Reference Model 4

4.1 Object Management Architecture

4.1.1 Introduction

The *Object Management Architecture Guide* (OMAG) describes OMG's technical objectives and terminology and provides the conceptual infrastructure upon which supporting specifications are based. The guide includes the *OMG Object Model,* which defines common semantics for specifying the externally visible characteristics of objects in a standard implementation-independent way, and the *OMA Reference Model*.

Through a series of RFPs, OMG is populating the OMA with detailed specifications for each component and interface category in the Reference Model. Adopted specifications include the Common Object Request Broker Architecture (CORBA), CORBAservices, and CORBAfacilities.

The wide-scale industry adoption of OMG's OMA provides application developers and users with the means to build interoperable software systems distributed across all major hardware, operating system, and programming language environments.

4.1.2 Reference Model Overview

The Reference Model identifies and characterizes the components, interfaces, and protocols that compose the OMA. This includes the *Object Request Broker* (ORB) component that enables clients and objects to communicate in a distributed environment, and four categories of object interfaces:

- *Object Services* are interfaces for general services that are likely to be used in any program based on distributed objects

- *Common Facilities* are interfaces for horizontal end-user-oriented facilities applicable to most application domains

- *Domain Interfaces* are application domain-specific interfaces
- *Application Interfaces* are non-standardized application-specific interfaces

These interface categories are shown in Figure 4-1.

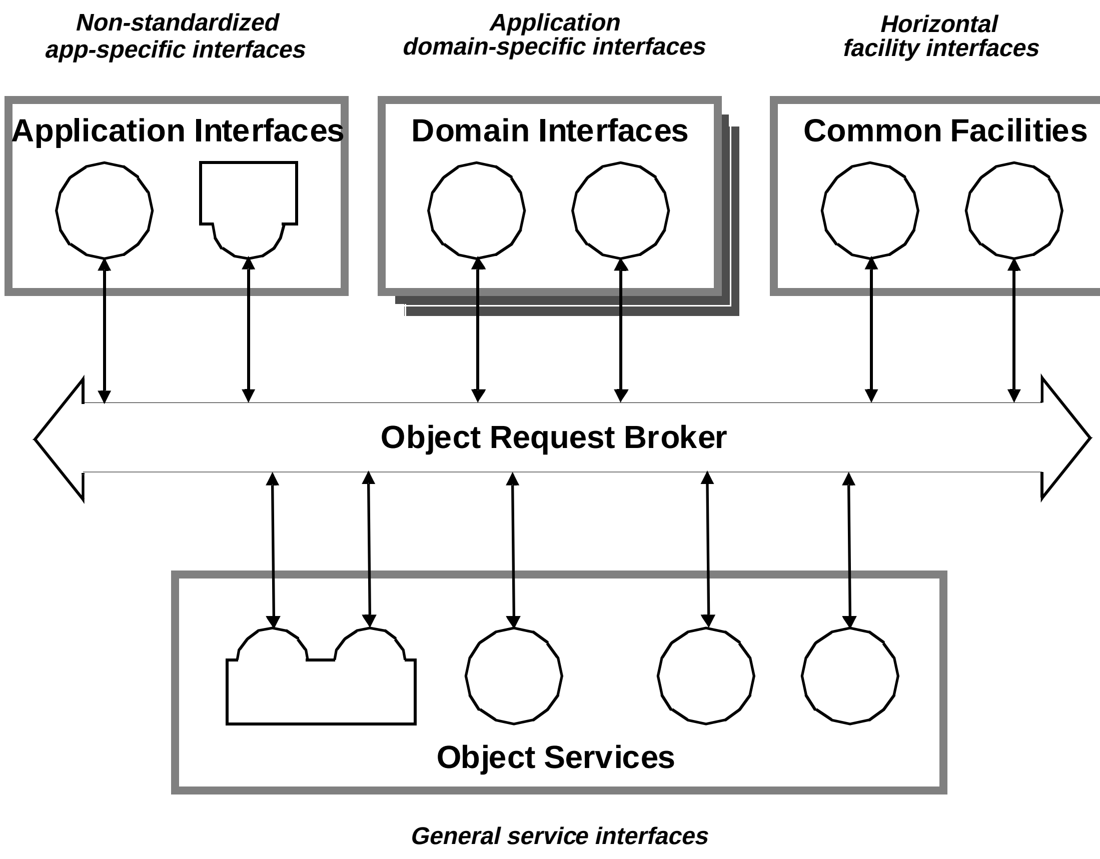
Non-standardized app-specific interfaces
Application domain-specific interfaces
Horizontal facility interfaces
Application Interfaces
Domain Interfaces
Common Facilities
Object Request Broker
Object Services
General service interfaces

Figure 4-1 OMA Reference Model: Interface Categories

A second part of the Reference Model, shown in Figure 4-2, focuses on interface *usage* and introduces the notion of domain-specific *Object Frameworks*. An Object Framework component is a collection of cooperating objects that provide an integrated solution within an application or technology domain and which is intended for customization by the developer or user. Object Frameworks are explained in more detail below.

4.1.3 Interface versus Implementation

It is important to note that applications need only support or use OMG-compliant interfaces to participate in the OMA. They need not themselves be constructed using the object-oriented paradigm. Figure 4-1 shows, in the case of Object Services, how existing non-object-oriented software can be embedded in objects (sometimes called object wrappers) that participate in the OMA.

4.1.4 Object Request Broker

The *Common Object Request Broker Architecture* defines the programming interfaces to the OMA ORB component. An ORB is the basic mechanism by which objects transparently make requests to - and receive responses from - each other on the same machine or across a network. A client need not be aware of the mechanisms used to communicate with or activate an object, how the object is implemented, nor where the object is located. The ORB thus forms the foundation for building applications constructed from distributed objects and for interoperability between applications in both homogeneous and heterogeneous environments.

The *OMG Interface Definition Language* (IDL) provides a standardized way to define the interfaces to CORBA objects. The IDL definition is the contract between the implementor of an object and the client. IDL is a strongly typed declarative language that is programming language-independent. Language mappings enable objects to be implemented and sent requests in the developer's programming language of choice in a style that is natural to that language.

4.1.5 Object Services

Object Services are general purpose services that are either fundamental for developing useful CORBA-based applications composed of distributed objects, or that provide a universal - application domain-independent - basis for application interoperability.

Object Services are the basic building blocks for distributed object applications. Compliant objects can be combined in many different ways and put to many different uses in applications. They can be used to construct higher level facilities and object frameworks that can interoperate across multiple platform environments.

Adopted OMG Object Services are collectively called CORBAservices and include Naming, Events, LifeCycle, Persistent Object, Transactions, Concurrency Control, Relationships, Externalization, Licensing, Query, Properties, Security, Time, Collections, and Trader. See “4.2 Summary of Object Services” for additional information.

4.1.6 Common Facilities

Common Facilities are interfaces for horizontal end-user-oriented facilities applicable to most application domains. Adopted OMG Common Facilities are collectively called CORBAfacilities and include an OpenDoc-based Distributed Document Facility.

A specification of a Common Facility or Object Service typically includes the set of interface definitions - expressed in OMG IDL - that objects in various roles must support in order to *provide*, *use* or *participate in* the facility or service. As with all specifications adopted by OMG, facilities and services are defined in terms of interfaces and their semantics, and not a particular implementation.

4.1.7 Domain Interfaces

Domain Interfaces are domain-specific interfaces for application domains such as Finance, Healthcare, Manufacturing, Telecom, Electronic Commerce, and Transportation. Figure 4-1, highlights the fact that Domain Interfaces will be grouped by application domain by showing a possible set of collections of Domain Interfaces.

4.1.8 Object Frameworks

Unlike the interfaces to individual parts of the OMA “plumbing” infrastructure, Object Frameworks are complete higher level components that provide functionality of direct interest to end-users in particular application or technology domains.

Object Frameworks are collections of cooperating objects categorized into *Application*, *Domain*, *Facility*, and *Service Objects*. Each object in a framework supports (for example, by virtue of interface inheritance) or makes use of (via client requests) some combination of Application, Domain, Common Facility, and Object Services *interfaces*.

A particular Object Framework may contain zero or more Application Objects, zero or more Domain Objects, zero or more Facility Objects, and zero or more Service Objects. Service Objects support Object Services (OS) interfaces; Facility Objects support interfaces that are some combination of Common Facilities (CF) interfaces and potentially inherited OS interfaces; Domain Objects support interfaces that are some combination of Domain Interfaces (DI) and, potentially, inherited CF and OS interfaces; and so on for Application Objects. Thus, higher level components and interfaces build on and reuse lower level components and interfaces.

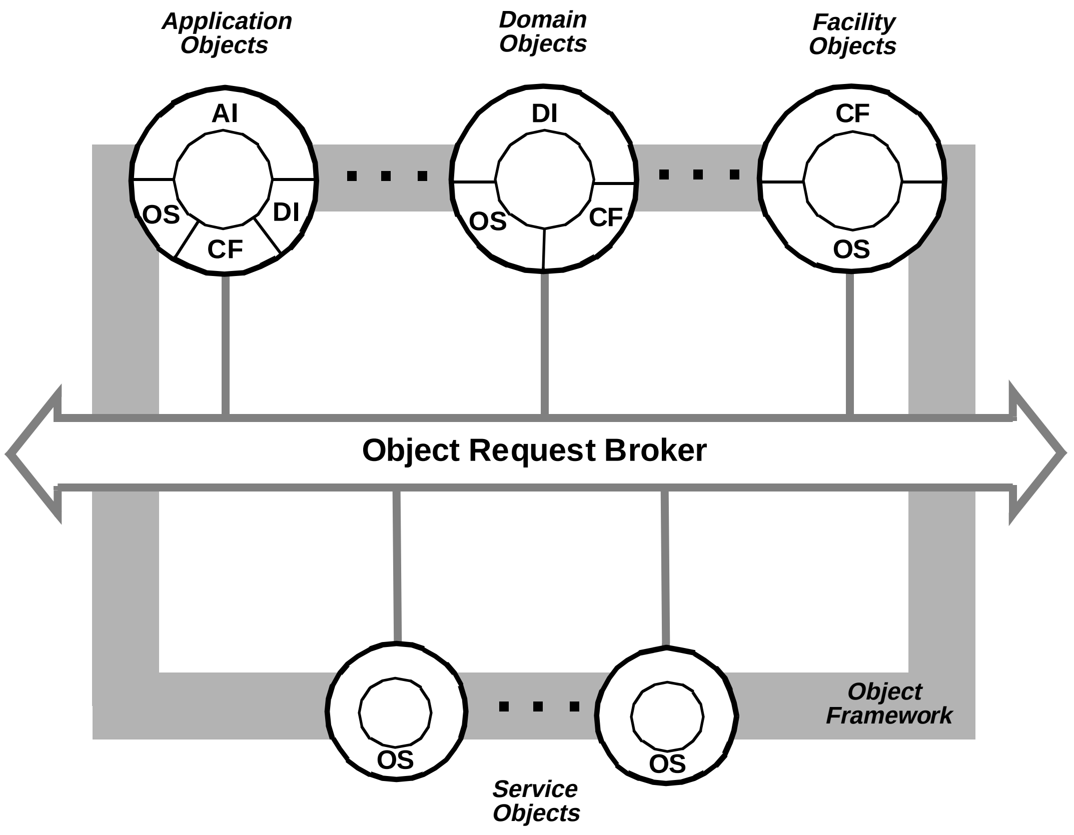

Figure 4-2 OMA Reference Model: Interface Usage

The concept of an Object Framework is illustrated in Figure 4-2. Objects are shown as an implementation “core” surrounded by a partitioned concentric shell (or “donut”) representing the interfaces that the object supports.

The picture shows the most general case where objects support all the possible interfaces for their category. In any given specific situation, degenerative cases may exist, such as Domain Objects that support only inherited Object Services interfaces (e.g. the event channel pull consumer interface) and no Common Facility interfaces, or Domain Objects that support neither Object Services or Common Facility interfaces in order to provide their functionality.

Figure 4-3 shows how objects in an Object Framework can make requests to other objects in the framework in order to provide the overall functionality of the framework. The picture shows three requests: one from an Application Object to a Service Object;

one from a Facility Object to a Service Object; and one from a Domain Object to an Application Object which could, for example, be a "call back" to a Domain Interface supported by the Application Object.

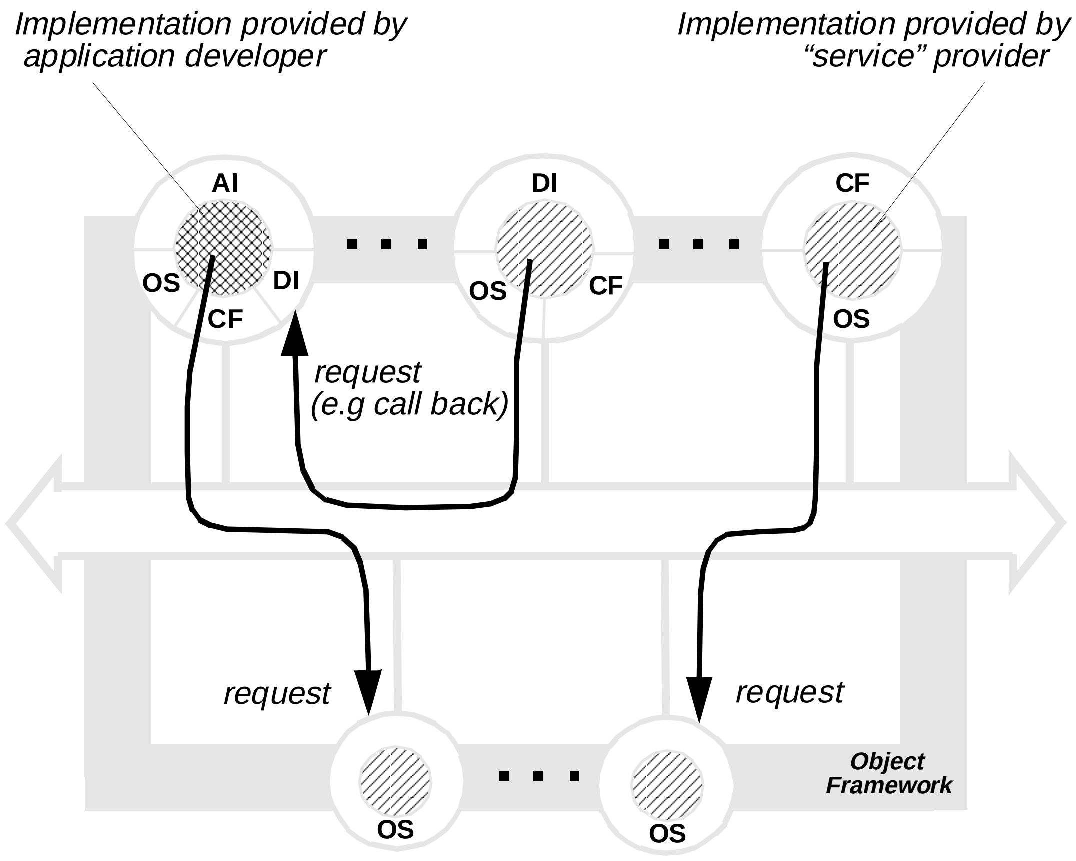
Implementation provided by application developer
Implementation provided by "service" provider
AI
OS
DI
CF
DI
OS
CF
CF
OS
request (e.g call back)
request
request
OS
OS
Object Framework

Figure 4-3 Example request flow (runtime reuse)

4.1.9 Object Framework Specifications

A specification of an Object Framework defines such things as the structure, interfaces, types, operation sequencing, and qualities of service of the objects that make up the framework. This includes requirements on implementations in order to guarantee application portability and interoperability across different platforms. Object Framework specifications may include new Domain Interfaces for particular application domains.

The application-specific part of an Application Object's interface is, by definition, not included in the specification of an Object Framework. This part is totally defined by the application developer. On the other hand, standardized interfaces that must be inherited and supported in order that the Application Object can function in the framework may form part of the framework specification.

4.2 Summary of Object Services

This section provides a brief description of each Object Service.

- The Naming Service provides the ability to bind a name to an object relative to a naming context. A naming context is an object that contains a set of name bindings in which each name is unique. To resolve a name is to determine the object associated with the name in a given context. Through the use of a very general model and in dealing with names in their structural form, Naming Service implementations can be application specific or be based on a variety of naming systems currently available on system platforms.

  Graphs of naming contexts can be supported in a distributed, federated fashion. The scalable design allows the distributed, heterogeneous implementation and administration of names and name contexts.

  Because name component attribute values are not assigned or interpreted by the Naming Service, higher levels of software are not constrained in terms of policies about the use and management of attribute values.

- The Event Service provides basic capabilities that can be configured together flexibly and powerfully. The service supports asynchronous events (decoupled event suppliers and consumers), event "fan-in," notification "fan-out,"—and through appropriate event channel implementations—reliable event delivery.

  The Event Service design is scalable and is suitable for distributed environments. There is no requirement for a centralized server or dependency on any global service. Both push and pull event delivery models are supported; that is, consumers can either request events or be notified of events.

  Suppliers can generate events without knowing the identities of the consumers. Conversely, consumers can receive events without knowing the identities of the suppliers. There can be multiple consumers and multiple suppliers of events. Because event suppliers, consumers, and channels are objects, advantage can be taken of performance optimizations provided by ORB implementations for local and remote objects. No extension is required to CORBA.

- The Life Cycle Service defines operations to copy, move, and remove graphs of related objects, while the Relationship Service allows graphs of related objects to be traversed without activating the related objects. Distributed implementations of the Relationship Service can have navigation performance and availability similar to CORBA object references: role objects can be located with their objects and need not depend on a centralized repository of relationship information. As such, navigating a relationship can be a local operation.

- The Persistent Object Service (POS) provides a set of common interfaces to the mechanisms used for retaining and managing the persistent state of objects. The object ultimately has the responsibility of managing its state, but can use or delegate to the Persistent Object Service for the actual work. A major feature of the Persistent Object Service (and the OMG architecture) is its openness. In this case, that means that there can be a variety of different clients and implementations of the Persistent Object Service, and they can work together. This is particularly important for storage, where mechanisms useful for documents may not be appropriate for employee databases, or the mechanisms appropriate for mobile computers do not apply to mainframes.

- The Transaction Service supports multiple transaction models, including the flat (mandatory in the specification) and nested (optional) models. The Transaction Service supports interoperability between different programming models. For instance, some users want to add object implementations to existing procedural applications and to augment object implementations with code that uses the procedural paradigm. To do so in a transaction environment requires the object and procedural code to share a single transaction. Network interoperability is also supported, since users need communication between different systems, including the ability to have one transaction service interoperate with a cooperating transaction service using different ORBs.

  The Transaction Service supports both implicit (system-managed transaction) propagation and explicit (application-managed) propagation. With implicit propagation, transactional behavior is not specified in the operation’s signature. With explicit propagation, applications define their own mechanisms for sharing a common transaction.

  The Transaction Service can be implemented in a TP monitor environment, so it supports the ability to execute multiple transactions concurrently, and to execute clients, servers, and transaction services in separate processes.

- The Concurrency Control Service enables multiple clients to coordinate their access to shared resources. Coordinating access to a resource means that when multiple, concurrent clients access a single resource, any conflicting actions by the clients are reconciled so that the resource remains in a consistent state.

  Concurrent use of a resource is regulated with locks. Each lock is associated with a single resource and a single client. Coordination is achieved by preventing multiple clients from simultaneously possessing locks for the same resource if the client’s activities might conflict. Hence, a client must obtain an appropriate lock before accessing a shared resource. The Concurrency Control Service defines several lock modes, which correspond to different categories of access. This variety of lock modes provides flexible conflict resolution. For example, providing different modes for reading and writing lets a resource support multiple concurrent clients on a read-only transaction. The Concurrency Control service also defines Intention Locks that support locking at multiple levels of granularity.

- The Relationship Service allows entities and relationships to be explicitly represented. Entities are represented as CORBA objects. The service defines two new kinds of objects: *relationships* and *roles*. A role represents a CORBA object in a relationship. The Relationship interface can be extended to add relationship-specific attributes and operations. In addition, relationships of arbitrary degree can be defined. Similarly, the *Role* interface can be extended to add role-specific attributes and operations. Type and cardinality constraints can be expressed and checked: exceptions are raised when the constraints are violated.

- The Externalization Service defines protocols and conventions for externalizing and internalizing objects. Externalizing an object is to record the object state in a stream of data (in memory, on a disk file, across the network, and so forth) and then be internalized into a new object in the same or a different process. The externalized object can exist for arbitrary amounts of time, be transported by means outside of the ORB, and be internalized in a different, disconnected ORB. For portability, clients can request that externalized data be stored in a file whose format is defined with the Externalization Service Specification.

  The Externalization Service is related to the Relationship Service and parallels the Life Cycle Service in defining externalization protocols for simple objects, for arbitrarily related objects, and for facilities, directory services, and file services.

- The Licensing Service provides a mechanism for producers to control the use of their intellectual property. Producers can implement the Licensing Service according to their own needs, and the needs of their customers, because the Licensing Service does not impose its own business policies or practices.

  A license in the Licensing Service has three types of attributes that allow producers to apply controls flexibly: time, value mapping, and consumer. Time allows licenses to have start/duration and expiration dates. Value mapping allows producers to implement a licensing scheme according to units, allocation (through concurrent use licensing), or consumption (for example, metering or allowance of grace periods through "overflow" licenses). Consumer attributes allow a license to be reserved or assigned for specific entities; for example, a license could be assigned to a particular machine. The Licensing Service allows producers to combine and derive from license attributes.

  The Licensing Service consists of a *LicenseServiceManager* interface and a *ProducerSpecificLicenseService* interface: these interfaces do not impose business policies upon implementors.

- The Query Service allows users and objects to invoke queries on collections of other objects. The queries are declarative statements with predicates and include the ability to specify values of attributes; to invoke arbitrary operations; and to invoke other Object Services.

  The Query Service allows indexing; maps well to the query mechanisms used in database systems and other systems that store and access large collections of objects; and is based on existing standards for query. The Query Service provides an architecture for a nested and federated service that can coordinate multiple, nested query evaluators.

- The Property Service provides the ability to dynamically associate named values with objects outside the static IDL-type system. It defines operations to create and manipulate sets of name-value or name-value-mode tuples. The names are simple OMG IDL strings. The values are OMG IDL *anys*. The use of type *any* is significant in that it allows a property service implementation to deal with any value that can be represented in the OMG-IDL-type system.

  The Property Service was designed to be a basic building block, yet robust enough to be applicable for a broad set of applications. It provides “batch” operations to deal with sets of properties as a whole. The use of “batch” operations is significant in that the systems and network management (SNMP, CMIP,...) communities have proven such a need when dealing with “attribute” manipulation in a distributed environment.

- The Security Service comprises:

  - **Identification** and **authentication** of principals (human users and objects which need to operate under their own rights) to verify they are who they claim to be.

  - **Authorization** and **access control** - deciding whether a principal can access an object, normally using the identity and/or other privilege attributes of the principal (such as role, groups, security clearance) and the control attributes of the target object (stating which principals, or principals with which attributes) can access it.

  - **Security auditing** to make users accountable for their security related actions. It is normally the human user who should be accountable. Auditing mechanisms should be able to identify the user correctly, even after a chain of calls through many objects.

  - **Security of communication** between objects, which is often over insecure lower layer communications. This requires trust to be established between the client and target, which may require **authentication of clients to targets** and **authentication of targets to clients**. It also requires **integrity protection** and (optionally) **confidentiality protection** of messages in transit between objects.

- **Non-repudiation** provides irrefutable evidence of actions such as proof of origin of data to the recipient, or proof of receipt of data to the sender to protect against subsequent attempts to falsely deny the receiving or sending of the data.

- **Administration** of security information (for example, security policy) is also needed.

- The Time Service enables the user to obtain current time together with an error estimate associated with it. It ascertains the order in which “events” occurred and computes the interval between two events.

  Time Service consists of two services, hence defines two service interfaces:

  - Time Service manages Universal Time Objects (UTOs) and Time Interval Objects (TIOs), and is represented by the *TimeService* interface.

  - Timer Event Service manages Timer Event Handler objects, and is represented by the *TimerEventService* interface.

- The Collections Service provides a uniform way to create and manipulate the most common collections generically. Collections are groups of objects which, as a group, support some operations and exhibit specific behaviors that are related to the nature of the collection rather than to the type of object they contain. Examples of collections are sets, queues, stacks, lists, and binary trees.

  For example, sets might support the following operations: insert new element, membership test, union, intersection, cardinality, equality test, emptiness test, etc. One of the defining semantics of a set is that, if an object O is a member of a set S, then inserting O into S results in the set being unchanged. This property would not hold for another collection type called a bag.

- The Trader Service provides a matchmaking service for objects. The service provider registers the availability of the service by invoking an export operation on the trader, passing as parameters information about the offered service. The export operation carries an object reference that can be used by a client to invoke operations on the advertised services, a description of the type of the offered service (i.e., the names of the operations to which it will respond, along with their parameter and result types), information on the distinguishing attributes of the offered service.

  The offer space managed by traders may be partitioned to ease administration and navigation. This information is stored persistently by the Trader. Whenever a potential client wishes to obtain a reference to a service that does a particular job, it invokes an import operation, passing as parameters a description of the service required. Given this import request, the Trader checks appropriate offers for acceptability. To be acceptable, an offer must have a type that conforms to that requested and have properties consistent with the constraints specified by an importer.

Trading service in a single trading domain may be distributed over a number of trader objects. Traders in different domains may be federated. Federation enables systems in different domains to negotiate the sharing of services without losing control of their own policies and services. A domain can thus share information with other domains with which it has been federated, and it cannot be searched for appropriate service offers.