5

# An Enterprise Trader Model for DCOM

*G. Outhred and J. Potter*
*Microsoft Research Institute*
*School of MPCE*
*Macquarie University, NSW 2109, Australia*
*{gouthred, potter}@mri.mq.edu.au*

**Abstract**

This paper proposes a trader model based on both the RM-ODP trader model and Microsoft's DCOM model. The trader is designed to integrate into a structured enterprise environment, providing access to and support for management of distributed services. The proposed model differs markedly from current trader models, developing a simplified service representation combined with constraints on trader interconnection topologies. The combination of these new features results in a model which is designed to be useable, efficient and scaleable avoiding the complexity associated with previous models.

**Keywords**

Trader Services, Distributed Systems, System Management

## 1 INTRODUCTION

A trading service in a distributed system provides a mechanism by which service instances can be advertised to clients. An example of a service instance under Microsoft's Component Object Model (COM) [1] is an object providing one or more public interfaces. Service instances may be transitory, existing only because of current demand, running on nodes that are not otherwise occupied or alternatively well known services running on dedicated machines located at strategic sites throughout the system.

From the client's perspective, a trader provides a flexible resource discovery service. When given a search description for a service, the trader attempts to find a matching service instance. A simple query to a trader may resolve to the first found instance of a particular service. A more complex query may request a list of services satisfying additional constraints particular to the service. For example, a client may requests a list of news servers that store a particular news group.

Microsoft's COM model describes a component system in which classes, applications and interfaces are identified using 128bit globally unique identifiers (GUID's). All classes must support an interface called IUnknown through which clients can query for other interfaces supported by an object. Distributed COM (DCOM) allows connection to or activation of remote COM objects. Knowledge of remote services is either captured in the system registry (a database local to each machine) or provided by the client. There is currently no support for services to advertise themselves to clients on remote machines.

J. Rolia et al. (eds.), *Open Distributed Processing and Distributed Platforms*

The trading service proposed in this paper extends the DCOM model to allow connection to remote services without knowledge of their location. The trader model developed to support this differs substantially from existing trader models [2-8] as it includes constraints on the interconnection of traders and simplifies the service representation within the trader. At the same time there has been a change in emphasis between the proposed trader model and previous trader models. The model proposed in this paper has been developed with the explicit intention of distributing the trading service within an organisation or enterprise. Previous models have been designed to support interconnection between autonomous traders called *Interworking* but have generally assumed that there will be little distribution within an organisation due to the overheads involved in inter-trader communication [4].

The application of the trader model to DCOM has two key benefits. First it leverages off the COM identification scheme, allowing the service descriptions to be kept simple. Second, it provides a higher level of service than is available using raw DCOM.

The next section of the paper presents the Enterprise Trader Model, detailing the environment in which we expect the trader to operate and the data structures and interfaces defined in the model. This is followed by a comparison with other models. Section four presents a prototype implementation developed using Microsoft's DCOM model and the final section lists possible extensions to the Enterprise Trader Model.

## 2 THE ENTERPRISE TRADER MODEL

The Enterprise Trader Model has been developed with the aim of integrating the trader service into a distributed system within an enterprise environment. It is assumed that in this environment multiple users are running multiple applications and that the users of these applications are organised into groups corresponding to the organisational structure. The topology connecting the machines belonging to the users is likely to reflect both the geography and to some extent the structure of organisation. Concerns about security are likely to prevent access to services running in the domain of other groups with the exception of public services exported from groups and common services provided by support units within the organisation.

In such an environment, available services will be divided according to the various structures within the organisation. For example the customer service group would require a different set of services to those used by the development groups. Both groups though, may share a common set of services such as mail and news servers. In order to support such information structures, the trader service must provide support for partitioning of service information and access control to such information.

In order to scale and meet performance and reliability constraints, the trading services will need to be distributed. The performance of a centralised service would be constrained by both network latency and reliability. We propose therefore that the structure of a distributed trader in such an environment should reflect both the network topology and the structure of the organisation. In this way the trader can exploit low latency network connections and the information partitioning associated with the organisational structure.

In the following model the trader mechanism has been developed in conjunction with constraints on trader interconnection policies. Previous trader models have only addressed independent traders interconnecting in arbitrary topologies. In these models, interaction between traders is dependent on the intersection of rules internal to each trader. The advantage of constraining the interconnection policy has been that it has allowed the design of the internal trader mechanism to be simplified with the result that it should be both more efficient and more manageable than previous traders. This has been at the expense of some flexibility but we believe that the loss in flexibility is outweighed by the gain in usability of the trader service.

The model description follows. The first section describes the information structures required to implement a trader, the second describes the interfaces and their underlying functionality, and section 2.3 describes the proposed constraints on trader network topologies.

## 2.1 Trader information structures

The core information required to implement a trader is captured in four tables. The local service table details services owned by the trader (those that have been published directly to the trader from a client). This table contains a description of the service, a mechanism for contacting the service and the identity of the service owner. The remote service table contains service descriptions for services propagated from other traders. The link table details the links maintained with other traders and the cache table stores the results from successful queries through this trader.

## 2.2 Trader interfaces

The functionality of the trader has been divided into the following interfaces: *ITraderPublish*, *ITraderSearch*, *ITraderLink* and *ITraderManage*. In the following sections, the key operations are described.

### *ITraderPublish*

```
Publish(ServiceDesc, ServiceID*)
```

To identify a service within COM, a GUID called a CLSID is used to return an instance of the required component. To allow users to differentiate between instances of a particular service, a display name is also published with the service. A service description combines this information together with information specifying where to find the service, who owns the service and the publication domain to which the service should be published.

The publication domain is the set of users to which the service is exposed. This is controlled by the client who can choose the traders to which the service is propagated. The ability to obtain knowledge of the users to which this will expose the service results from the constraints placed on trader interconnections to be described in section 2.3.

When a service description is passed to a trader, that trader is designated the owner-trader for the service. A new GUID called the ServiceID is generated and returned to the caller. When a service description is propagated to another trader, the ServiceID is attached to the service description and the service owner field is changed to identify the owner-trader.

```
Revoke(ServiceID)
```

To revoke a published service, the service owner uses the ServiceID previously obtained when the service was published. The revoke operation removes service entries from all traders to which the service description was originally published. If the service is subsequently republished, a new ServiceID is used. This prevents access to the service by clients using cached service entries who may not be part of the new publication domain.

### *ITraderSearch*

To search for a service a client provides a search description containing a CLSID and optionally, a name for the service. The client may specify the traders to which the search is propagated otherwise this is determined by the trader receiving the search request. For a search requiring tighter resolution of matching services, the client may provide a script to be run against the service interfaces.

```
Search(SearchDesc,IUnknown*)
Search(SearchDesc,SearchLoc,ServiceList*)
Search(SearchDesc,SearchLoc,SearchScript,ServiceList*)
Search(ServiceID,IUnknown*)
```

The *ITraderSearch* interface provides clients with access to the trader search mechanism. The search operation is overloaded with several levels of complexity. The basic search requires a simple description and if successful returns an instance of the required service. A more complex search specifies the range of traders to search and returns a list of matching services. The most flexible search uses a script that acts on the interfaces of the services found. This script indicates to the search engine whether the service is appropriate and may give the service a ranking. The list of appropriate services is returned to the user, ordered according to the ranking provided by the script. The final search method allows a service to be returned using a ServiceID. This provides the ability to reconnect to a particular service in a location independent manner.

On receiving a search request, a trader checks its local database for matching services and then searches its database of remote services published to it by other traders. At the same time it searches a cache of results from recent search requests serviced by this trader. If none of these searches are successful then the search description may be propagated to other traders.

The choice of trader to which searches will be propagated is closely related to the allowed trader topologies and will be discussed in section 2.3.

### *ITraderLink*

The ITraderLink interface encapsulates the operations supporting communication between traders.

```
Search(InternalSearchDesc)
SearchReturn(InternalSearchRet)
```

Search and SearchReturn are the methods that allow traders to propagate search requests to other traders. The return from the search is asynchronous and is identified using a GUID called a SearchID, generated by the trader through which the search was initiated.

```
ServicePropogate(InternalServiceDesc)
ServiceRevoke(ServiceID)
```

ServicePropogate allows traders to propagate services to other traders. Services are propagated to the traders specified in the InternalServiceDescription, a modified version of the original service description supplied by the client. A service revoke is propagated to all traders who originally stored the service description. When a service is revoked, cache entries will fail as the ServiceID is no longer valid.

```
Resolve(ServiceID,ReturnFlag,IUnknown*)
```

Resolve is used to check with the owner trader whether a service entry is still valid. If the ReturnFlag is set, resolve also returns an interface pointer to the service instance.

*ITraderManage*
The trader management interface provides functions to create, destroy and manipulate links between traders, control access to traders and manage the databases stored in a trader.

## 2.3 Trader interconnection topologies

The following interconnection model has been developed to complement the trader mechanism described in the previous sections. The motivation behind the introduction of the model was to simplify the design of the trader mechanism, primarily in specifying search and publish operations, to improve the efficiency of the trader in matching requests to services by finding physically local or appropriate group services first and to provide the client and system administrator with a clear insight into the behaviour and structure of the trading system.

The proposed trader topology constrains trader interconnections to tree structured hierarchies. Trader nodes that are leaves on one hierarchy may act as leaves on other hierarchies. Traders at nodes further up the tree may have multiple children but only one parent. In the hierarchy information is propagated from the leaf nodes towards the root of the hierarchy. Similarly, searches propagate from the leaf nodes to the root of the hierarchy. An example trader network is shown in Figure 1.

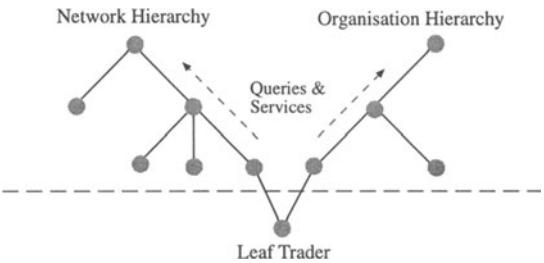


**Figure 1** An example trader topology.

One advantage gained from such a topology is that a client can easily determine to whom they are making published services available. This is because a client can only find services that exist on the path from the leaf node on which they are connected to the root. Services not on this path cannot be seen.

Search paths are well defined, there are no loops and the search is guaranteed to terminate. If the hierarchy being searched reflects the network structure of the organisation then a search is guaranteed to find the closest available service in terms of network delay. If the hierarchy reflects the logical organisation of the company, then the first services the search will find will be services local to the group in which the user belongs. In order to support both the above cases as well as other specialised hierarchies, each client trader may connect to multiple hierarchies.

Since search paths are well defined and the client passes all queries through one trader, the leaf trader is able to speed up searching by caching the results of past searches. Similarly all traders at nodes in the network speed up searches by caching services that are available at higher nodes on the hierarchy.

To connect to these hierarchies it is assumed that every user will run a trader as part of the operating system services on their node. Their personal trader will be responsible for connection to the appropriate hierarchies for that user. For example, a salesperson may automatically connect to the sales node in the organisational hierarchy providing them with

access to the sales databases, and connect to their local node in the network hierarchy to provide them with access to their local printers and mail server.

A disadvantage of constraining the topology is that the expression of differing information structures is limited. The particular constraint of the topology to a set of hierarchies has disadvantages in that all unsuccessful searches reach the root before failing. This could result in a heavy loading of this node. At the same time it is easy to predict and manage the loading within the system by placing the traders serving the root node on dedicated hardware or distributing the load across a number of replicated servers.

A similar search structure has been used as part of the scaleable object location service model proposed in the Globe Project [9]. In this model all services propagate to the top of the tree, and the structure of the tree is transparent to the client. This does not allow clients to restrict the domain to which their services are available and potentially places a large load on the root node. The model also only allows for a single search tree, requiring the service space to be broken down in ways that may not be optimal for the organisation.

## 3 COMPARISONS WITH OTHER MODELS

There are a number of proposed trader models. The basis for most current work is the trader model accompanying the Reference Model for Open Distributed Processing (RM-ODP) [2], a standard developed by ISO/IEC. The CORBA Trading Object Service [3] is based on the RM-ODP model and shares most of its design characteristics.

The RM-ODP model differs notably from our Enterprise Trader Model in that services descriptions are instances of a predetermined service type. The service type is a template that describes a set of properties that the user supplies when exporting a service. Different service types exist for all services published in the system. This has the advantage that more information is available on particular services instance in the system but requires that a set of service types must be developed and propagated to all traders in the system. If compatibility is required in other systems then this type information must also exist in those systems.

The Enterprise Trader Model uses a single service description representation for the following reasons:

- With a constant size service description, the structures required to store service descriptions are much simpler resulting in faster searching, the possibility of caching and simple transmission of service descriptions.
- If service type descriptions are not used, a system wide type library is not required, allowing the implementation of a distributed trader system to be more easily accomplished.
- We believe that the most common request to the trader will be for an instance of a particular class with few other constraints. If more complex queries are required then running a script over the service interface provides a more flexible alternative to storing data describing the service with the trader. An example could be that I want to find a mail server that has my mail on it. My name won't be a parameter for the service type under the RM-ODP model but should be available through interfaces exported by the mail server.
- The generation and use of service types adds another level of complexity into the system. Clients and servers must know the service type aprori in order to provide the correct information to find or register a service. In the RM-ODP model, the responsibility for developing service type descriptions has not been associated with either the client or the server resulting the possibility of multiple service type descriptions existing and evolving separately from both clients and servers.

In both the RM-ODP model and the CORBA Trading Object Service there is no constraint on the possible interconnections between traders. This means that both these models have

weak specifications about how search domains are specified, about when searches terminate and about where services are published. In CORBA Trading Object Service, to specify a search domain, the client can only specify the maximum number of links to traverse from a given starting trader and the maximum number of service entries to search. The client has no control over which links are searched or at which point the search terminates. When publishing a service, the client has no control over where the service is published or to whom the service is exposed. The inclusion of an interconnection policy in the Enterprise Trading Model provides the client with enough information to control a search and to select the domain to which services are published.

A distributed trader model has been proposed in [4]. This model takes a single logical trader and distributes it using the X500 architecture. The result is a complex mapping from service type to database. Distributing a trader in this manner improves query performance but does not yield benefits such as location dependent searching which promotes the discovery of local services first and allows caching of past query results. It also does not provide the client with any intuition as to the trader's structure thus preventing any input from the client to guide searching and publication.

# 4 IMPLEMENTATION

A prototype trader has been implemented in C++ using DCOM on Windows NT 4.0. The following section describes the trader architecture. Section two details how the trader integrates with DCOM.

## 4.1 Overview

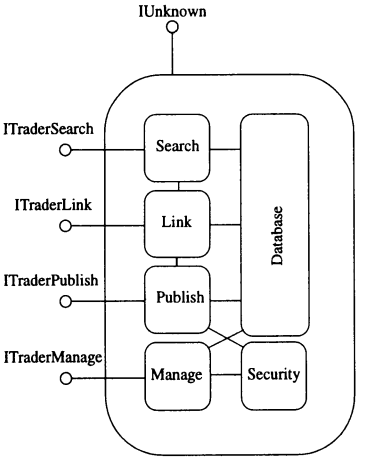


**Figure 2** The Trader Architecture.

The trader has been divided into a number of separate modules to simplify implementation and to separate areas of functionality. The architecture is shown in Figure 2. The database module encapsulates the data structures required to store the trader service descriptions, link information and access control. The search and publish modules support the *ITraderSearch* and *ITraderPublish* interfaces and the management module supports the *ITraderManage* interface. The link module supports interaction with other traders using the *ITraderLink*

interface and the security module provides support for authentication of users publishing to the trader.

The database module provides an interface for querying and managing the data structures used by the trader. To simplify prototyping all the data structures are stored in tables within an SQL database. This database is accessed using the Jet database engine which provides support for executing SQL queries and for managing replicated databases.

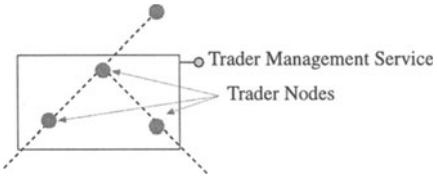


**Figure 3** The trader management service may support multiple trader nodes.

A Trader Management Service (Figure 3) runs on each workstation. This service provides a container allowing a single workstation to support multiple trader nodes. It provides common support required by the trader nodes in the form of a database engine and a scripting environment as well as providing a common access point to the trader services. Nodes may be freely migrated between container services and nodes from multiple hierarchies may exist within a single container.

## 4.2 Integration with DCOM

To allow the trader to integrate transparently with DCOM, the trader service intercepts calls to a number of system API's. The calls that are captured are those used to create object instances and those that register and deregister objects from the running object table. Capturing calls to create objects allows the trading service to provide connections to remote objects without modifying existing software. If a service is not available locally and can be run remotely, the trading service initiates a simple search to find the object. When a service is running it can register itself in the running object table, allowing other objects to connect to it instead of creating new instances. When selecting service instances to return this information is used by the trading service to bias towards returning services that are already running.

The registry is a system database which contains details of the objects available on a system and how to create them. When a trading service is started on a system, this information is extracted from the registry and added to the local service table. This allows clients searching the trader directly to find services that are locally available first.

The service control manager (SCM) is the mechanism by which remote objects are created under DCOM. It supports a single interface which takes a class identifier and location of the remote object and returns an pointer to an interface on that object. At the same time it establishes proxies and stubs to marshal remote procedure calls. The SCM interface is used by the trader to create connections to remote objects.

To identify services to the trader a COM type called a Moniker is used. Monikers are a flexible naming system allowing the identification of class instances. They are persistent and allow dynamic binding to service instances. Since Monikers can be converted into a textual representation they can be easily stored within a database.

# 5 FUTURE DEVELOPMENT

There are several areas in which we plan to carry out further research; the following sections outline a few of these.

## 5.1 Support for resource management

The structure of the trader lends itself to a number of forms of resource management which may be implemented by capturing and processing service requests. Simple examples could be load balancing across a number of service instances or selecting the node on which to create a particular service instance. The trader is a natural place in which to intercept queries for particular resources and redirect them to appropriate resource management agents.

To support this it is proposed to develop a resource management interface for the trading service which allows particular service classes to be attached to a resource management service. The particular resource manager could either be generic, providing for example round robin allocation of resource requests across a number of servers or specific, using information about a particular service to optimise the provision of that service. It is not intended that the actual resource management modules be included within the trader as this would result in unnecessary complexity and lack of flexibility; instead the trader will redirect requests via the resource management service.

## 5.2 Active searches

The prototype implementation does not yet address active scripted queries. These are queries where a script is passed as part of the query process and is then run on each service that meets the basic search criteria. It is proposed that an engine such as the Java Virtual Machine will be used to execute the scripts thus providing a flexible and secure mechanism by which clients can provide customised queries to select appropriate services. This method may be slower than storing property information with the service description but it is expected that those searches that required this level of detail in selecting the service will benefit from the increased flexibility while searches that require generic services will be processed faster than would otherwise be possible.

## 5.3 Trader topologies

The basic constraints applied to the topologies in this model have resulted in significant improvements in the usability and efficiency of the trader model. Further investigation is required into modifying the topologies to provide support for migration regions, to optimise the choice of search path and into policies for dividing the service space into separate hierarchies.

## 5.4 Trader toolset

A set of tools for specifying publication and search domains, modifying trader interconnection topologies and controlling access permissions would provide support for integration of trader services into commercial system. It is envisaged that these tools will use graphical representations of the trader interconnection topologies to allow users to directly select branches on which to publish service or over which to perform searches.

## 6 CONCLUSION

This paper has presented an Enterprise trading model with particular application to DCOM. The model provides insight for the client into the trader structure allowing the client to control the domain to which any services they publish are exposed. It has achieved this through imposing constraints on the interconnection policies for the trader services. These constraints have also enabled a simpler implementation capable of significant performance increases through caching of service entries.

We have also simplified the service representation within the trader model, removing the need not only to predefine service types but also to propagate this information to other traders and clients. The simpler internal data representation should allow searches to execute faster and service descriptions to be propagated more efficiently. In order to allow more complex queries it is proposed to use scripts acting directly on the service implementation. These scripts use knowledge already available to the client in the form of the service interface and provide a more powerful tool than could be implemented using rules constrained to data within the trader.

A prototype of the trader service has been implemented on DCOM and is currently undergoing testing.

## 7 ACKNOWLEDGMENTS

The authors would like to acknowledge the help of David Holmes in providing valuable feedback during the development of this paper and Brent Curtis who was closely involved in the initial development of the ideas underlying the trader interconnection model.

## 8 REFERENCES

[1] The Component Object Model Specification Microsoft (1995).
[2] ISO/IEC DIS 13235 Draft Rec. X.9tr - ODP Trading Function, ISO/IEC June 1995.
[3] OMG RFP5 Submission: Trading Object Service, OMG, May 1996.
[4] A. Richman and D. Hoang, Trader Interoperability: Why Two Models Are Better Than One. in Proc. *ISCIS'95 The Tenth International Symposium on Computer and Information Science*, Turkey, 1995.
[5] J. Deschrevel, The ANSA Model for Trading and Federation APM.1005.01 Architecture Projects Management Limited (APM), Cambridge UK, July 1993.
[6] M. Bearman and K. Raymond, Federating Traders: An ODP Adventure, *Proceedings of the IFIP TC6/WG6.4 International Workshop on Open Distributed Processing*, Berlin, Germany, 8-11 October, 1991, publ. North-Holland, pp. 125-141.
[7] L. Augusto and E. Madeira, A Model for a Federated Trader, *Proceedings of the International Conference on Open Distributed Processing,* February 1995, Brisbane, Australia.
[8] K. Mueller-Jones, M. Merz and W. Lamersdorf, "The TRADEr: Integrating Trading into DCE", *Proceedings of the International Conference on Open Distributed Processing*, February 1995, Brisbane, Australia.
[9] M. van Steen, F.J. Hauck and A.S. Tanenbaum, A Scaleable Location Service for Distributed Objects, *Proceedings of the Second Annual ASCI Conference*, Lommel, Belgium, June 1996, pp. 180-185.