FINNISH DATA COMMUNICATION ASSOCIATION STY

UNIVERSITY OF HELSINKI

TITLE: Federation transparency in the ODP Trading function

SOURCE: Finland

PROJECT:

STATUS: National body contribution

DATE: June 1994

NOTES: Finnish National Body contribution, to be discussed at the joint SC21/WG7 meeting in Southampton

Federation transparency in the ODP Trading function

Abstract

Trading is an essential infrastructure function specified within ODP (Open Distributed Processing) reference model. The purpose of trading is to support location transparency, i.e. hide the location of service providers, even if they are located in separate domains. Cooperation between trading service providers (traders) is called federation. In order to achieve federation transparent service, the traders have to mediate service type differences, negotiate policy differences and tolerate technology differences between domains.

In this paper we suggest an alternative way to model trading by using ODP concepts of contracts and templates. We clarify the trading functions relationship to the overall ODP infrastructure, in respect to interceptors and type management. We also show a computational viewpoint model of trading community, based on the general concepts of ODP object. This includes discussion on policies governing the behavior of trader object. Finally, we describe a simplified model for achieving federation transparent trading service.

1. INTRODUCTION

ODP (Open Distributed Processing) is a framework for systems that support heterogeneous distributed processing [ODP885, Ray93]. This framework has been under development since the late 1980's within ISO and ITU as a joint effort. The standardization effort produces a reference model that organizes the discussion about distributed systems. It allows to realize the benefits of distributing information processing services. It also supports the design process of new standards and real systems by dividing the complex tasks into viewpoints and into standardized service functions within the viewpoints.

The goal of the ODP framework is to provide a distribution transparent environment for software components to interwork, despite they are supported by heterogeneous platforms. Software components (and hardware, as well) are modeled as objects. Objects request services (defined actions) from their environment (other objects). Distribution transparency requires transparency of failures, locations and implementation technologies of service providers.

Within this framework trading is a essential infrastructure function. It can be used to achieve location transparency. Traders provide to their clients a network wide selection among interchangeable service provider objects. The selection is based on object properties. Static properties describe the abilities of an object, dynamic properties the status of an object. The trader selects a server object for a given task by considering the dynamic status of service providers, service requestors and the underlying system infrastructure.

As the global network is composed of several autonomous, heterogeneous subsystems, an implementation of trading functionality can not be centralized. Each autonomous subsystem should have a trader administering services. To achieve location transparency throughout the combined network, traders have to cooperate. This cooperation is called federation. An important goal is that the objects using trading function, should not be forced to see administrative boundaries between subsystems. Instead, trading should be a federation transparent function. This property gives tools to the ODP infrastructure to provide distribution transparency of all services.

In this paper, we start by clarifying the relationship of trading function to the overall ODP infrastructure. Trading function should hide from a service user the locations of service providers in different domains, which can be administrative, technical or political. However, traders must utilize other ODP infrastructure functions to achieve this transparency. Also, we suggest an alternative way to model trading by using ODP concepts of contracts and templates.

Next, we show a computational viewpoint model of trading community, based on the general concept of ODP object. We compose the trader object from a private data storage, a set of properties and a set of trading operations. We elaborate on the trader properties needed in an isolated subsystem.

Then, we augment the model to cover multiple subsystems. Federal cooperation between trader objects is hidden from the utilizers of trading function. A mechanism for the traders to retrieve information about other traders is suggested and the federation model is compared with the earlier suggestions in the trading function standard draft.

Basic goals of our model is to minimize information stored in each trader and to minimize the differences between local and federated operations within the trading community.

2. TRADING COMMUNITIES

A basic modeling concept in ODP reference model is an object. Objects interact with each other via interfaces. An object can initiate operations at another object by binding itself into an interface at the target object. Two of the key problems are, how the objects know where the possible interfaces are located, and how to choose between them. Objects willing to interact may be located at distinct nodes, they may use different types of operations and they may be introduced to the global system independently.

A supporting service is needed for objects to locate each other. The objects need a logically centralized location at which they may request information about available interfaces. This service facilitates late binding of objects. Late binding means that an initiating object may invoke an operation at another object created later than the initiating object itself. Thus, the binding should not be done statically at the time of object creation. Late binding has the benefit of allowing flexible system evolution. It also supports provision of availability and fault tolerance within the system, as objects may dynamically change the service provider they choose to interact with.

This supporting service, called trading function, is a fundamental infrastructure function introduced within the ODP reference model. The purpose of trading is to achieve location transparency. Traders provide to their clients a network wide selection among interchangeable service providers, based on properties and dynamic state of service providers, service requestors and the underlying system infrastructure. Traders differ from name or directory servers by offering a nondeterministic translation from an attribute based description of a required behavior to a location of an object interface. So many dynamically chancing factors affect the decisions of a trader that it can produce different results depending on requestor or time.

A trading community is a collection of objects, having the roles of clients, servers and a trader. The community is shown in Figure 1. Server objects may, by an export operation, inform the trader about their interfaces and properties, such as the type of service they provide and the behavior they promise to follow. The properties may also be dynamic like, for example, the status of the server.

The trader stores this information as a service offer. Clients may request sets of service offers to be shown. A client request operation is called import, and it consists of predicates on service type and service provider properties. The reason for importing offers is naturally the need to interact with the server object, and the client binds itself to the server without further intervention of the trader.

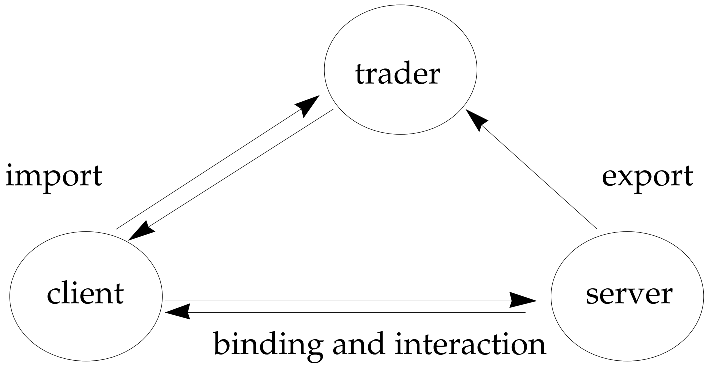

Figure 1. Trading community.

Service offers and import requests can both be modeled as contract templates in ODP terminology [ISO10746–2]. An ODP contract is a description of the objects behavior during an interaction between objects. Thus, for an interaction, each participating object must possess a corresponding contract. A contract template is a collection of details needed in the creation of a contract. Both the service offer and the import request specify predicates on which a contract can be created. The matching pair of contract templates contains enough information to actually create the contracts by combining the offer and the request. The task of the trader is the matching of templates. The client has to initiate the establishing behavior of the binding between the client and the server. The client creates a contract and informs the server that it requires the server to create a corresponding contract and to follow the behavior stated in that contract.

As the global network is composed of several autonomous, heterogeneous subsystems, an implementation of trading functionality can not be centralized. Each autonomous subsystem should have a trader administering its services. To achieve location transparency in the combined network, traders have to cooperate. This cooperation is called federation. An important goal for the trading function is, that the objects using trading function should not be forced to take into account the administrative boundaries between subsystems. Instead, the trading should happen in a federation transparent way.

As cooperation between traders is designed, some requirements derived from the goals of the ODP framework must be considered. First, the traders must be autonomous. The administrator of a trader may independently select a reasonable set of service types known in the community and a set of policies which affect storaging and searching of offers, security and participation in federations. Second, the traders may be heterogeneous. The engineering of trading service may be different at separate subsystems and the technology with which the trader has been implemented may differ. Also the requirements set by trading community in regard with trading service quality, e.g. cost, speed and accuracy of the relayed offers, may be different. Policies regulating the behavior of the trader may also differ from a subsystem to another. Policies cover decisions like: Who may export services, who is allowed to import certain services, and what happens if the trader policy and the client request contradict. Third, there may be also technology boundaries to cross over even in the communication between traders. ODP reference model offers interceptors to mediate between objects using different communication mechanisms or different type specifications. These interceptors are introduced before the traders communicate. Figure 2 represents two trading communities in federation.

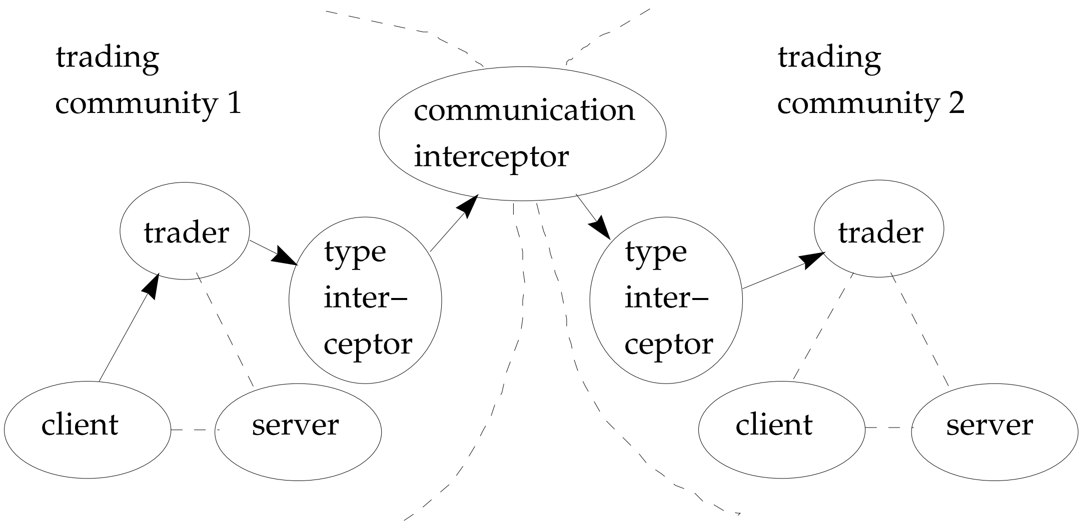

Figure 2. Federated trading.

3. PROVIDING TRADING SERVICES WITHIN A SINGLE SUBSYSTEM

Let us first consider a single trader that works alone to serve some clients and servers. The group of objects playing roles of traders, clients and servers, is called a trading community. A computational viewpoint description of the trading community and its object templates is shown in Figure 3.

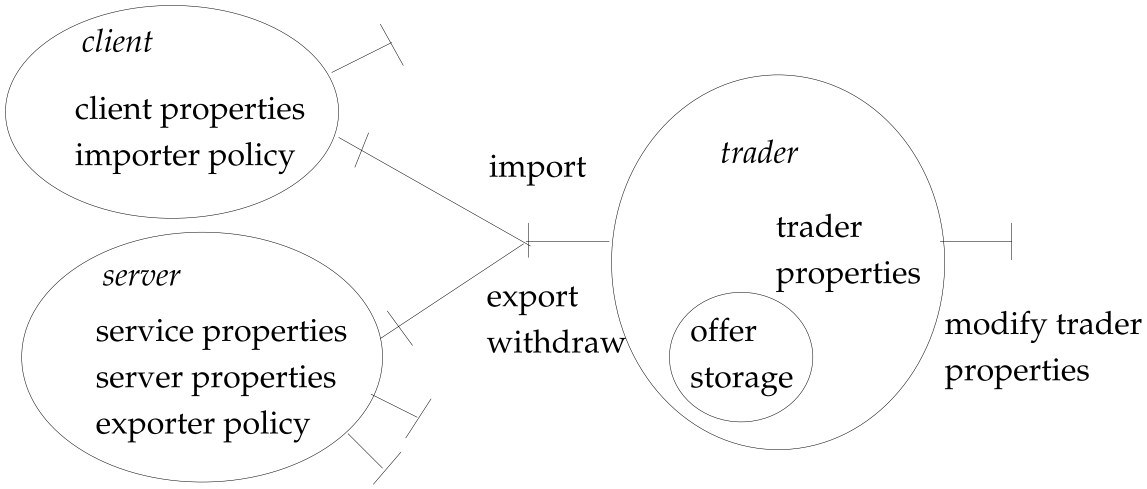

Figure 3. Single trading community.

A trader template (model) consists of the following components:
– templates for operations import, export and withdraw
– storage for offers
– set of trader policies guiding the behavior of the trader, arranged as trader properties and
– templates for trader management operations.

Trader properties needed in a single trading community are:
– service offer acceptance policy
– storage policy
– import request acceptance policy
– import action policy, consisting of
    – trader assignment role
    – local search policy and
    – resource consumption policy
– trader management security policy.

Some of these properties are private to the trader, i.e. the trader does not expose them to other objects, but uses them as a behavioral guideline. Some other properties are public and can be revealed to objects requesting trader properties.

In the ODP framework objects need to have a security policy. The trader security policy is a combination of import and export acceptance policies and trader management security policy.

The trader community also includes clients and servers. As being a client or a server are only roles, each object template includes the following components:
– exporter policy
– importer policy
– management security policy
– public properties

- private properties
- mission specific operations
- management operations.

The trader must offer operations for servers to export their service offers. The export operation behavior within the trader is regulated by service offer acceptance policy and storage policy. The acceptance policy restricts untrusted service providers from joining the trading community. To implement its purpose, the trader must have a storage for offers. How this storage is organized is an administrative decision, that can also be taken at trader implementation time. The logical structure of the storage is expressed as a trader property called storage policy. Storage policy can arrange offers into groups for example by service type or other predicate, or require hierarchical storage with contexts representing organizational boundaries. By storage policy, also the backup policies needed for service offers, can be dictated.

The servers must also have an operation on the trader to withdraw their offers. The decision to withdraw is done by the exporting server alone. It is possible, that the service offer is known to some clients in the community, when the exporter decides to withdraw it. The consequences of offers becoming stale can be reduced by the export acceptance policy and importer policies used. The service offers may carry a property value implying the probable time to live for the exported offer. Also the service request may carry a predicate requiring an adequate long availability time for the offer to be selected. If it is desired that the trader knows which clients hold copies of a service offer, other agents within the trading community can provide the necessary information. These agents can modify the properties of a service offer or provide dynamic values for the offer properties.

Servers are also able to restrict the set of clients allowed to import the exported offer. The exporter has an exporter policy associated with it. Within the service offer, this policy can be stored as a set of predicates considered by the trader when selecting suitable offers for importing clients. The exporter may for example restrict a service to be used within one organization only, although the federated trading community may span several organizations.

For the objects in client role, the trader offers import operations that may vary in the extent of information obtained. Import operation may retrieve only the identifiers needed for binding or also the property values of the offers. The import operation requires that the trader accepts from the importer a contract template (service request) and then searches its service offer storage for a set of contract templates (service offers) that can be joined with the service request contract template to form a contract. A service request includes predicates, which must be fulfilled by the suggested service offers. The set of acceptable service offers must be further checked, so that also the service exporter can accept the service import to take place.

It should be kept in mind, that also traders can act in a server role with respect to the client. Traders offer a service that allows the client to request information about servers in the community. The server group should also include the serving trader itself, and thus, the importer may request the properties of the trader as well as properties of other service providers.

Strongly related to storage structure is the local search policy, that can be decided by the administrator of the trader, within the selection possibilities that are left by the trader software implementation. The local search policy regulates the search order within the local offer storage and may for optimization reasons restrict the search to certain subcontexts of the storage. The search is also restricted by the maximum resource consumption allowed for an individual import request. The local search policy and the resource consumption policy are main components of the import action policy. Import action policy may also state the assignment role of the trader in the community. Trader assignment role is for example dispatcher, trustee, consultant or coordinator [WoTs91], depending on who takes the decision if policies contradict or the service request predicates do not produce a selection narrow enough.

The matching process between service offers and service requests is guided by the importer policy, the predicates given by the importer in the service request and the import action policy of the trader. Both the importer and the trader may set predicates concerning resources consumed in the search, for example in terms of time or costs. Trader and importer policies may contradict. Different assignment roles of traders may require different decisions – some let the trader policy to overrun, some others may totally lack the trader policy. Unwanted importers can be ruled out by the import acceptance policy.

We have listed here several policies, that can be dictated by the trader administrator. The administrator must have operations for managing the trader and its policies. Some of the policies, as storage policy, are implicit and dictated by the implementation. However, even the inherent policies represent important properties of the trader. For the administration purposes the trader must offer management operations. As all the policies are represented as private or public properties of the trader, only one operation is needed: modify trader property. By this operation the administrator of the trader is able to guide the behavior of the trader in different operations and also restrict the usage of those operations by the other members of the trading community.

4. PROVIDING TRADING SERVICES IN A COALESCED NETWORK

Let us then consider new requirements caused by the cooperation between traders. Traders must again offer the same operations for the members of their local trading communities and administrators. The only change in the trader behavior is that the import operations may now expand several trading communities, and therefore, the trader must obtain information about other trading servers.

If a trader works alone when serving an import request, it needs only to search its own storage of service offers. If it has decided to use federation, it relays the import request to other traders. The relaying trader acts in a client role in regard to the other traders, in behalf of its own client. When the import operations terminate, the sets of offers are passed to the trader in client role. The trader may temporarily store the offers for further use, but it is not obliged to do so.

Some difficulties arise, because the trading communities are autonomous and are allowed to independently decide which clients are served, and furthermore, may independently decide what service types are supported. The first problem can be solved by letting the administrator of a remote trader to allow the local trader to act as a client, within the normal trader security policy. It is more difficult to find the traders, from which to import. This is discussed in chapter 4. Differences in supported service types need careful consideration.

Traders need a possibility to test the compatibility of the service types they know before comparisons between service requests and service offers originating from separate subsystems can be done. It is not reasonable to compare the whole type space every time an federated import is attempted. Neither it is reasonable to compare whole type spaces with every possible federation companion in the network beforehand, especially if the network is large or changing over time.

The current Trading function standard draft introduces a federation establishment mechanism suitable for small or relatively static configurations of traders. The federation model of [BeRa91, ODP807, issue list of ODP897] uses two protocols: one between the client and its local trader, and another between two traders. The protocol between two traders includes active exporting of trading service offers to all potential cooperation parties, negotiation of contracts and storage of the resulting federation export and import contracts into the interacting traders. Thus, even traders that probably don't use the possibility of federation, establish these federation contracts. Knowledge about remote traders is stored into the trader as a link domain consisting of trading service contracts, analogically to service offer domain consisting of service contract templates. At the import operations of clients, the local trader acts in behalf of the client and initiates an import operation at the federated trader. The federated trader offers special operations for importing traders which correspond to the import operations for local clients. The duplication of import operations is used because service types must be intercepted when the import activity crosses a subsystem boundary.

In a small network, the trader can store knowledge about cooperating traders and compare beforehand the type spaces in order to decide what service types are compatible between traders, and what interceptors are possibly needed to create mappings for technical differences, for example different names of the same service type. In order to store full knowledge about the possible services available from other traders "globally" the trader must behave in the following way:

Let us consider federation among N traders, each having M known service types in common. Each liaison between two traders require storaging of a tuple { target trader identifier, service type available, interception knowledge, restrictions for importing }. Each liaison requires a broadcast in the global network to reach possible traders, followed by a bilateral negotiation to form a contract about all M service types and interceptors needed at the federated import operations. This means N broadcasts and N x N x M negotiations in total. After negotiations each trader has a storage of N x M tuples, regardless whether the liaisons are going to be used or not. The negotiations must be repeated from time to time to refresh the knowledge.

We suggest a simplified mechanism. In a large network of traders, the comparison of service type spaces may be more efficiently done by demand. When a trader wishes to import offers of a certain service type, it requests possible federation targets from a repository of trading servers (see chapter 4). It also requests descriptions of the service types those traders support. From these replies the trader may decide, which other traders offer compatible service types and what interceptors are needed. It may store this information for further use.

To make the requests for service types possible, the trader properties must be supplemented by a service type federation policy. This policy consists of a reference to the locally used type management system and an optional reference list to known federation targets, each with an incremental list of intercepted service types. Service type federation policy is included in the import action policy. In other words, the trader is aware of the service types available locally and also aware, how to intercept service type information when importing from another trader. The trader collects the interception information incrementally.

Let us again consider the federation among N traders. Each trader now imports offers from K other traders. When a client requests a service, the trader first imports a set of trader offers from a known source. Next the trader negotiates directly with the selected K traders and stores a maximum of K x M tuples. No broadcasts are needed, and a maximum of N x K x M negotiations may take place at independent times. In a typical case, N is large, but K and M are small.

By using federation, a client may now import from the local trader a set of service offers originating from separate subsystems. As the trader behavior in the import operation has now been changed, we must consider also the corresponding predicates that the client may set for the operation. The client may guide the trader in federation by either allowing or forbidding offers originating from some federating traders, or by allowing a certain amount of hops from trader to trader. Some restrictions, such as cost and time, also apply when the trader passes the request to other traders. The trader must then split the maximum allowed resource consumption between traders involved.

5. TRADERS SEARCHING FOR COOPERATORS

One of the design problems in trading function is, how the traders learn about each other. A trader is created within a single subsystem and initially it has no information about other traders. The administrator may define the trader security policy so, that the trader may import offers from some neighboring traders. However, the amount of traders may grow, and it would be convenient for the trader to automatically find new traders from the network. Traders are by nature servers themselves. Traders may import service offers also for their own needs, not only for the needs of their clients. It is a security policy decision if the trader is allowed to import trading service offers of new and possibly untrustworthy subsystems from the trusted neighbor trader.

Administrators of traders need to export new traders once to a few other traders. As traders from time to time import trading service offers from each other, they gradually learn to know newly created traders and can start to federate with them. Each trader has a partial view of the global set of traders, and no central repository exists. Thus scaling problems are avoided. The method resembles the probabilistic sparse diffusion multicast presented in [Sch91]. The spreading of a trading service offer is restricted by the same mechanisms as normal services. This exporting system contains no active negotiation of federation contracts.

Let us consider an example of creating a new trader at the Department of Computer Science (CS) at University of Helsinki (UH). The administrator creates first a type management system that contains for example printers, documents and editors. Next, the administrator selects a location and instantiates the trader. As an administrative operation the trader gets the following policies:

- storage policy: storage is structured by service types, no backups – except on trading service offers
- service offer acceptance policy: only CS department may export services, all trading service offers are accepted by first negotiating with the administrator
- import request acceptance policy: only CS department may import all services, but named individuals from other organizations may import document services and all trading service offers are revealed to others if not denied by the exported offer itself, local importers are allowed to import documents and printers from remote trading communities
- trader management security policy: only system managers are allowed to modify policies
- trader assignment role: trader policies override importer policies
- resource consumption policy: a single import is restricted to 1 minute.

In order to facilitate federation, the administrator export two offers to the new trader: one for trading service at Computing Centre of UH and another for trading service at Technical University of Helsinki. As the foreseen importers are elsewhere, the administrator next exports the new trader to the traders of Tampere University and Computing Centre of UH. Exporter policy expresses that these trading service offers may be spread to all requesting objects.

6. SUMMARY

In this paper we have proposed a simplified model of trading function that allows cooperation between autonomous trader objects. Cooperation provides objects, that request some actions from their environment, a federation transparent view to a set of service providers in a coalesced network. Basic goals of the model have been to minimize information stored in each trader for federation activities, to minimize communication needed in establishing federation liaisons and to use same policy regulated import operations for local and federated trading.

A trading community consists of a group of clients, a trader and a group of servers. Servers announce their abilities to produce a certain service by exporting a service offer to the trader. Clients may request the trader to search a group of service offers, constraining the suitability by giving some matching rules expressed using service and service provider properties.

A trader may extend the search area of service offers from the locally know trading community to another community, if it has knowledge about at least one other trader that is willing to cooperate. A remote trader may allow the searching trader to request information about offers using the same request operation as the client in the local trading community. When federation is in use, the search progresses in stages. First, the client requests the local trader for a set of service offers. Due to the request, the trader makes a corresponding request for service offers to another trader, acting as a normal client, in behalf of somebody else. The trader may request several other traders either parallelly or serially. When these requests have been served the trader collects all service offers, possibly makes some selection among the offers, and then sends the information to the client as the response to the original request. The decision on what other traders to use in search for suitable offers is based on the import action policy of the trader. The service type federation policy includes information about suitable cooperating traders and possibly also hints for deciding what requests are worth passing to each trader.

An important aspect in federation is, how the traders learn about each other. Spreading the locations and properties of traders can be done by the traders themselves, but there need to be some inherent knowledge to start with. The administrator may present some neighboring traders to the local trader, and the trader itself is exported to some other traders. Each trader becomes then aware of a partial view of the global set of traders and their locations in the reachable network. Using this knowledge, the trader can find out candidates for federation and request directly the properties of remote traders. If the remote trader is willing to cooperate, the trader (or the administrator) may use its response for adding new items into the import action policy.

REFERENCES

BeRa91 Bearman, M., Raymond, K., Federating Traders: an ODP Adventure. International IFIP Workshop on Open Distributed Processing. Berlin, Germany, 1991.

ODP807 ISO/IEC JTC1/SC21/WG7 N807. Working Document on Topic 9.1 – ODP Trader. June 1993.

ODP885 ISO/IEC JTC1/SC21/WG7 N885. Working Document output from the interim meeting of Turin, Italy. November 1993.

ODP897 ISO/IEC JTC1/SC21/WG7 N897. Information technology – Open Distributed Processing – ODP Trading Function. WG 7 working document. November 1993.

ISO10746–2 Information Technology – Open Systems Interconnection, Data Management and Open Distributed Processing – Basic Reference Model of Open Distributed Processing – Part 2: Descriptive Model. November 1993. Committee Draft.

Ray93 Raymond, K. A., Reference Model of Open Distributed Processing: A Tutorial. International IFIP Conference on Open Distributed Processing. Berlin, Germany, 1993.

Sch91 Schwartz, M., F., Resource Discovery in the Global Internet. Department of Computer Science, University of Colorado, Boulder. November 1991. CU–CS–555–91.

WoTs91 Wolisz, A., Tschammer, V., Dynamic Binding of Service Users and Providers in an Open Services Environment. Computer Networks, 1991.