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 ANSA Model for Trading and Federation

Jean-Pierre Deschrevel

Abstract

This report describes the ANSA trading service and the ANSA view on federation.

The trading description includes the definition of an information classification scheme, the computational model for trading and the evaluation of computational and engineering related considerations.

The federation concept is then introduced and motivated. The ANSA principles behind federation are provided. Trading across federation is then analysed in terms of discontinuities and boundaries.

APM.1005.01 **Approved** 15 July 1993

Architecture Report

Distribution:
Supersedes:
Superseded by:

The ANSA Model for Trading and Federation

Architecture Report

The ANSA Model for Trading and Federation

Jean-Pierre Deschrevel

APM.1005.01

15 July 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 Overview

A major advantage that distributed object environments bring to the programmer is the ability to build applications that reuse or can interwork with existing components. Mechanisms to assist in locating and using existing resources are therefore of critical importance when constructing object-based systems, the more so when those systems are distributed on a large scale and may span organisational or even national boundaries.

During the life of an application, whether distributed or non-distributed, the relationship between each part of it and with its support infrastructure may change frequently. Associations between the components may be determined:

- when the component is being constructed (i.e. at compile-time)
- by a configuration process after compile-time but before run-time (i.e. linking time)
- at run time.

In each case the general problem is how to determine what objects, libraries or services are available and then identify one or more that is suitable.

Many distributed system architectures provide for name services that allow clients to locate services at run-time by mapping a attributive name for a service onto a service address which can be used to invoke that service. This Architectural Report describes the ANSA trading service, which allows clients to locate a service which performs a specified function rather than one with a particular name. For a useful analogy, consider searching for a service using the telephone directory; the Yellow Pages allow you to look up for a function (e.g. "plumber") and give support for making a choice, whereas the ordinary directory is only useful if you know the name of the service provider ("John Smith").

Trading is an important component of the ANSA distributed system architecture, and also forms part of the forthcoming Basic Reference Model for Open Distributed Processing being drafted by ISO/IEC JTC1/ SC21/WG7.

1.1 The concept of trading

ANSA provides the application programmer with a distributed object-based programming model in which objects provide services to each other, with one object potentially providing and using several services simultaneously. Before a consumer can use a service it needs to have access to a provider that offers the required service. In general the knowledge of what services are available in a particular community of service providers is distributed over that community, and no individual service consumer has sufficient knowledge to identify the most suitable service provider from amongst all those available. This problem is addressed by introducing a specialised service whose purpose is to accept and store *service offers* from potential providers and hand out this

information on request to potential users. This agent or broker for service providers is known as a *trading service*. Trading is defined as:

> *the activity of choosing services, such that they match some service requirement. The choice is based on the comparison of the specification of a service required (provided by a prospective consumer) and the service specifications supplied by service providers or their agents.*

A trading service is not the only service that can store knowledge about and references to other services; on the contrary, in ANSA any object and service can do this. The trading service is however specialised in imparting and distributing knowledge about other services. We need not limit ourselves to a single trading service; many such services may coexist and can be made to cooperate. ANSA's object model and support infrastructure are designed to allow trading services to be written as ordinary application-level objects rather than forming an integral part of the infrastructure. The trading service can then be made available to all objects.

1.2 A practical example

The operation of a trading service will be introduced with the aid of a brief example.

Imagine that a network-based data-handling application has been written which uses a sorting service. In an object-based design, such a service might be implemented as a collection of operations provided at the interface of an object. Different sorting services can be present in the network, providing the application with a variety of services from which to choose. So some form of run-time selection from the set of available sorting services is needed.

A trading service meets this need by allowing generally-available services to register themselves when they become available, and prospective users to interrogate the trading service to locate a service at the time when one is needed.

In a typical implementation of trading the service provider (in this case the sorting service) registers the availability of the service by invoking an operation on the trader to export its offer, passing as parameters information about the offered service. A service provider will typically make this registration as part of an initialisation process when it is first activated. In order to do this, it must possess a reference to the trading service; this could be passed as a parameter to the service provider by its creator, or built into the object as a service reference that is “hard-wired” in.

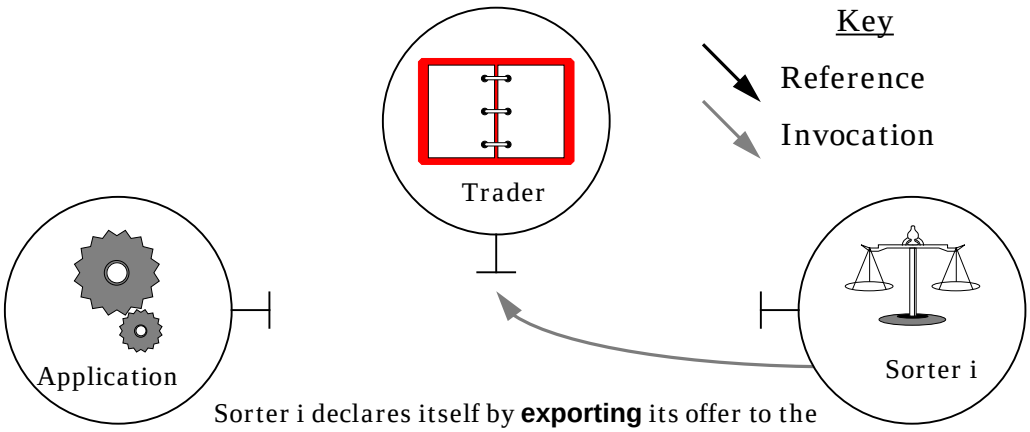
The registration operation carries the following information:

- An interface reference[1] that can be used by a client to invoke operations on the advertised service, in this case a service that sorts data.
- A description of the interface type of the offered service (that is, the names of the operations to which it will respond, along with their parameter and result types).

1. Interface references are context-relative, location-transparent names for interfaces. More information on interface references can be found in [AR.007].

- Information on the distinguishing properties of the offered service; in the case of a sorting service this might include a measure of the speed of the algorithm, cost in terms of resources, capacity, and so on.
- In which of the trader's hierarchy of contexts the offer should be registered.

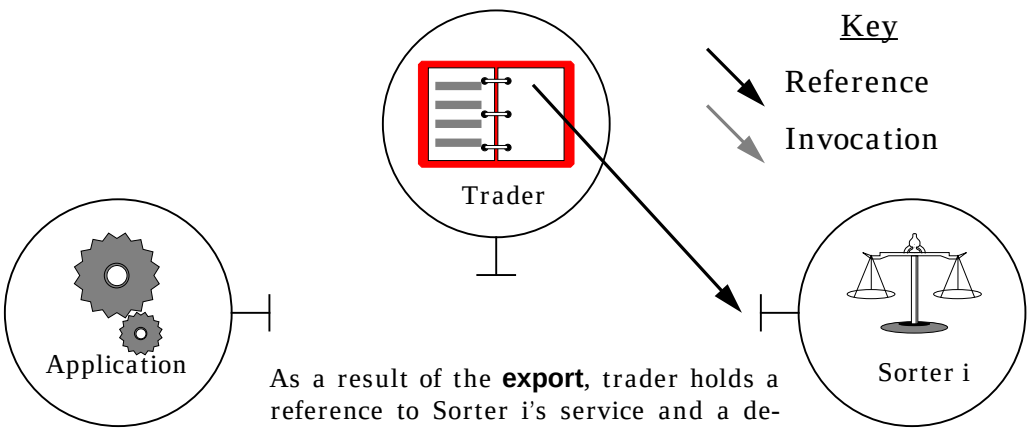
Figure 1.1: Registration of a service to a trader (1).



Sorter i declares itself by **exporting** its offer to the trader, passing a reference to the service that it provides as one parameter.

On receiving the export request the trader simply stores the service description and reference as an offer in the specified context within its internal database. Contexts are used to organise and subdivide the potentially very large collection of offers held by a trading service. For the purpose of administration and search the trader's database can be structured as a directed graph of contexts, each of which holds other contexts and/or offers. There is no one prescribed way of relating offers to contexts; in this example the sorting service's offer might be placed in a context devoted to sorting services. Contexts and the related concept of administrations are fully described in §2.2 on page 8.

Figure 1.2: Registration of a service to the trader (2).



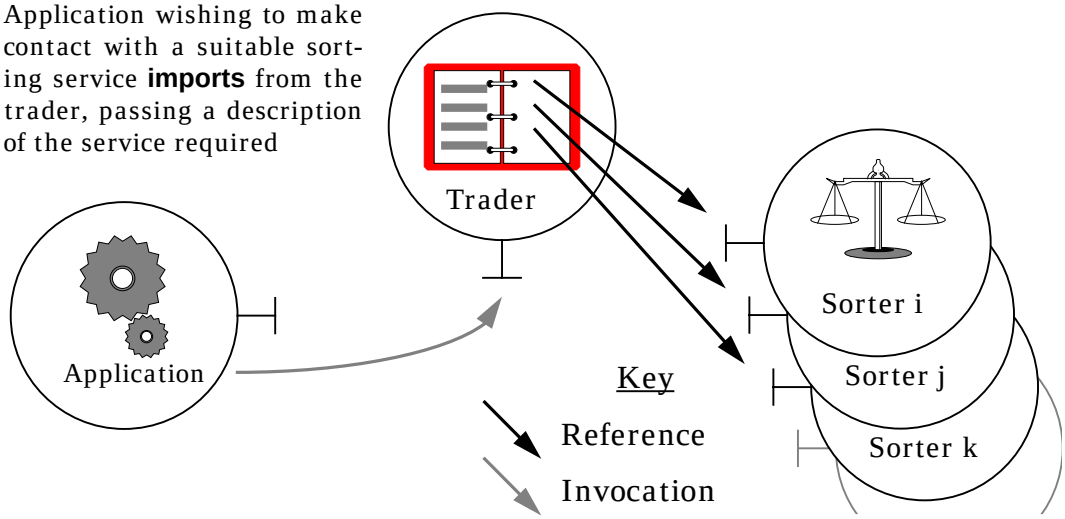
As a result of the **export**, trader holds a reference to Sorter i's service and a description of the service offered in its database

Whenever a potential client, such as a data-handling application, wishes to obtain a reference to a service that does a particular job, it uses its reference to the trading service (which it too was given when it was created) to invoke the trader interface and import an offer, passing as parameters a description of the service required. This description has a similar general structure to that used during the registration operation. In particular, it includes:

- a type description that tells the trader what operations the application is going to expect to be able to invoke on a service obtained from the trader;
- constraints on the acceptable values of particular properties - for example a minimum acceptable sorting speed for a particular amount of data or the lowest resource consumption;
- which contexts to search.

Given this import request, the trader checks all the offers in the requested set of contexts looking for acceptable service offers. Limiting the search process to only part of the context graph can speed up the search, and also allows the service consumer to exclude offers in inappropriate contexts from being considered. If necessary, the entire offer database can be searched.

Figure 1.3: Importing a service from the trader (1).



To be acceptable, an offer must have a type that conforms to that requested, and properties that match the constraints specified.

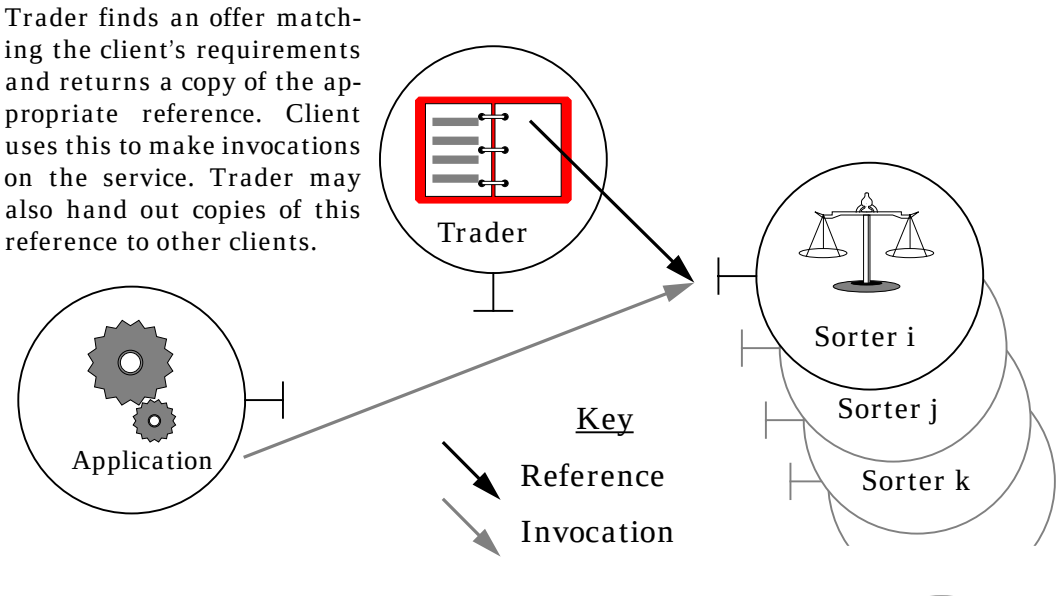
To conform, the offer type must have at least the set of operations wanted by the importer, each with the correct number of parameters and results of appropriate types (see §2.1 on page 7 for the definition of “appropriate”). It is quite in order for a service offer to possess a superset of the requested operations - the prospective client will simply never call these “surplus” operations. However, an offer with a type that does not include all the requested operations cannot safely be selected, since this prospective client is likely to attempt to invoke operations that the offered service does not possess.

If no acceptable offers are located then the client is informed that the import failed. However, if one or more acceptable offers are found then one of them is selected at random, and a copy of its interface reference returned to the importer. The randomisation is included to help load balancing by sharing

clients among all the eligible service providers rather than always selecting the same one. This is particularly beneficial if one of the offers pertains to a server that has crashed, leaving a stale offer behind.

The client is also informed of the exact properties of the selected offer so that it may tailor its use of the service to match its properties.

Figure 1.4: Importing a service from the trader (2).



Once the trader has supplied an interface reference to the client it takes no further part in the interaction between the client and the service provider. The client can use it as many or as few times as it pleases, immediately or later, all without further involvement of the trading service.

The offer remains in the trader's database until explicitly withdrawn, and in this way many clients may be put in touch with the same service over a long period of time. When an offer is withdrawn it is not the trader's responsibility to inform clients that the service is no longer available; all clients have to be prepared for any service to disappear at any time, and take appropriate remedial action such as going back to the trader and importing another suitable service.

1.3 Roadmap

This report is divided into 6 parts, which are:

- This introduction and overview, which presents trading by means of a simple example, and introduces the concepts that will be described in this report.
- Three classification schemes are introduced to partition the offers in a trading database. The classification is made according to contexts, types and properties.
- A computational model of trading is provided including the definition of three context interface types. Context behaviour is described as well as the related issues.

- In chapter 4, engineering considerations are analysed.
- The concept of federation is introduced in chapter 5. Arguments in its favour are given and its principles are presented.
- Then, federation is described in terms of boundaries. A description of how trading can be provided across boundaries is then provided, and the concept of interceptors is introduced.

2 Classification Scheme

The ANSA trading service uses three classifications schemes to categorize service offers. The offers are simultaneously classified according to the type of their interface, the administration to which the server belongs and to some defined properties.

Other classifications schemes than these three can be introduced. For instance security considerations are likely to influence, and be integrated into, the context structure, one may have a security infrastructure requiring that a separate classification scheme be designed.

2.1 Service types

A service in ANSA is provided at an *interface*. The interface provides a number of distinctly-named *operations*, each with one or more possible *terminations*. Terminations are also distinguished by name, although for convenience there is usually one termination whose name is the empty string (the *anonymous termination*) which is used to carry the "normal" or "expected" result from an invocation. An operation may carry zero or more arguments and each termination can carry zero or more results, thus providing a completely symmetrical request/result structure. A set of operations with their respective terminations (including parameters and results) determine the interactions that a service can engage in at a particular interface. The *type* of an interface describes the names and parameters of each of the operations and terminations, and therefore the structure of the permissible interactions with the service.

Object models which support only a single result from an operation can be thought of as a special case of the ANSA model in which each operation has only one termination, passing a single result.

For the service reference provided by the trader to be acceptable to the client, it must be able to respond to all the operations that the client may attempt to invoke, and must not return any terminations that the client is not prepared to handle. This relationship, informally known as the *no surprises rule*, is codified as *conformanc*e. Conformance is defined in terms of four mutually recursive rules:

- Interface conformance

  The type of interface I1 conforms to that of I2 if and only if for each operation O2 in I2 there exists an operation O1 in I1 with the same name whose operation type conforms to O2

- Operation conformance

  The type of operation O1 conforms to that of O2 if and only if the parameter list type of the arguments of O2 conforms to the parameter list type of the arguments of O1, and the response type of O1 conforms to the response type of O2

- Response conformance

  The type of response R1 conforms to that of R2 if and only if for each termination T1 in R1 there exists a termination T2 of the same name in R2, and the parameter list type of T1 conforms to that of T2

- Parameter list conformance

  The type of parameter list L1 conforms to that of L2 if and only if L1 and L2 have the same length and the type of each interface in L1 conforms to that of the corresponding interface in L2

The conformance relationship is transitive, that is:

$$A \textit{ conforms-to } B \wedge B \textit{ conforms-to } C \Rightarrow A \textit{ conforms-to } C$$

This interface type conformance statement is defined in detail in the interaction model of the ANSA computational model. See [AR.001].

2.2 Administrations and contexts

There can be many service providers and consumers in a distributed system. Such a large community can generally be broken down into several administrations. An *administration* is defined as a set of community members that are subject to a common source of authority and that share a collection of goals. To perform their duties and achieve their goals, community members may use services that are offered by community members in the same and other administrations.

The exchange of information between administrations requires two conditions to be satisfied:

- it must be possible to use services provided in another administration (i.e. interactions must be able to go from one administration to another);
- information must be available that allows objects in one administration to determine whether objects in the other offer useful services. This requires mechanisms for exchanging type and property information between administrations.

Administrations can be linked in different ways; the following sections describe two models of relationship.

2.2.1 Hierarchical organisations

An administration may be divided into subadministrations. The members of a subadministration are governed by the authority delegated to the subadministration from the administration to which it belongs. A subadministration is no different from an administration and may itself be divided into smaller subadministrations, thereby forming a tree structure. In this tree, each vertex denotes an administration and each directed edge denotes the *is-a-subadministration-of* relationship. The *is-a-subadministration-of* relationship is transitive, i.e.

$$A \textit{ is-a-subadministration-of } B \wedge B \textit{ is-a-subadministration-of } C \Rightarrow A \textit{ is-a-subadministration-of } C$$

The administration graph is directed and acyclic (DAG).

This structure is well suited to hierarchical organisations, but is not appropriate when it is necessary to achieve interworking between autonomous

administrations (such as two companies, or even two sub-administration of a same structure). The concept of *federation* is employed to encode the relationship between such peer administrations.

2.2.2 Federal model

In large-scale distributed computing systems, the existence of centralised ownership and universal and technical control cannot be assumed. The only way to achieve cooperation between individual systems avoiding any hierarchical structure is to adopt a "federal" style. Federation enables systems to negotiate sharing of service without losing control over their own policies and services. See §5 *Federation*.

A federal approach to a link between administrations is characterised by four principles:

1. an administration must not be forced to perform an activity for another administration;
2. an administration must have freedom of association with respect to the federation;
3. each administration determines which resources it wishes to share with other administrations;
4. each administration determines how it will view and combine the resources that are made available by other administrations.

2.2.3 The context structure

A trading context is associated with each administration. The trading context is defined as:

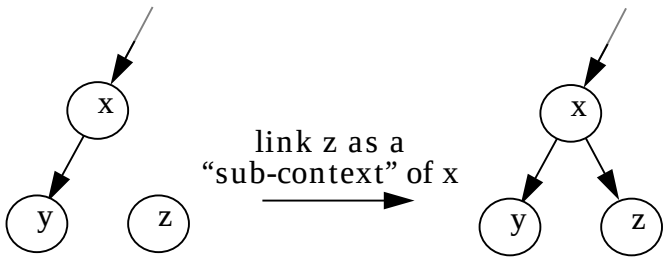
> *A set of service offers that pertain to service providers that are under the authority of a particular administration.*

Each trading context is labelled with the name of the administration with which it is associated.

Links are created between contexts that reflect the relations defined between administrations. Following the previous descriptions of administration relationships, two different kinds of link can be created between contexts: the hierarchical and the federal link.

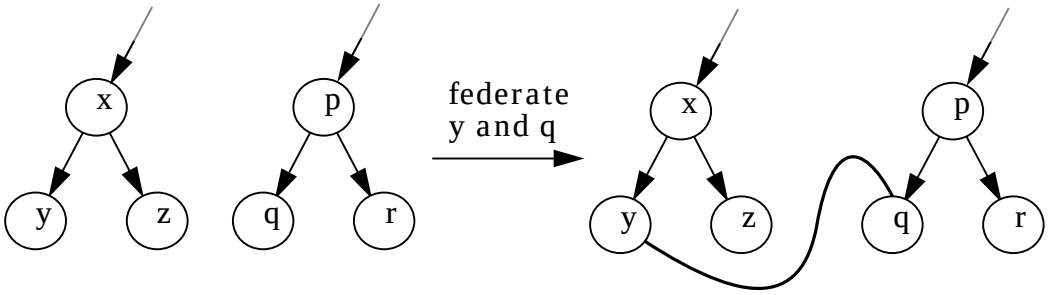
The hierarchical link represents an is-a-subadministration-of relationship between the administrations. The context related to the subadministration can be searched for information by the other context. It is thus possible to construct a tree that reflects this structure, and forms a convenient basis for organizing the trading offers in a form that scales well and is easy to navigate.

Figure 2.1: Hierarchical link between two contexts.



Contexts can also be linked to represent a federal relationship between the administrations to which they are related.

Figure 2.2: Federation of two contexts.



Each context identifies the information it will make available to, and permit to be searched by, the linked context.

2.3 Service properties

A *property* is a characteristic of a service that is not expressed by its type. Examples of properties are the quality of a service (e.g. response time or throughput) or the cost associated with the use of a service. Properties are associated with a service instance to help a client choose among services which all support the same interface type and which are members of the specified administration(s). At the time an instance is registered with the trading system, the component advertising the instance associates property (Name, Value) pairs with the instance. When a prospective service consumer queries the trading system about service instances which match a particular type, the query is refined by placing constraints on the property values that such an instance must possess.

The trading service cannot guarantee the validity nor the correctness of a particular property value. There is no way for a trading service to check that the value associated with a property is correct at the time a service offer is passed to a service consumer, or at the time the service consumer starts using the service.

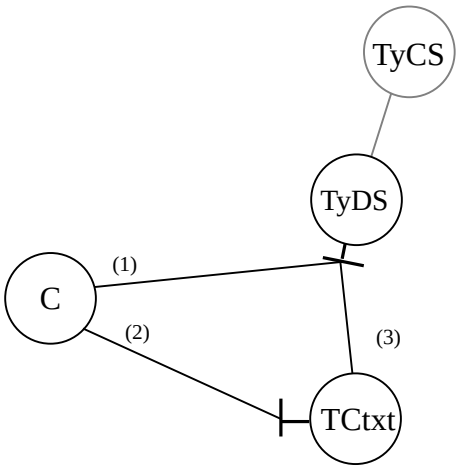
Note that properties of services can be static or dynamic. An advanced functionality for trading could be to be able to deal with dynamic properties. For example, a printing service may have a property that represents the work load that is currently involved on a printer. In this particular case, a dynamic work load property might be an important criterion in choosing a service.

3 Computational Model for Trading

This chapter describes the ANSA trading service from a computational viewpoint. Trading contexts are defined and used in the following to refine the model and introduce the issues related to trading.

Every trading interaction begins with an operation invocation at a particular trading context interface: any object wanting to use the trading service needs to know, at least, the reference of a trading context by which it can start interacting and gain information about other contexts and offers. For particular queries, a type description service is involved.

Figure 3.1: A Computational model for trading.



(1): instantiation of TyDS with the service interface type required or provided.
(2): trading service invocation, providing the interface reference of TyDS.
(3): conformance request for a potential offer described by a service TyDS'.

TCtxt	Trading Context.
TyCS	Type Conformance Service.
TyDS	Type Description Service.
C	Client.

The model, for a simple interaction between a client and the trading service, involves three objects as shown in figure 3.1. In this model, the client can be either a service provider or a service consumer. The three different services identified as part of the overall trading service are:

- the *Type Description Service*
  TyDS stores interface type definitions. After being initialized by the *Client*, TyDS provides an interface with two operations: the first one permits a type description to be returned, and the second allows the conformance between the type held and another one to be tested. When a trading service client wants to perform an export or import on a trading context interface it needs to provide a description of an interface type. This is done by providing the reference to a TyDS interface.
- the *Type Conformance Service*
  A Type Conformance Service object has been shown in figure 3.1 to introduce the idea that the conformance test can be provided by a separate service, independent of TyDS.
- the *Trading Context*
  TCtxt embodies a trading context, and maintains knowledge about all the

offers that pertain to this context. TCtxt maintains also information about other trading contexts to which it is bounded. When a client makes a query about other contexts or which involves other contexts, TCtxt will invoke other TCtxt objects, becoming itself a client to these services. TCtxt tests conformance between interface types by invoking TyDS objects.

The import scenario is simple. The client instantiates a TyDS service with an interface type description of the service it will be looking for. It then invokes the trading context providing the reference of TyDS, plus information about the service (a property constraint expression). The trading context then searches its database of offers asking TyDS for conformance tests. In order to do so it has to provide the TyDS service with the reference of another TyDS service.

Import query may require TCtxt to invoke other trading contexts. Export requests only involves one trading context.

3.1 Trading Contexts

This section presents interface definitions for a trading service. Two types are defined which provide a set of building blocks permitting easy implementation of a basic trading service without having to build in policy decisions too early. These two interface types are:

1. the *BasicContext* type which gives access to a simple trading service. As we will see further, two distinct trading context behaviours can be associated with this interface definition
2. the *SearchingContext* type which permits implementation of searching policies.

Each interface provides access to a single-context trading service; full multiple-context trading is provided by linking together multiple instances of these two basic types. Engineering considerations will motivate the decision to co-locate or distribute these single-context trading services.

3.1.1 The BasicContext interface type

The *BasicContext* interface is the simplest form of trading service. The type definition for `BasicContext` is the following[1]:

1. The DPL syntax has been used in the following to describe the interface types. Detailed information about this syntax can be found in [TR.031]. As a remainder of this syntax rules, names and parameters of operations can be described as:

<operation-name>
“(“ <argument-list> “)”
“-> (“ <result-list> “)”
“->” <termination-name> “(“ <result-list> “)”

```
BasicContext = type (
     Register
          (T:AbstractType Service:T Values:Properties)
          ->   (Offer)
          ->   Failed(String)
     RegisterMonitor
          (T:AbstractType M:Monitor Values:Properties)
          ->   (Offer)
          ->   Failed(String)
     LookUpOne
          (T:AbstractType Requirement:Constraint)
          ->   (T Properties)
          ->   Failed(String)
     LookUpAll
          (T:AbstractType Requirement:Constraint)
          ->   (type(next() ->(type(realtype()->(AbstractType)
                                    service() ->(T)
                                    values()  ->(Properties) )
                               )
                          -> end()
                    )   % This declaration defined a sequence
               )        % of record of service offers.
          ->   Failed(String)
     Link
          (OtherContext:BasicContext Name:String)
          ->   ()
          ->   Failed(String)
     UnLink
          (Name:String)
          ->   ()
          ->   Failed(String)
     Contexts
          ()
          ->   (Set.of(String))
          ->   Failed(String)
     Select
          (Path:Sequence.of(String))
          ->   (BasicContext)
          ->   Failed(String)
     ) % end of BasicContext.
```

In the `Register` operation, `T` is a run-time representation of a type, `Service` is a reference to the service being registered (which is of type `T`) and `Values`/`Properties` are specifications of service properties. The `RegisterMonitor` operation differs form `Register` in that the reference to a monitor is provided instead of the reference to the service. This is done to permit external policies to be introduced in trading; as it is described §3.2. The result of these operations is a reference to an interface presented by the newly-created `Offer`, which can be used to withdraw the offer:

```
Offer = type (
     withdraw() -> ()
                -> Failed(String)
     )    % end of offer.
```

The `LookUpOne` operation permits a service offer to be obtained, given the required type and constraints to be satisfied. Alternatively `LookUpAll` will

return all the service offers registered in the context that satisfy the type and constraints specified.

The `Properties` and `Constraint` types are not formally defined. The syntax and semantics of the properties and of the constraint expression on them form an open issue which needs further study. A simple solution can be to define it as a `String`. Different parties will still have to agree on how to express properties within the string.

The `Link` operation permits a named link to be made to another context, deleting any previous link of the same name. Either a `BasicContext` or a `SearchingContext` may be linked; they both have types that conform to `BasicContext`. Links can be withdrawn by invoking `UnLink` with the name of a previously defined link.

The `Contexts` operation returns a list of all the contexts for which links exist. The `Select` operation gives the interface reference of a context given a path composed of link names, permitting navigation through the trading graph.

The `Select` operation is implemented by taking the first name component in `Path`, fetching the named linked-context, then recursively calling `Select` on this context with the rest of the path. As previously, the interface reference returned by this operation has the type `BasicContext` which permits either a `BasicContext` or a `SearchingContext` to be referred to, since the latter conforms to `BasicContext`. If `Path` is an empty set, the reference returned is the one of the invoked context.

Declarations of `Monitor` and `AbstractType` can be found respectively in §3.2 and §3.3.

The `BasicContext` gives the ability to register and look-up offers. It provides also the tools to link the contexts one to another to construct a context graph, and to navigate this graph. Two trading behaviours can be associated with this type definition. A minimal trading context service and a full trading context service are respectively specified in the following sections.

3.1.1.1 A leaf-context behaviour

A *leaf-context* service is one which fails any attempt to be linked to other contexts. Such a service is able to trade service offers which belongs to the administration to which it refers, but not to maintain links with other contexts. It is thus a leaf in a context graph.

This context will fail any invocation of the link, and therefore of the unlink operations. The `Contexts` operation always returns an empty set, and `Select` either fails or returns the reference of the context itself.

3.1.1.2 A branch-context behaviour

A complete service is provided by a branch-context. This context associates to the service offer handling, the ability to be linked to other contexts and thus to take the place of a branch in a context graph. The operations have the original behaviour as described in §3.1.1.

Thus a BasicContext is a trading context which can be linked to other contexts in order to build a trading graph. Leaf-context and branch-context provide the modularity to construct structured graphs.

Navigation between contexts must be done by the client of a context, using `Select` to go from one context to another. All offer searching is limited to within the context itself, with other searches being programmed by the client.

3.1.2 The SearchingContext interface type

The SearchingContext is a context with a built-in policy for searching a set of BasicContexts (in fact *SearchingContext* really denotes a family of similar types differing only in what searching constraint operations they support). The SearchingContext interface type is the following:

```
SearchingContext = type (
     SetSearchingConstraints
               %    a set of operations specific to the searching
               %    policy provided by the context.
               %    the signature of these operations may vary
               %    according to the requirements
     Register
          (T:AbstractType Service:T Values:Properties )
          ->   (Offer)
          ->   Failed(String)
     RegisterMonitor
          (T:AbstractType M:Monitor Values:Propertie )
          ->   (Offer)
          ->   Failed(String)
     LookUpOne
          (T:AbstractType Requirement:Constraint )
          ->   (T Properties)
          ->   Failed(String)
     LookUpAll
          (T:AbstractType Requirement:Constraint)
          ->   (type(next() ->(type(realtype()->(AbstractType)
                                    service() ->(T)
                                    values()  ->(Properties) )
                               )
                          -> end()
                    )   % This declaration defined a sequence
               )        % of record of service offers.
          ->   Failed(String)
     Link
          (OtherContext:BasicContext Name:String)
          ->   ()
          ->   Failed(String)
     UnLink
          (Name:String)
          ->   ()
          ->   Failed(String)
     Contexts
          ()
          -> (Set.of(String))
          ->   Failed(String)
     SelectWithPolicy
          (Path:Sequence.of(String) PolicyName:String)
          ->   (SearchingContext)
          ->   Failed(String)
     Select
          (Path:Sequence.of(String))
          ->   (BasicContext)
          ->   Failed(String)
     ) % end of SearchingContext.
```

The `SetSearchingConstraints` operation defines the searching policies that can be used in the context. This set of constraints is dependent on the environment and the application that are served by the trading context (see §3.4). An example of a basic set of constraints might be: *search-local-only*, *search-distant-only* and *search-both*.

The `Register`, `LookUp` and `Link` operations are identical to those in `BasicContext`. As with `BasicContext`, the `Contexts` operation lists all the contexts for which links exist. `SelectWithPolicy` returns an interface reference to the named context as a `SearchingContext` with the desired trading policy. This operation permits searching linked `SearchingContext` in the particular way which is specified by the `PolicyName` parameter. It provides the functionality to search wide and complex trading networks.

3.1.3 Trading context creation

The creation of trading contexts is done by requesting a trading context factory. ANSA architectural principles on how computational objects are created are given in [AR.001]. The invocation of a trading factory service can cause the creation of a new computational object. The initialisation process for a context creation may include the opening of a management interface as well as the context interface. it may also define initial bindings and set searching policies. The exact specification of the trading factory is for further study.

A trading context management interface can provide the operations to create contexts.

The invocation of a trading context factory may not induce the creation of a computational object: a new context interface may be open on an already existing trading object. Trading contexts needn't lie in separate computational objects. The way trading context services is distributed into trading objects is an implementation issue described §4.2 on page 21.

3.2 Dynamic services

Two implementation concerns are of particular importance in the area of trading: resource usage and load sharing. Both issues are important in the provision of efficient and scalable systems. Many other implementation aspects have been addressed elsewhere in ANSA; they are not peculiar to trading, as the trading service is just another ANSA application.

It is expected that a service that is exported is available when imported and used by a client. There may however be some delay between the service being exported and it being imported. There may be another delay after being imported and before actually being used. During all this time the service would be in existence, and thus consuming resources, but idle.

To make efficient use of the resources in a system, it would be an advantage to be able to make a service offer, but delay creating the service until a later stage, such as the time that some client imports the service from the trader. The `RegisterMonitor` operation in the two context interfaces described provides this facility. The type and properties of the service which is to be exported are provided as arguments like for a `Register` operation, but instead of passing the interface reference of the actual service the reference of a monitor service is provided. This monitor service will act as an agent for the service exporter. When the service registered is going to be imported (i.e.

passed as a result of a LookUp operation) the trader object will invoke the *Monitor* agent to obtain the actual reference of the service. When the monitor is called, the desired service type and properties are included to make sure that the monitor knows exactly what service instance to make available.

Doing this before returning from the import operation means that the client will not experience any delays in its first interaction with the imported service. The interface type to the monitor looks very much like a trading interface:

```
Monitor = type (
     MonitorGet(T:AbstractType Requirement:Constraint)
               ->   (T Properties)
               ->   Failed(String)
     ) % end of monitor
```

As seen in the previous type definition, the monitor takes a constraints expression stating the required service properties, and returns an interface reference to the service instance (T argument), together with the actual properties associated with the service instance.

This mechanism is provided to off-load policy enforcement from the trading service. Different uses have been identified so far:

- to permit notification of imports for monitoring purpose
- to notify the server that it might be used (e.g. for activation purpose)
- to delay the creation of the service until import time
- for authentication
- more generally for resource and load management.

The precise specification of the *Monitor* service is not provided and represents an area for further study.

3.3 Trading types and type libraries

In the trading context interfaces, the type of the requested or provided services is represented as abstract data type (AbstractType). This abstract data type is a computational interface providing two operations:

```
AbstractType = type (
     GetDescription()
               ->   (Syntax)
               ->   Failed(String)
     ConformsTo(T:AbstractType)
               ->   true()
               ->   false()
               ->   Failed(String)
     ) % end of AbstractType
```

This type describes an interface provided by a *Type Description Service* (this service was firstly introduced on page 11) whose purpose is to represent types and test conformance at run-time. The reference to this service is passed to the trading service each time a representation of an abstract type is required by clients or servers (i.e. for Register and LookUp operations).

Each type description service is bound to a type conformance service, to be able to perform the `ConformsTo` operation (see figure 3.1 on page 11). Therefore the type description services might be offered at interfaces of a single type management object internalizing the conformance function. The management of type might also be co-located with the trading service to speed up type matching in searches. The way the type description service and the trading service are co-located, distributed or replicated is a matter for the implementor, and depends on the required functionality, reliability, and performance of the service[1].

`GetDescription` returns a representation of the type as an abstract syntax tree for its signature. The exact specification of the type `syntax`, although it has not been completely defined yet, has to be agreed by the systems that potentially want to interact.

The `ConformsTo` operation tests if the type it describes conforms to the abstract type passed as an argument (this test can be performed by using a `GetDescription` operation and comparing the syntax trees according to the rules specified in the ANSA Computational Model [AR.001]).

It is often convenient to give names to types and to provide a *type library*. This can easily be accomplished by exporting the type description interface reference to the trading service, into a *type library* context.

A library context can be constructed to maintain a type graph, in which each exported type is inserted at a position in the type hierarchy compatible with its signatures.

Thus the library is a specialized trading context, and can be federated with other contexts. The library can also provide the service of creating type description interface references for type specifications provided as syntax trees (perhaps in textual form). In this case the library pre-computes the subtype, supertype relationships between all the types it contains so that the `ConformsTo` operation becomes one of table lookup[2].

A trading service that supports library contexts can extend the basic `Register`, `LookUpOne` and `LookUpAll` operations to provide variants that use the textual names of types, rather than interface references to type descriptions services. However, it is then necessary to ensure that the interpretation of names made by importer and exporters is not out of step with those made in the type library. Defining variants of `Register`, `LookUpOne` and `LookUpAll` that use type names rather than type description (as for example in ANSAware) is therefore not generally recommended.

3.4 Searching trading networks

A trading network is created or extended when one trading service publishes its service in a context managed by another trading service. The architecture does not impose any constraints on the pattern of interconnection. The trading network can thus span multiple trader objects, be distributed, can be very large, and may contain loops. Searching such a network for suitable service offers can be expensive in several respects. In some cases the searching

1. See chapter 4 *Engineering considerations*

2. This is simply a design choice trading the space requirements of the resulting table against the cost of executing the conformance algorithm.

algorithms can be optimized if application specific constraints are imposed on the structure of the trading network. Here we do not assume any such optimization.

Separate trading networks can be federated using a set of import/export contexts. Service offers that are advertised (exported) in a context in one trading network are made visible in other trading networks by explicit creation of an export context (for a particular federation) in which such an offer is then placed. Importing service offers from other trading networks is achieved by the creation of an import context through which one or more export contexts of other federated trading networks can be accessed. The indirection through the import and export contexts allows administrators of federated systems to impose the controls that stem from the concerns of ownership and autonomy, exercised by each organisation. Users of the now federated system can remain unaware of the indirection. In effect, the import and export embody the trading policy for the federation. Chapter 6 *Federation and Trading* explores the policy issues for each ends of the federation boundary.

The parameters that influence the way in which a trading network is searched are defined in the `SetSearchingConstraints` from the `SearchingContext` interface.

In a large trading network searching may take a long time. This can either be because of the policy used or because of communications delay in the interactions between remote trader objects. Delays can be curtailed by having the calling context impose a response deadline, after which a response from the called context will be ignored. Alternatively deep searching policies may be detected before a search starts, and an estimate of the cost of the search could be made.

3.5 Security

The trading service is not the only agent that maintains knowledge about services in a system. An important principle in the ANSA architecture is that the capability of maintaining and passing information about services is granted to all the entities that are present in the system. An interface reference may be passed by any entity to any other entities; this capability is not peculiar to trading and the trading service must be considered as just another ANSA application. Similarly, trading does not play any special rôle in a system's security architecture, and the relationship between trading and security is a policy concern for the designers of individual systems.

More information about the ANSA security architecture can be found in [RC.322].

The general-purpose trading service is just a means by which consumers can obtain interface references for services. In this case the service provider itself must control access to a service advertised in the trading service, presumably by authenticating the prospective client when it attempts to use the service and applying its own security policy to decide whether to honour the request.

On the other hand, some systems may require a trading service with a security policy of its own, so that it can apply access control to clients wishing to use its services. For instance, it may be desired to limit the visibility of the information maintained in a particular context or restrict the publication of

services in particular contexts. All such trading security policy decisions are the prerogative of the designer rather than the architect.

It is believed that contexts and flexible ways of linking them give designers the tools with which service offers can be separated and access to them controlled, so that security concerns can be reflected in trading system designs.

4 Engineering considerations

4.1 Mapping onto the X.500 directory service

The X.500 directory is a joint recommendation defined by CCITT and ISO for an open directory service. The definition of this directory is intended to be very general, to maintain information about a great diversity of entities, such as people, organisations, resources, objects, and so on, and to be flexible enough to be used for many purposes. The trading service is in comparison a very specific “yellow pages” style directory service.

A few attempts have already been made to provide a trading service based on a X.500 directory (see [Dudet] and [Almgren]). The approach taken by [Almgren] is of a layer structure, with a mapping of functionalities and data structures between a Trader and a Generic Name Server (GNS[1]) which have been investigated. The study shows that a trading layer can be structured over most of the name servers including X.500, although a trading service makes very simple use of the functionalities offered by these name servers. The main benefits come from the consistency and availability this structure provides to trading.

The approach taken by [Dudet] is of a direct mapping between an X.500 directory and a trading service. This direct mapping is possible, although difficulties result from the differences between the information models used, especially relating to the notions of types and properties. A straightforward mapping between the information classifications satisfies both models, but is to the detriment of the flexibility of the trading service. On the other hand the definition of a particular X.500 entry structure which would fit trading purpose is against the genericity of the standard.

These considerations don’t take into account the ANSA naming model issues that can be raised when using an X.500 directory service. More information can be found in the architectural report on the ANSA Naming Model [AR.003].

4.2 Trading service, and traders

A trader is defined in engineering terms as an object that performs the activity of trading on the basis of the whole or part of the set of service offers.

The trader does not have a particular status in the ANSA architecture, but must be considered as any other object in the environment. As any other object, a trader can use all the functions and techniques provided by the ANSA Engineering Model. Transparency functions can be introduced to abstract from the issues related to concurrency of invocation of operations, failure and recovery of failed objects, location, replication to enhance availability,

1. GNS has been defined for the purpose of the study and presents a generic interface, and encompasses most of the functionalities provided by name servers. It can be seen as a common denominator between name services.

migration, and others. Techniques such as atomicity, transaction, or group functionalities can also be used.

We saw in the computational model that no constraints are imposed on the pattern of interconnection of trading contexts. From an engineering point of view, there are also no constraints on how contexts are distributed and replicated into traders.

Trading contexts are created from a computational view-point by invoking a factory service. When a new context is created it can be linked to other contexts by importing their references in the new context, or by exporting the new reference to other contexts. Using a factory service doesn't mean that a new engineering object will result from the creation of a context, it may just be an instantiation of a new interface of an already existing object. But, again, the way contexts are associated to engineering trading objects results from distinct trade-offs that will be faced by the implementor of the trading service.

4.3 Distribution and replication of information

The way in which the trading database of offers is partitioned, replicated and distributed may have strong implications on the efficiency of the trading activity. In general it will pay to place a trading object close to the objects that will interact with it frequently. This means that the contexts and type structure need to be so designed that this partitioning coincides with the locations of objects that belong to those domains and register and search the database for services of that type. The designer is given considerable freedom to structure the database so it may suit the environment in which it is to operate.

In situations where objects are extremely mobile, a suitable co-location policy may not be appropriate. A more centralized trader may then be a solution. This does however introduce some scaling problems as well. As the number of objects in a system increases, so will the number of offers in the trading database, and so will the number of interactions with the trading database. In some cases, this may lead to delays, and the only solution is to partition the database and introduce several traders. The precise trade-off between the number of partitions and traders and a reasonable throughput of trading related interactions must be decided in each application.

Replication techniques may be used to improve the dependability of the trading service. It is clear that keeping several replicas of (parts of) the trading database consistent will introduce an overhead caused by additional interactions between traders. The consistency of the replicas can be offset against the availability of the trading service. In case dependability is privileged, the spread of the information between the various replicas of the trading database will not be instantaneous: importers and exporters might be confused by different view over the service offers space.

4.4 Interface References

Services are accessed through the use of interface references. These references are names. They consist of an information structure composed of a set of alternative addresses to be used for coexisting protocols, plus some end-to-end check data. Relocators can be used transparently for objects that move frequently. As references are moved from place to place they may need to be

edited so that access to the referenced object will follow a correct path through various infrastructures. This requires that we be able to detect references in arguments and results. The type descriptions of the operations holding reference are contained in the type description of the interfaces to which they belong. Interface type descriptions can be intercepted during trading.

A full description on how interface references are constructed, used and intercepted can be found in [AR.007].

4.5 Type management

To match service offers and requests, the type descriptions of both the required and provided services must be made available to the trading service. This is achieved by using *type description services*; services whose purpose is to represent types at run-time, and which the trading service can use to test conformance. This conformance test can be performed by a separate *type conformance service*, as explained on page 11.

The way in which these three services are co-located, distributed, and/or replicated, is a matter for the implementor of the trading service, and depends on the required functionality, reliability and performance of the service. Two possible strategies are:

1. the type conformance service consists of a dictionary of type names chosen by the programmer, each accompanied by a record of the names of the types to which it conforms. Traded types are simply represented by their names, and the type conformance database is maintained by hand by the trader administrator. This simple solution suffers from the possibility of error when updating the database of type relationships, and requires agreement on type names throughout the trading community.
2. an automatic type conformance checker can be implemented using the formal type conformance specification given above. This is more complex to implement, requiring support from IDL compilers and other program construction tools that build interfaces. However it does remove a possible source of human error in trading.

5 Federation

Two concerns have to be considered when designing a scalable distributed system: heterogeneity of resources and autonomy of administrations. When distribution spans organizational and technology boundaries it becomes necessary to consider the overall structure as an evolving federation of cooperating autonomous systems: the federation is not itself a distributed system in the sense of being a superior naming, management and security domain.

This chapter introduces the concept of federation and presents arguments in its favour.

5.1 Introduction to federation

A federation is an organisational structure in which the parties (administrations) negotiate the extent to which they wish to share resources, and thus surrender their exclusive authority over those resources. This structure contrasts with hierarchies in which parties rely on control by a centralised organisation and authority for interworking. The concept was first introduced by Heimbigner and McLeod in their work on information bases (see [Heimbigner 85]). Four principles were identified that characterise a federative approach to the interconnection of computing components:

- a component must not be forced to perform an activity for another component;
- a component must have freedom of association with respect to the federation;
- each component determines which resources it wishes to share with other components;
- each component determines how it will view and combine the resources that are made available by other components.

In summary, federation permits a partial sharing of resources and cooperation by negotiation.

5.2 Motivation for federation

Current paradigms for distributed computing focus on developing an ever growing infrastructure to provide a distribution transparent “homogeneity” based on uniform models of communication, naming and security. Hierarchy is the dominant concept for providing scaling: hierarchy requires adherence to the same standards for communication, naming and security mechanisms across all systems within the scope of the hierarchy.

Hierarchies work best when they are pre-planned to accommodate known or anticipated requirements; practical experience shows that file systems and

class libraries wear out and require rearrangement when extended beyond their original scope. Changes low down in the hierarchy have only local impact and can be tolerated: changes near the root have global impact and thus are effectively prohibited. This freezes the initial structure for all time.

Unfortunately “the” global distributed system has not been pre-planned, nor is it every likely to be. Each standards organization or vendor may feel it has the prescience to conceive “the” global distributed system, but market forces, technical innovation and human perversity will ensure that several such global systems will have to interoperate.

Even if the commitment to agreeing a single standard were in place, technology and organizational change would undermine it. Technology changes continuously; company re-organizations, mergers and sell-offs occur in much the same time frame, whereas it takes five to ten years to achieve convergence in a global market. Thus the problem that has to be faced is that the components and the users of distributed systems will change faster than systems can be installed.

Distributed systems are installed piecemeal, each initially as a self-contained homogeneous environment and then interconnected to other islands of homogeneity as new applications are proposed to link the capabilities of previously autonomous systems. Users will not be prepared to pay the price of converting the systems to some common technology base to enable interworking; if the systems belong to different organizations the people involved are unlikely to relinquish their management controls to one another, or to some hierarchical superior.

Thus the key to practical distributed systems is **federation** of autonomous systems. Federation is an approach to interworking between systems without recourse to central authority for anything: no hierarchy to fit into and no coercion on how each system organizes its internal affairs. Each system (perhaps domain is a better term) in a federated system is responsible for itself. There must be freedom of association.

5.3 Achieving federation

Federation must address two issues: **mutual suspicion** and **technical discontinuity**.

Mutual suspicion requires an approach to security which emphasizes accountability for processes that go across the federation in addition to access control to individual objects since, in a world governed by contracts rather than central authority, the only response to misdemeanour or negligence is recourse to the law. Consequently there is a need for security models that not only feature authentication but also concepts such as transfer of privilege and audit.

The processes that cross administrative boundaries will be governed by interworking contracts between the organizations involved: there will be active components - interceptors - in each domain that enforces or at least monitors interaction relative to contracts. Central authority cannot be relied upon to provide interception for several reasons: there may be no authority to which both organizations are prepared to defer; if such an organization does exist, it is inevitably slow to respond and will take too long to accept and propagate changes.

5.4 Federation principles

A few principles have been identified in the ANSA architecture as the foundation of federation. These principles are:

- context relative naming;
- trading;
- references;
- abstract data types;
- interface references;
- encapsulation;

The arguments for context relative naming are all political. These arguments can only be overcome by the global agreement, central allocation and sanctions to avoid pollution by rival schemes. There is no fall-back position if agreement cannot be reached and the result will be name clashes in a merged system.

Context relative naming (in contrast to global naming) gives individuals autonomy in choosing names and allows interconnection of instances of the name space (e.g. instances of the same distributed operating system) since every name can be traced back to its source. This autonomy is particularly useful if for cloning a system (e.g. by selling copies of it on floppy disc). Each disc can be identical, but is distinguished by the context in which we install it.

Context relative naming is a simple, mature and stable technology: it will always work, without the risk of name clashes and can easily accommodate “global” name spaces (in context). Thus as an approach it offers less risk and competitive advantage. Global naming can however be used as an optimization in a flat context, so long as it is not forgotten that it is a context and to regenerate the handle when names are moved out of it. The utility of absolute names can be achieved by convention: interested parties can erect identical pseudo roots above the point of interconnection for consistency; they can do so without concern for the apathy or opposition of other parties.

Trading is the process of finding things by description rather than by name. The arguments against the stability of naming systems mentioned earlier lead one to the conclusion that it is misguided to expect names to remain meaningful for very long, except in the local context. Therefore means have to be provided for applications to find resources and services by pointing at some organizational context that is trusted and describing what is required rather than relying on simple naming conventions. For the ANSA trading service, the description consists of type signature and properties.

The results of trading are references. References are names. Since they are found either by trading or as arguments or results of interactions they need not be meaningful to people, provided that the system can unpick them. They are names and not addresses since objects may relocate between the time they are traded and the time they are used. The reference must contain enough information to enable relocation of the object[1].

Abstract data types (i.e objects) give very simple semantics for interworking and portability: they divorce programs and data from issues of representation. Everything is described in terms of behaviour: what you can do to it.

1. Interface reference and relocation are described in [AR.007].

Everything has an interface containing operations. This is sufficient as a model for programming: it works for simple objects like integers and booleans; it works for system objects like semaphores and files; it works for complex objects like databases. Code generators can recognize that the behaviour of simple objects can be mapped onto bytes and instructions. There is no need to force the programmer to worry about this. Moreover if a type is missing in some environment, and access to it is via operations, it can be provided as function, rather than being built in, easing the migration of software between systems.

References provide full distribution transparency: there is no need to care from a computational point of view, whether a service is provided in the same address space, on the same computer, or remotely across a network. Mechanisms can be transparently inserted to give engineering guarantees (e.g. replication for availability) and change the system designer mind about which guarantees he wants without having to change the source: it would let our compilers generate correct engineering code rather than let programmers build it incorrectly.

Interworking also benefits from the use of abstract data types: if arguments and results take the form (from the programmers view) of references to abstract data types, then there is complete abstraction away from the concrete representation aspects of the communications (e.g. protocols, shared memory) and interactions can occur over **any** communications channel.

Encapsulation can be stated simply as the principle that every object should defend itself and be clearly separable from the supporting infrastructure. An object should see the infrastructure as just another service, albeit an indispensable one. This gives the ability to move software between infrastructures and to apply different management policies to different objects. (In the worst case of federation every object would be autonomous.) Clearly there are benefits in sharing mechanism, but this should be a local configuration decision and invisible to objects in other domains.

6 Federation and Trading

Federation appears to be a key concept in the development of scalable distributed systems, and for building distributed systems from previously disconnected components. In both cases the notion of *boundary* is critical: to mark the point at which administrative and technological autonomy is permitted. Where a boundary exists, an *interceptor* is required either to filter out undesired interactions, or to enable previously impossible ones.

The word interceptor used in this document refers to the agent that has to be introduced at a boundary to realise actions such as: checking, notification, authentication, conversion, translation.

In the first part of this chapter, federation is described in terms of boundaries. A description of how trading can be provided across boundaries is then provided, and the concept of interceptors is introduced.

6.1 Boundaries

This section attempts to classify the boundaries, and describes how trading is related to them.

6.1.1 Administrations

An administration consists of a set of members that are subject to a common source of authority and that share a collection of goals, values and beliefs. Boundaries between administrations may coincide with one or more of the following:

- Organisational boundaries
- Accountability boundaries
- Judicial boundaries
- Economic boundaries
- Contractual boundaries
- Social boundaries
- Political boundaries
- Cultural boundaries

It is important to identify these boundaries, as they have a profound impact on the structure of the system and extent to which representations of values in the computer system can be manipulated by interceptors.

Crossing administrative boundaries requires a great deal of skill in mapping values and beliefs. This mapping must be left to human beings as only they have a grasp of the problem in the enterprise perspective.

The relationship between administrative domains is modelled by hierarchical or federal links: federation is not the only way to cross administrative

boundaries. The hierarchical model is of a strict dependency between an administration and a sub-administration, which implies a transfer of authority and goals.

6.1.2 Classification schemes

Classes are a means by which objects can be classified in sets. The reasons for adopting one classification scheme rather than another are steeped in the values and beliefs of those that are expected to use the classes.

Many classification schemes exist. Objects may usefully be classified according to:

- the way in which they interact (their type);
- they way in which they deal with names;
- the way in which their properties are derived and expressed;
- the technology used for their interconnection;
- the way in which they are managed;
- their significance in terms of access rights to them.

For each of these (and other) classifications, the semantics of the objects must be studied carefully.

Crossing class boundaries can be automated, but is restricted to the extent to which sets of rules can be derived from the comparison of values and beliefs (an enterprise issue).

The ANSA information model is intended to provide a language in which the shared conceptual schema of interacting objects can be captured.

6.1.3 Type system boundaries

The ANSA computational type system classifies interfaces according to their operation signatures (operation names, parameters, termination names, and results). It specifies the rules by which conformance between two types can be established [AR.001]. This is clearly a central concern in the provision of trading services.

Different type systems may differ in:

- the language used to describe interfaces;
  The languages used to specify the interface of a service (e.g. IDL in ANSA) can be different. In ANSA, as in other architectures, the description of the interface is independent of the particular implementations of a service. In the trading service computational model, this interface specification is the one represented by the type description.
- the encoding of data;
  Different encodings of data can be used. A type system boundary might impose differences in the way that abstract data types are represented on the wire. This encoding is usually related to the language used to describe the interface. However, it can also be considered as a binding issue and can be addressed when addressing interconnection boundary issues.
- the way conformance between interface declarations is established;
  Two environments might associate divergent meanings to the concept of *conformance*. For instance, particular systems determine conformance

using concrete type representations whereas in other systems (as in ANSA), conformance might be based on abstract data types.

- the interface type of applications that cross boundaries.
  An application that appears in several environments might be implemented by objects that have different interface types in each environment. These differences in the interface type must be made transparent to an invocation that crosses the boundary. These differences must also be made transparent in the trading service when realising a search across a boundary.

Crossing type system boundaries requires recourse to the semantics of the classification scheme employed (the abstract types in an agreed representation) [RC.358].

6.1.4 Naming system boundaries

Differences between naming systems result in:

- different naming domains;
  Since this is a desirable feature, trading across naming domain boundaries results in gaining knowledge about objects in other domains. It is therefore not a problem.
- different naming conventions;
  Differences in naming conventions can only be resolved by considering the semantics of the names so that meaningful mappings can be generated.
- disjoint naming contexts;
  The context relative naming conventions adopted in ANSA require disjoint naming contexts, so this again is not a hindrance.
- differences in naming network structures.
  Differences in naming network structures may require the statement of searching policies to be converted. In some cases this may not be possible without human intervention.

More information about the ANSA naming system can be found in [AR.003].

6.1.5 Property boundaries

Properties provide a finer classification of the services by associating information about the service to the interface type. This classification enables a distinction to be made between the abstract services that can be found behind an interface. Unlike interface typing which is based on formal description of interfaces and can then be mechanised, property classification relies on an abstract description of the service and requires human interpretation.

When a new service instance is created and exported to the trader, the association of values to properties permits it to be distinguished from the other service of the same type. Property boundaries may appear where:

- differences can be found in the set of properties used to characterize a particular service;
- there is a discontinuity in the way properties are expressed;
- there is a change in the way constraint expressions on properties are provided to a trading service.

When crossing a property boundary it will be necessary to provide translations of properties. If a property is unknown on one side of the boundary, it must be left out. If however another similar property exists, then a mapping may be possible.

6.1.6 Interconnection boundaries

Technological discontinuity must be assumed to be inherent in distributed systems and their federation. Differences in the technology used for interconnecting objects can be found in:

- protocol used;
- addressing scheme.

Usually, interceptors are provided to hide such technological differences from the applications. However, interconnected systems may have different ways of constructing interface references.

The service offers contain information about the ways in which they are prepared to be connected to other parts of a distributed system. The trading system needs to be able to deal with differences in which this information is provided and represented.

Remote interaction may introduce communication delays which may affect the behaviour of interconnected trading services (time out etc.).

Gateways by themselves may require to cooperate with the trading service in order to make boundaries transparent. See §6.2.9 for more information.

In summary: the provision of a trading service between interconnected systems needs to consider:

- the way interface references are constructed;
- the way technological information is represented and used in trading;
- the topographical issues.

6.1.7 System management domains

All managed objects within a particular system management domain are subject to the same set of management policies. Within a management domain, management constraints are expressed and applied uniformly.

Objects in ANSA manage themselves, that is, each managed object contains a component that can respond to system management operation invocations. In different systems or different parts of a large distributed system, such operations may take different forms. Since system management concentrates on allowing human operators to monitor and control the way objects use resources, it mostly pertains to local cost optimisations.

The objects that provide the trading service can pertain to distinct management domains. The domains can have different management structures or management policies. These differences introduce discontinuities that will have some effect on the provision of a cross boundary trading service.

The effect of relocation of services on the accuracy of the information held by the trading service must also be considered.

6.1.8 Security domains

All objects within a security domain are subject to the same security policies. Within a security domain security constraints are expressed and applied uniformly.

The security framework developed in ANSA allows multiple security policies to coexist. Spanning security boundaries needs the security policies to be "federated" (the word is used to qualify an interconnection between security policies where each party maintains his autonomy. In this use of the word there is no organisational connotation). More information on the ANSA framework for federating secure systems can be found in [RC.322].

Trading has to be provided across security domains. Two aspects are relevant to trading and need to be considered:

- trading is an application and, just as any other application, has to fit into the security structure;
- the sharing of information within a trading service also has to fit into the security structure.

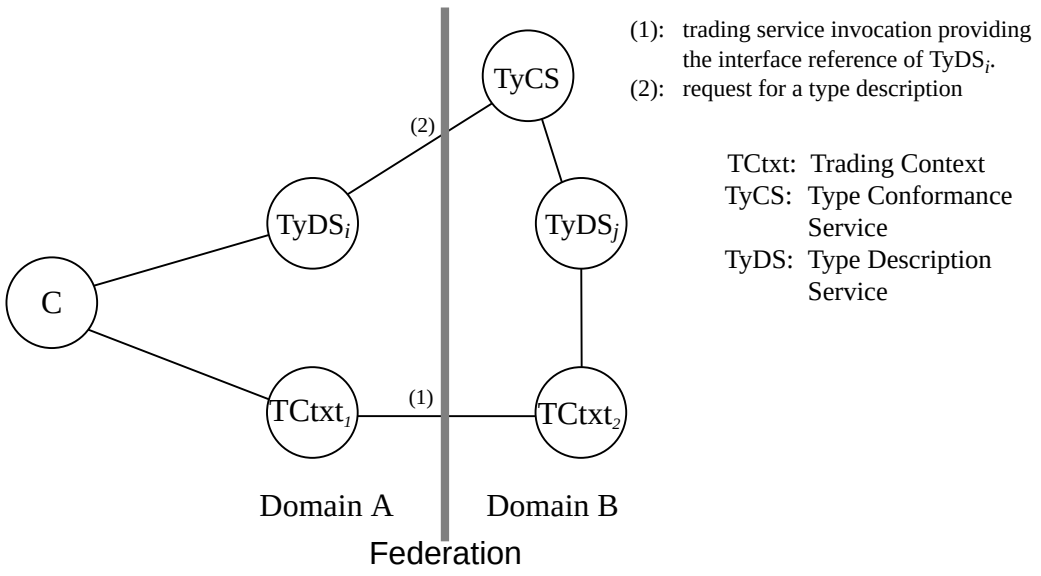
6.2 Trading across boundaries

6.2.1 Cross boundary model

Providing a trading service across a boundary requires interconnections between objects from both sides taking part in the service.

Providing a trading service across a boundary requires the trading models of both sides to be interconnected. It may require trading contexts (TCtxt) to be interconnected, and the type conformance service (TyCS) of one side to invoke the type description service (TyDS) of the other side. An interconnection scenario based on the computational model of trading introduced on page 11, is shown figure 6.1.

Figure 6.1: Cross boundary trading model.



Other configurations from the one presented by figure 6.1 are possible: a client may interact directly with $TCtxt_2$ without passing through $TCtxt_1$, or $TyDS_i$ can be located in Domain B. These cases may introduce the same or fewer cross boundary issues than represented in the model. In the figure, the trading context $TCtxt_2$ asks $TyDS_j$ for a conformance test to be done with $TyDS_i$. The request for a conformance statement could have been directed to the type description service of the client ($TyDS_i$). It seems however that this configuration, although slightly more complicated, would raise the same issues.

Figure 6.1 highlights that the provision of a inter-domain trading service requires two invocations to cross the boundary: an invocation of a trading context and a request for a type description.

The figure suggests that no more than two type description services need to be used when testing for conformance. This might not generally be the case since the test is a recursive process. TyCS might be required to interact with more than two type description services.

6.2.2 Administrative boundaries

An administrative boundary represents a cooperative link between administrations: any transaction across the boundary necessarily undergoes a change in management responsibility. Each administration may wish to impose access controls according to its security policy, and monitor the interconnection for accounting purposes, etc.

Administrative boundaries require interception of interactions at the interconnection point. The purpose of the interception is to check, to record and to transform the interactions. The interceptors do not substitute for the controls imposed by the objects themselves.

The trading service is already highly influenced by administrative considerations. A trading context may have security policies that dictate how information can be provided across administrative boundaries. If this control is insufficient, interceptions can be used to gain more control. Otherwise interconnection of trading service does not require more interceptions at administrative boundaries than any other application.

To provide the trading service across a boundary, two interceptors are needed. One to realize interception between trading contexts, and another to intercept the invocation of the type description service.

Each party defines the set of service offers that will be provided to the members of the federation. This set of service offers is called the *exported context* in §3.4 on page 18. An administration can implement a particular trading context that receives all invocations from other administrations.

The following two figures present how the trading service interconnection model can be modified to introduce interception at administrative boundaries.

Figure 6.2 presents an inline solution. Inline interceptors are placed in the invocation paths that crosses the boundary. Their rôle is to intercept any invocations that cross the boundary and to apply specific access control. The localization of interceptors can be at the source of the invocation, at the destination, or at both ends.

Figure 6.3 presents an off-line solution where “negotiation” is made before the interaction takes place (See [AR.007]). These off-line interceptors, which are not in the invocation path, provide communication between both parties such

Figure 6.2: Inline interception for administrative boundaries

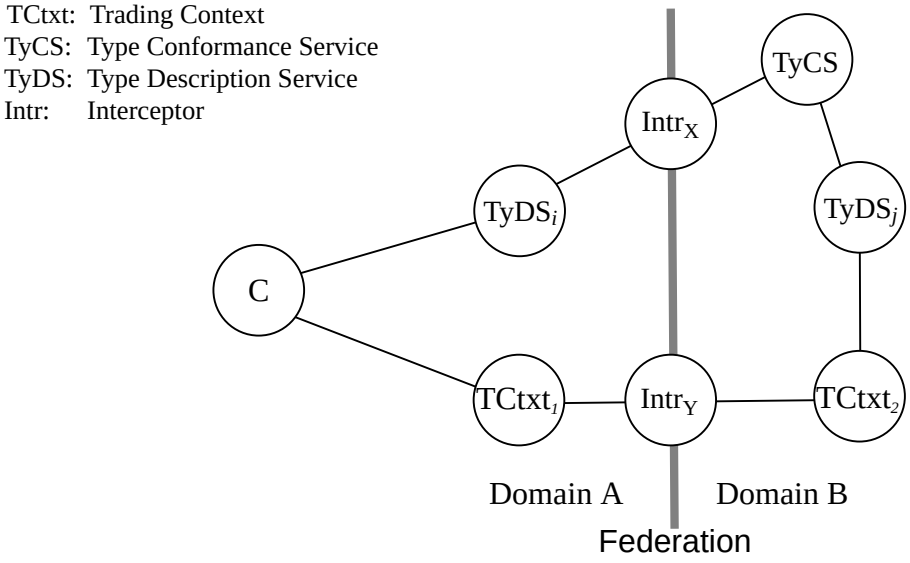
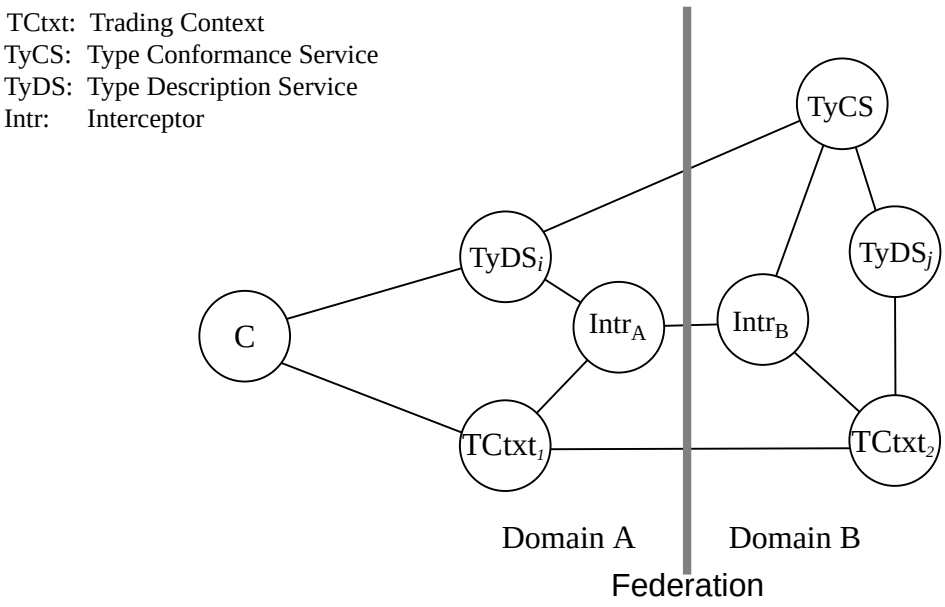


Figure 6.3: An off-line interceptions for administrative boundaries.



as negotiation, check, etc. Before a client starts any cross boundary interaction, it invokes the service of an interceptor. This interceptor negotiates with another interceptor in the other side for a cross boundary invocation. The action can be to authenticate objects and provide cryptographic keys for the interaction, for instance. This interception can be required before each interaction, or may allow several successive interactions.

The figures do not show how the interceptors will interact with the trading model objects. Mutual suspicion dictates the requirement to have interceptors distributed in two symmetrical and bi-directional halves, one in each administration.

These interceptors will have to fit into the security, the monitoring and the management frameworks.

6.2.3 Type systems boundaries

Type system boundaries influence the provision of the trading service in two respects:

- the first requirement is that the objects that compose the trading service must be able to interact between themselves;
- then, in providing the service, the type description provided by the client has to be understood in the other environment.

The differences that can be found in heterogeneous type systems were discussed previously, and can be listed as differences in:

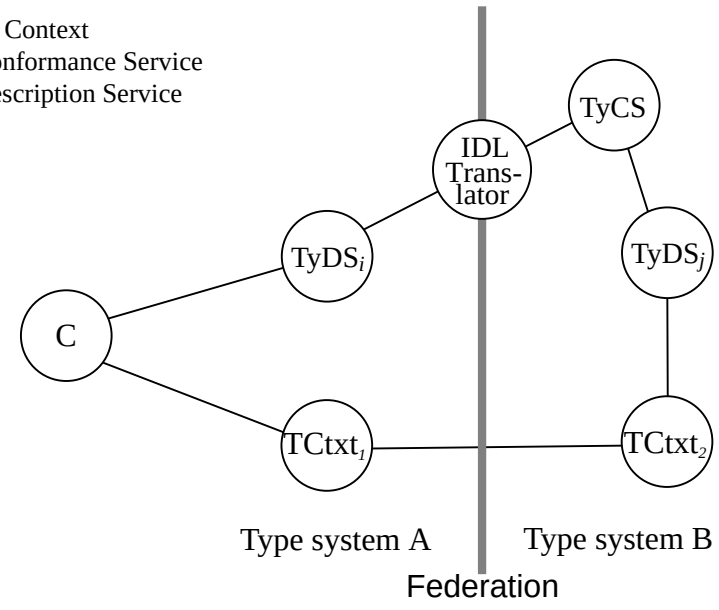
- the language used to described interfaces (expression, grammar);
- the encoding of data;
- the way conformance between interface declarations is established;
- the interface type declaration of applications that cross the boundaries.

The following sections show how solutions using interceptors can tackle these differences.

6.2.3.1 Type description language translation

Interconnection of type description languages can be done by an interceptor introduced between the type description service of one side and the type conformance service on the other side. The purpose of this interception is to translate the description of interface made in one language into a description that is understandable in the other language.

Figure 6.4: Type description language translation.



The definition of this translation may not be an easy task. Several problems may arise that can introduce limitations in the interworking or an erroneous conformance statement.

Some types might not have any correspondences in the other environment.

Ambiguous translation is particularly hard to tackle.

Before tackling the issue of interconnecting type description languages, differences between the interaction models should be studied. The interface description languages are designed to fulfil the requirements of interaction models. Tackling differences in languages should be more than realising just a protocol conversion, it should require us to consider at an abstract level the mapping between the interaction models.

The location, or the distribution of the interceptor (if mapping is possible) will emerge from these considerations.

6.2.3.2 Data representation

Differences in the way the data is encoded “on the wire” is a protocol interconnection issue. Some systems may support various interaction protocols to minimise the risk of protocol clash, and to avoid having to use protocol convertors. Section §6.2.5 on page 38 shows how technical discontinuities will influence trading.

6.2.3.3 Understanding conformance

Two environments may have divergent views on the way conformance between services is established. In ANSA conformance is achieved when something of one type can be replaced by something of another type without introducing interaction errors.

Some systems determine conformance using concrete type representations.

Other systems determine conformance using semantics associated with the services. This is usually done to avoid type clashes. Type clashes occurs for example when two interfaces share the same signature for operations that provide completely distinct services.

This is a type conformance service semantic problem. It may introduce failure in the search of service offers, as well as misunderstanding the requirements of a client. It introduces constraints in the transfer of searching policies between systems. Differences in meaning of conformance may be caused by differences in interaction models used by each system.

In the ANSA computational model (see [AR.001]), conformance is defined by recursive rules, which means that an unspecified number of conformance checks may be required to test the conformance of particular interfaces.

Names can be given to types, and a type library be constructed by exporting type description service interface references to a “type library”. It would provide all the trading facilities to navigate the type graphs, as well as associating information with the types. This extra information would provide semantics to the conformance test. Interconnecting type spaces thus becomes linking library contexts; and the issues described in this section can be approached as trading system interconnection issues.

6.2.3.4 Application interconnection

Considering the trading service application, a trading context object in an environment *A* might have a different interface type to a trading context object in an environment *B*. This difference in interface type can also be found in the type description service. If no transparency is provided, a trading context has to be aware when making a query in a foreign environment that the interaction will not be the same as for internal queries. In particular cases where the interface types are similar enough to allow a mapping of their

operations, an interception can be carried out transparently between the objects.

Discontinuities in interface type can be found in many applications. This is clearly not particular to the trading service.

If interceptions are introduced to realise interface mapping for a particular application, it will have to be taken into account during the trading process that might precede the interaction. The discontinuity and the mapping must be known when stating the conformance.

6.2.4 Property boundaries

We have introduced three different kinds of discontinuity in the previous chapter:

- differences in the set of properties associated with a particular type;
- discontinuity in the way properties are expressed;
- change in the way constraint expressions are provided to the system.

A service can be classified in different environments by different sets of properties. These differences come from distinct points of view in the values used to qualify services. As example, consider a printing service that has a *cost_per_page* property which is not present in all environments. A trading request that crosses the boundary will either fail, or just take into account the properties of the constraint expression that are part of the intersection of the environments property sets. This issue needs to be considered from an information viewpoint.

Discontinuities in the way properties are expressed is typically a naming problem. In our printing example, *quality* in one system can be called *qualité* in a peer system, or the value *A* associated with a *quality* property can correspond to a poor resolution in one side and to a high one in the other side. Such differences require the intervention of a person to define the mapping of the names. A name bridge object can be introduced between the client and the trading context. This boundary is service type specific. One particular property naming bridge should be defined for to each service type.

The last discontinuity is a generic issue that can appear when interconnecting trading services. The way constraint expressions are defined is part of the information models used for trading. In other words, the property classification can be part of a general schema that embrace all the information available and relevant for trading. Mapping constraint expressions is then an information viewpoint issue that cannot be addressed yet since a detailed information model has yet to emerge from the work of ANSA. This problem has been addressed by Litwin et al [Litwin] (INRIA) in their multi database system MRDSM.

6.2.5 System interconnection boundaries

When studying the boundaries in the previous chapter, three issues that are related to the provision of the trading service across technological boundaries were identified.

- differences in the way interface reference are constructed;
- differences in the way technological information is provided, represented and used in trading;

- and the topographical issues: local system, local and wide area network boundaries and differences in network topology.

Technological boundaries can put forward requirements for interception to be realized within the communication infrastructure. Interception can introduce some specific requirement to the trading service. This issue will be considered later on in a separate section.

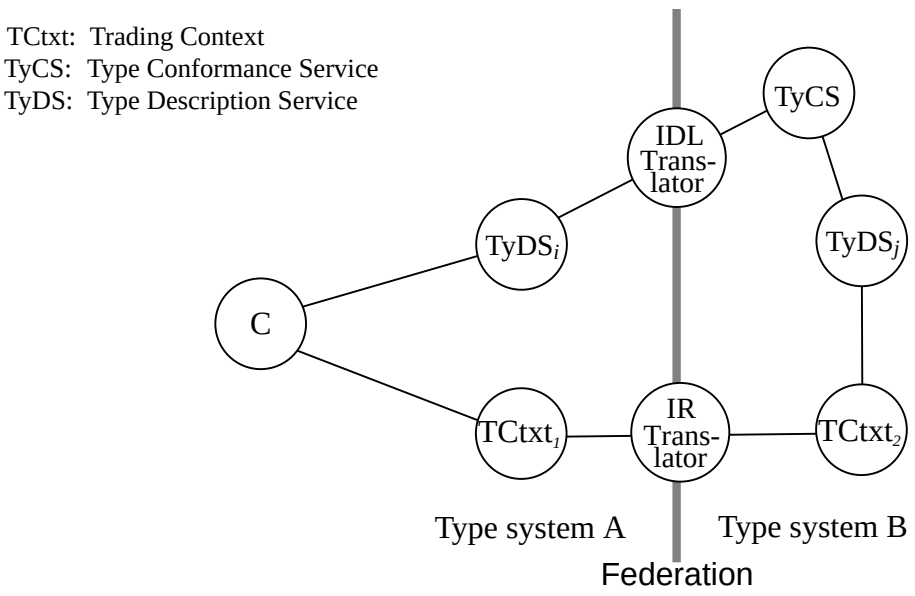
6.2.5.1 *Interface reference*

In ANSA, addressing is achieved by the use of interface references. An interface reference is an information structure composed of protocol hints and addresses. It allows objects to give context-relative location-transparent names to interfaces (see [AR.007]).

Interface references might not be constructed the same way in different systems. This difference is very natural since different technologies place different requirements on an interface reference structure.

Interface references can be expressed in the ANSA interface description language as a data type constructor. Different interface reference structures can then be tackled as differences in interface description languages, using IDL translators (see previous §6.2.3.1). Translating objects can then be introduced in the trading model to overcome interface reference structure discontinuities. Translations are required for the interaction with the type description service and for trading context object interactions. Any interface reference that crosses the boundary needs to be translated (interface reference of the type description service, of a provided service, etc.). Interceptors can be introduced as presented in figure 6.5.

Figure 6.5: Interface Reference translation.



A particular case might be when an interconnection has to be realized with a system that uses a global addressing scheme (i.e. a system without interface reference abstraction). From the ANSA side, external addressing and references to other protocols can always be enveloped within an interface reference. From the opposite side the problem can be difficult to tackle and may require the use of complex bridges, association tables, etc.

6.2.5.2 *Technical information*

From an information viewpoint a service offer and in particular the data included in the interface reference gives the trading service the possibility of extracting a great deal of technical information about the services. Examples of information that can be extracted are:

- the different protocols that are supported by a server or its infrastructure;
- the type of infrastructure that support the service;
- the location of the service;
- the use of distribution transparencies (e.g. replication);
- data associated with quality of service considerations.

All this information can be used in trading for purposes like service offer classification, or expression of searching constraints.

A discontinuity may appear in the way this information is extracted, represented and used by the information systems on different sides. It represents a discontinuity in the information viewpoint in the data used and the associated semantics. How different information structures can be brought together is outside of the scope of this report. It is an issue that needs to be discussed in the context of an information model.

6.2.5.3 *Topographical discontinuities*

The physical separation of interconnected systems has to be considered when trading. Distances directly translate into communication delays. The whole set of offers located at a distant site will suffer from an introduction of a time constant in the communications. These delays might be ignorable for some services, but can be significant for others.

Topographical discontinuities form a constraint on the provision of the trading service: some data may have to be reconsidered (such as performance properties). As for interface reference information structure discontinuity, this pertains to the information viewpoint and requires to be discussed in the context of an information model.

6.2.6 System management domain

When interconnecting system management domains several issues that have direct influence onto the trading service arise:

- differences in the management structures or in the management policies that drive the trading service;
- management actions on services may influence the validity of the information maintained by the trading service.

Systems interconnection requires management domain interconnection. If not enough attention has been paid when considering these issues, management policy clashes may be observed in the interaction or in the semantics of service provision. Interaction between trading objects in both sides might be restricted and controlled by management policies since management is dictated by considerations such as security, accountability, etc.

Trading objects follow management policies that are provided from three sources:

- the nucleus that dictates most of the constraints, freedom, and requirements that are applied to an object;

- exporting a service offer into the trading service can be seen as introducing a management policy relative to the offer (the context, for which request it can be provided, which clients, etc.);
- importing a service offer can also be seen as introducing a management policy into the trading service (expected service, properties constraints, searching policy, etc.).

Differences in management domains then influence trading in the transfer and the interpretation of the policy that comes from the client. How, for example, should searching constraints be propagated? Thus management policies directly influence the semantics of the trading service. This is an issue that need to be considered from the information viewpoint.

The second point noted in this section was related to the consistency of the service and the related service offer maintained and possibly replicated by the trading service. For consistency, the trading service must be notified each time a significant change in a server configuration is made. Relocation of server does not require an immediate update of an existing offer, it can be made transparent by the use of a locator. The accuracy of information has also to be maintained for the information that is shared by the cooperating trading services.

6.2.7 Security domains

Considering trading in terms of the security architecture:

- trading is an application and just as any other application, fits into the security structure, i.e. its interactions are controlled;
- the sharing of information within a trading service also has to fit into the security structure.

The various objects that form the trading service are subject to security policy as for any other application objects. Interactions at the interfaces of the objects must be controlled. Because the security domains remain disjoint in a federation, the cross boundary interaction would have to conform to the security policy of each side.

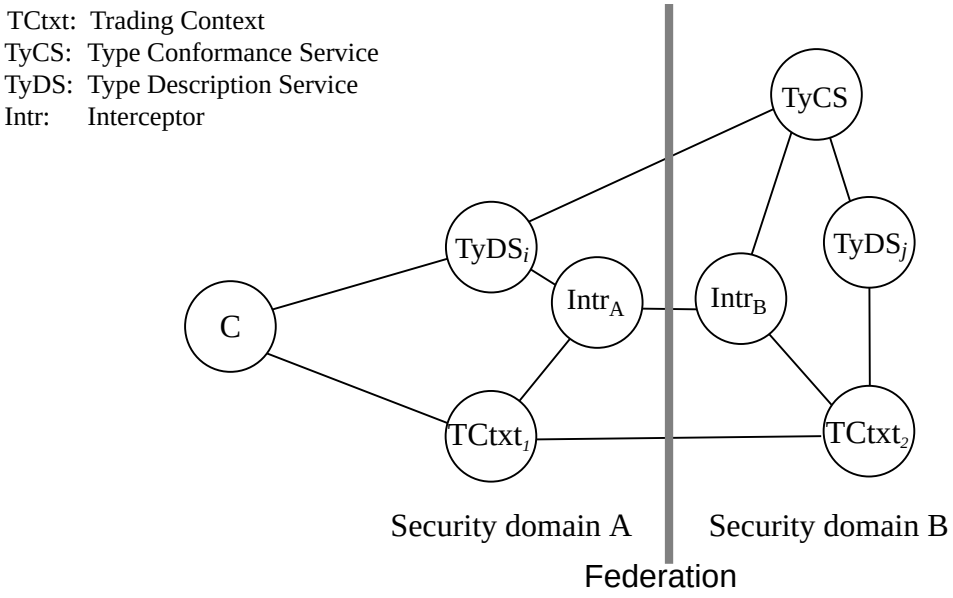
The information transfer realised by trading services also has to fit into the security structure. The objects that perform the trading service may restrict the sharing of data according to the clients. However, trading can not be restricted to be performed by some objects, since all objects have the ability to trade (i.e. transfer information about other services). This may require interceptors to control the transfer of interface references between security domains. It is however the responsibility of the objects finally to decide whether to supply a service.

Interceptions motivated by security considerations have to be performed on each side of the discontinuity, an interceptor being required to enforce each of the security policies. Figure 6.6 presents an off-line introduction of interceptors. Other solutions can use inline interception or may combine both type of configurations.

6.2.8 Information viewpoint issues

In the preceding pages, information viewpoint issues were raised that are relevant to naming, or to semantic considerations on names or services. Information viewpoint issues are the most crucial and difficult to tackle

Figure 6.6: Off-line interception for security boundaries.



because no mechanisms can be introduced to handle completely these concepts and human intervention becomes necessary.

Federation will then be highly dependent on how naming will be carried out and how information will flow from one environment to another.

An ANSA information model is currently under development. It will provide a framework for communicating information between groups that have potentially different values, policies and objectives.

The following information viewpoint issues were exposed in the previous chapters:

- property classification discontinuity; differences in data used to classify services of one type, differences in how information is extracted from this data for the purpose of trading and differences in the construction of constraint expressions;
- differences in the way conformance is defined; which affects to the semantics of the trading service (and particularly the TyCS objects in our model);
- system interconnection discontinuity that expresses differences in how technological and topographical information is extracted and used for trading;
- system management discontinuity in searching policies and the consistency of service offer information.

These points represent some particular and practical issues that can be raised when interconnecting trading systems across a federation. They also demonstrate that these issues need to be addressed at an abstract level. A framework for an information model is needed: as a basis for trading to use, classify and transfer information, and as a framework to understand federation of trading services.

Trading uses three classification schemes (interface type, properties which are application specific and context). Other classification schemes can be introduced in providing a trading service (e.g. Security considerations are

reflected in the context classification, but it might be useful to have a partitioning of the service offer space for security purpose that is independent to the context structure). Independent systems will have difficulties in making their trading services cooperate if they use distinct classification schemes. Each classification scheme is motivated by particular considerations making use of specific kind of information (e.g. the context structure merge administrative, security, accountability and other considerations). Independent trading services will face difficulties to cooperate if they use distinct sources of information to classify the trading offers.

The following appears to be particularly significant in providing a trading service across federation:

- the interconnection of information models;
- classification schemes;
- security architecture;
- naming.

6.2.8.1 Naming system boundaries

The ANSA naming model (see [AR.003]) contains a description of the naming systems used for the provision of the trading service, and discussion on the federation of naming systems. Trading uses many related naming systems such as:

- the invocation naming system;
- the attributive naming system;
- the type naming system;
- the property naming system;
- the administrative naming system;
- and potentially other naming systems (for instance: searching policies.)

As was seen in the boundary presentation, differences between naming systems result in:

- different naming domains;
- different naming conventions;
- disjoint naming contexts;
- differences in naming network structures.

The previous discussion about boundaries could have been made from a naming system perspective. Interconnecting type systems for instance could have been approached as interconnecting type naming systems.

Differences in naming domains are inherent to federation for the administrative, invocation and attributive naming domains. Different domains for type and property naming systems means differences in the concepts that are named. This is not desirable in a federation but would represent a reality.

Differences in naming conventions can be made transparent by using name bridges. Naming conventions can be considered to dictate composition of names in two respects:

- the syntax used to form correct names (that can be automated);
- the association of semantics in the composition of names.

Naming networks have to be interconnected in each of the naming systems used for trading.

6.2.9 Interceptors

Interceptors provide transparency of access across boundaries. They can be called *agent*s, *gateway*s, *protocol converter*s or *monitor*s depending on the boundaries that have to be overcome.

If only an administrative boundary is being crossed the interceptor applies administration specific access controls and re-assigns responsibilities. As already seen in the previous discussion, two different kind of interceptions can be realised: in the invocation path, or before the interaction. The interceptor can be located at the source, at the destination or on both sides.

Interceptions can be made in different viewpoints:

- computational interceptions take place in the interaction model, are constructed for each interface, are therefore interface type specific and will be type checked;
- engineering interceptions take place in the infrastructure and are computational model type generic. They must be able to deal with invocations as well as engineering optimization across the boundary.

Computational interceptions require the type description of the service that will cross the discontinuity. Both interceptions require interface references to be edited for multi-hop addressing or for a capture to be made.

An interceptor can be either fixed or nomadic. Fixed interceptors provide immediate access across administrative and technology boundaries while nomadic interceptors provide store and forward access via media interchange. Only fixed interceptors are relevant to trading.

interface references can be transformed by editing them to provide multi-hop (context relative) addressing or by capturing them in a proxy object and transmitting a reference to the proxy. Data types which cannot be transformed must be instantiated as objects in the interceptor and a reference to them transmitted onwards. The transport of immutable objects may be optimized by transmission of a template rather than an object reference, except in the case of objects that require access controls - this is because the controls could no longer be enforced on the copy.

The full architecture for interception is contained in [AR.007].

6.2.9.1 Need for type description capture

To perform the required interceptions, interceptors need access to the relevant type descriptions.

Passive clusters have their type descriptions embedded in their representation.

The type description of an argument or result parameter is contained in the type description of the interface containing the operation passing the argument or result (i.e. they are part of the same type description). The roots of these type descriptions are the type descriptions of traded interfaces.

The interface type descriptions can be captured when trading before the interaction takes place between the client and the server. It then requires the cross boundary trading service to capture and make the interface type description available for interception. When an interface type has been

captured, all the types that are present in the captured type graph will then be known by the interceptor and may then be intercepted as they cross boundaries.

The required type, described by the client, is used rather than the provided type because this is all that is needed to define the interactions and is the one that is exchanged as an argument of the import.

This means that:

- once a type has been traded[1] then all the types in its signature can be intercepted;
- that no type can be intercepted until its description is known: it has been traded itself or is included in the type description of a traded type.

This then leaves the problem of who trades the trader.

If a trader is accessible through an in-line interceptor then that can be used to trade subsequent traders, but the interceptor must provide special initialisation functions to trade at least an initial trader across itself in both directions (see next section).

This initialisation can be done by having the interceptor federate traders on either side of its boundary, by having the interceptor become a proxy trader or by building a trader into the interceptor itself. Initialising an in-line interceptor is analogous to mounting a nomadic interceptor.

Interceptors may be visible in the engineering or computational viewpoints. Engineering interceptors will be type transparent and thus capable of intercepting interfaces of any type. Computational interceptors will have to be specially constructed for each interface type to be intercepted and will be type checked. This can be done automatically from the type definitions which can either be pre-compiled or interpreted at run-time.

6.2.9.2 Need for interface reference capture

Interface references need to be captured and edited to permit invocations to be intercepted. This can be done since interface references are part of the interface description language and can therefore be recognised within invocations. The capture of interface references can be effected when trading, and more generally during any interaction that cross the boundary.

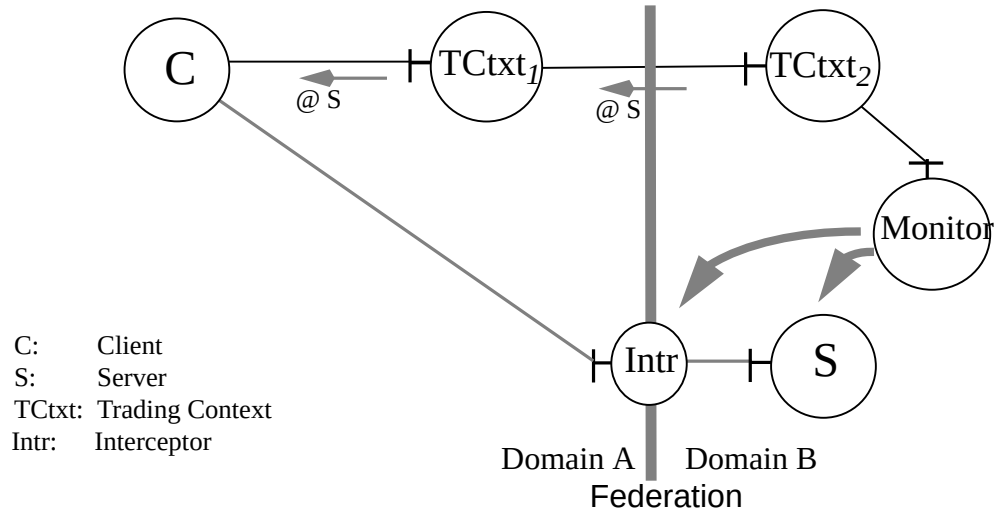
Within the simple trading service model, interactions between components transfer interface references. These interface references have to be captured if interceptors are required to mask boundaries as discussed in the previous chapters.

The result of a federated trading query is the transfer of one or more interface references across the boundary. In the ANSA interaction model, any object has the ability to transfer interface references to other objects. It seems therefore that the requirements for capturing and editing interface references are not particular to the trading service. However, the trading service transfers interface references for new bindings between a client and servers, bindings that are not part of the set of pre-existing cross boundary bindings. New interactions can potentially require dedicated interceptors to be created, records to be kept, etc.

1. Trading is used here in a wide sense and refers to the transfer of information about services.

This is a recursive issue: the trading service is important in federation for cross boundary interceptions; and interceptions can also be required for providing the cross boundary trading service. The recursion can be broken by introducing an initialisation mechanism by a bootstrap interceptor that can be used to start the process and trade subsequent trading services.

Figure 6.7: Introduction of a monitor agent.



When a trading service fulfils a cross boundary request it has to provide the information necessary to realise interception. The trading service can notify a monitoring agent that an invocation is expected to cross the boundary, and provide crucial information such as the type of the interface that will be invoked. The monitoring agent can then instantiate interceptors and edit the interface reference of the provided service offer. See figure 6.7. The monitoring agent (according to management policies) can then accomplish actions such as creating a dedicated interceptor, instantiating a generic interceptor, notifying the server or editing the interface reference.

References

[Almgren]

Almgren, G. and Anderson, M., “Open System Trader using X500”, ISA Project, Televerket, September 1991.

[AR.001]

Rees, R.T.O., “ The ANSA Computational Model”, AR.001.01, APM, Cambridge UK, January 1993.

[AR.003]

van der Linden, R., “The ANSA Naming Model”, AR.003.01, APM Ltd, Cambridge UK, February 1993.

[AR.007]

Herbert, A.J., “Addressing and Interception”, AR.007.00, APM, Cambridge UK, to be produced.

[AR.008]

Rees, R.T.O., Bull, J., “A Framework for Federating Secure Systems”, AR.008.00, APM Cambridge UK, March 1992.

[Dudet]

Dudet, M., “X.500 Directory for those familiar with Trading Concepts”, ISA Project, SEPT, 42 rue des Coutures, BP 6243, 14066 Caen cedex, December 1990.

[Heimbigner 81]

Heimbigner, D. and McLeod, D., “Federated Information Bases - A Premilinary Report”, In *Infotech State of the Art Report: Database*, Vol. 9, Pergamon Infotech Limited, Maidenhead, UK, 1981, pp. 383-410.

[Heimbigner 85]

Heimbigner, D. and McLeod, D., “A Federated Architecture for Information Management”, ACM Transaction on Office Information Systems 3(3), pp. 253-278, (july 85).

[Litwin]

W. Litwin, L. Mark and N. Roussopoulos, “Interoperability of Multiple Autonomous Databases”, ACM Computing Surveys, Vol. 22(3), pp. 267-293, September 1990.

[TR.031]

Howarth N.J., Watson A.J., Rees R.T.O., Otway D.J., “DPL Programmers' Manual”, APM/TR.031.00, APM Ltd., Cambridge UK, February 1993

[RC.358]

Watson A.J., “Types and projections”, APM/RC.358.03, APM Ltd. Cambridge UK, April 1992.