Architecture Projects Management, Cambridge, 1993.

[ARM]

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

[Craft 85]

Craft, D.H., “Resource Management in a Distributed Computing System”, PhD Thesis, Technical Report No 73, University of Cambridge Computer Laboratory, 1985.

[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.

[Herbert]

AJH/RMN paper - details (and paper) to be supplied.

[ICL 93]

“ICL Proposal in Response to OMG Object Services RFP 1”, ICL and BNR Europe, 1993.

[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.

[ISO 93]

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

[OMG 92]

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

[OMG 93a]

"Joint Object Services Submission: Submission Overview", Document Number 93.7.1, Object Management Group, 1993.

[OMG 93b]

"Joint Object Services Submission: LifeCycle Services Specification", Document Number 93.7.4, Object Management Group, 1993.

[RC.258.03 92]

Watson, A.J., "Types and projections", Architecture Projects Management, Cambridge, 1992.

[RC.339.02 92]

Watson, A.J., "Revising the DPL type system", Architecture Projects Management, Cambridge, 1992.

[TR.037.00 93]

Nicolaou, C.A., "ANSAware use of DCE/Posix Threads and RPC", Architecture Projects Management, Cambridge, 1993.