

## Traders: the Yellow Pages for Middleware

Mirion Bearman  
Kerry Raymond  
CRC for Distributed Systems Technology

### Management Introduction

Every day, customers use the telephone Yellow Pages to locate the goods and services that they need. The telephone Yellow Pages is organised by the classification of services, and entries in the Yellow Pages usually contain additional information to assist customers in choosing suitable suppliers. Middleware clients have the same need to locate middleware services that match the client's needs based on the type and characteristics of those middleware services. In middleware, it is *traders* that provide the Yellow Pages service.

A trader is a middleware component that enables clients to find suitable servers at run-time. Like the telephone Yellow Pages, a trader enables a client to browse the set of servers. But unlike the telephone Yellow Pages, the trader also can select services that match the client's needs. By using a trader, a distributed application can be programmed to automatically reconfigure itself when servers fail or are replaced by new servers or whenever a better service becomes available. Without traders, programmers typically must "hard-wire" server names into client programs; reconfiguration requires programmer intervention and application down-time, both highly undesirable for mission-critical applications.

This analysis introduces the concept of traders and demonstrates how traders deliver value to organisations, at minimal investment risk.

### Trader Overview

A trader is a middleware component that enables servers to *export* (advertise) their services and enables clients to *import* (select) suitable servers based on the type and characteristics of the services. Figure 1 shows the interactions of a trader with a client and a server.

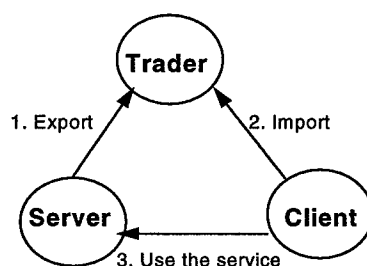


Figure 1: Clients and servers interacting with a trader

## Step 1: Export

When a server wishes to advertise its service, it *exports* a *service offer* to a trader, which stores the service offer in its database. A service offer contains:

- the service type that identifies the kind of service on offer
- the interface reference (low-level address information used by the middleware to locate a server)
- other properties (name-value pairs) which characterise this service

Properties are assertions made by the server about its service. Figure 2 shows an example of a service offer for a bank's Savings Account service.

Savings_Account		----->	Service Type
<Reference to Brisbane_City_Branch>		----->	Interface Reference
Bank_Name	"Best Bank" (string)	----->	Properties
Interest_Rate	8.45 (float)		
Cheque_Book	true (boolean)		
Expiry_Date	31/12/96 (date)		
Cheque_Clearance	3 (day)		

Figure 2: An example of a service offer

Properties are used by the clients to select the most appropriate service from the range of similar services. For example, these properties can characterise:

- functional aspects of the service, e.g. cheque writing facility on a savings bank account
- commercial aspects of the service, e.g. interest rate on credit balances
- quality of the service information, e.g. cheque clearance time
- information about the offer itself, e.g. the expiry date of the offer

## Step 2: Import

A client imports service offers from a trader by specifying:

- the required service type, e.g.

Savings\_Account

- the required property values, expressed in a notation similar to SQL and many popular programming languages, e.g.

Interest\_Rate > 8.25 and Cheque\_Book

- the preferred property values, usually expressed in terms of maximising or minimising certain properties, e.g.

max Interest\_Rate

The *required* property values are used to find suitable services, whereas the *preferred* property values are used to select the best of those found to be suitable.

In the telephone Yellow Pages, the service provider decides what extra information (if any) is provided about their service (e.g. special features, opening hours). With each entry having different information, it is difficult for the customer to compare the services based solely on these entries. In contrast, service offers exported to the trader are more structured. Each service type has a set of mandatory properties and a set of optional properties, all with well-defined types. The server must specify the values of all mandatory properties in the exported service offer, and can add extra optional properties if desired. Defining the properties of a service type ensures that the client has a meaningful basis for comparison between similar services.

The trader processes the import request by extracting the service offers of the nominated service type from its database which satisfy the required properties. The trader then ranks these suitable service offers according to the preferred property values and returns the “best” service offers to the client.

### **Step 3: Using the service**

Using the interface reference contained in a returned service offer, the client can invoke the desired service. By using a trader to discover suitable services at run-time, distributed applications can easily reconfigure themselves to make best use of the changing environment in which they operate. The clients involved do not require any prior knowledge of servers that they might use at run-time, avoiding the need to hard-wire server names in the application.

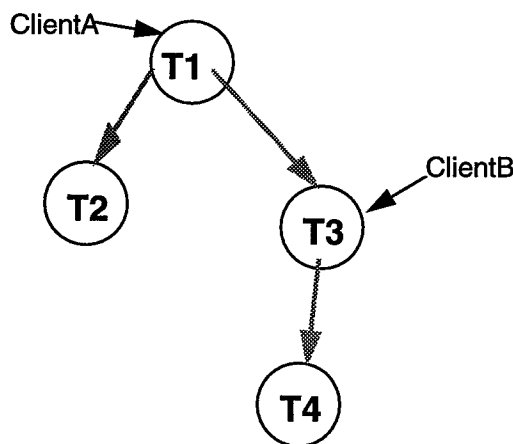
### **Scalability through Federation: Evolving Systems**

If middleware is to be successful, its components must be capable of managing the complexity of large-scale systems. However, most systems do not start as large systems, but evolve into larger wider-reaching systems over a period of time. Realistically, many traders will be initially deployed as stand-alone components to manage service offers within a particular part of an organisation. Gradually, applications will need to interwork with a wider range of services offered by other parts of the organisation, by customers and suppliers, or by the world at large, e.g. through the Internet. Therefore, the trader will need to be able to inform the clients about the ever-growing set of available services.

One solution to this evolution would be to export a service offer for every service into every possible trader. However, for reasons of autonomy, security, efficiency and availability, the more effective solution is to *federate* the traders, enabling the traders to share their knowledge of the services in their individual domains.

A trader federation is created by placing *links* between traders. When a trader is linked to a second trader, the first trader can call on the second trader for information about service offers held by the second trader. Thus the clients of the first trader have access to the combined set of service offers (subject to any restrictions imposed by the second trader). Note that the clients of the second trader do not have access to the first trader's service offers (unless a second reverse link is established), enabling tight control on the visibility of service offers. Federation provides an efficient yet controlled sharing of service offers.

There is no need for a trader to be linked directly to all other traders, since import requests can be propagated through the *trading graph* of indirectly linked traders, as illustrated in Figure 3. To propagate an import request in a trading graph, each trader acts as a client to its linked traders, passing the original client's import request throughout the trading graph. In Figure 3, a client of trader T1 could have its import request propagated from T1 to T2 and T3, and from T3 to T4. However, a client of trader T3 could only have its import request satisfied by the service offers held by traders T3 and T4.



**Figure 3: An example of a trading graph**

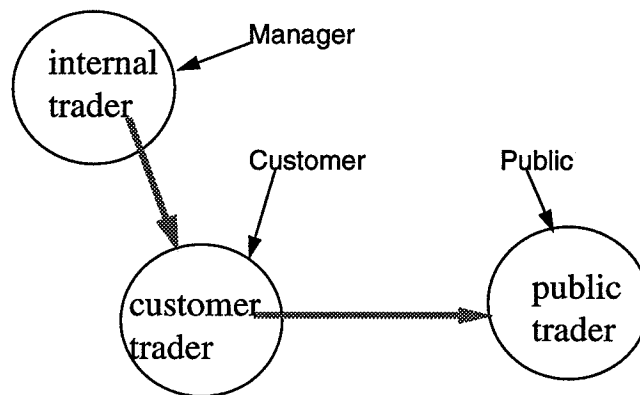
Federated traders permit remote service offers (within the set of federated traders) to appear to a client as if those service offers are held by the client's local trader. It is not necessary for a client to know if service offers are obtained through trader federation, where the service offers returned were found, or any details of the federated traders. A client of a trader imports only from its local trader and will *transparently* access other traders in the federation group via the links of the trading graph.

### **Scalability through Federation: Devolving Systems**

In some organisations, the domain of a trader can grow dramatically, resulting in the trader having more service offers, servers, and clients that it can manage efficiently. Trader federations can also be used to support the partitioning of a large trader into smaller traders. By classifying service offers into groups appropriate to the structure and

operation of the organisation, the trader administrator can partition the total service offers of a large hard-to-manage trader into a number of smaller manageable traders. These traders can be linked into a federation to support the desired visibility between the separate traders and the parts of the organisation that they reflect.

Figure 4 illustrates how a bank might devolve its trader into three smaller traders. The first trader contains bank services that are only for internal use of the bank (e.g. investing the bank's funds on the short-term money market); the second trader contains bank services that are available to its customers (e.g. deposits and withdrawals from savings accounts), while the third trader contains bank services that are available to the public (e.g. creating a savings account or applying for a loan). By linking the traders as shown in Figure 4, the bank's internal operations would use the first trader and, through federation, be able to access service offers for all the bank's services. The bank's customers would use the second trader and would be able to find customer and public services of the bank. Finally, the general public would only be able to select from the public services available in the third trader.



**Figure 4 - Example of a devolved trader**

The advantages of such devolution include:

- exploiting the knowledge of the kinds of service offer held by each trader to enable more efficient searching of the trader graph
- simplifying the administration of the individual traders as they contain fewer service offers

### **Changing property values**

Traders will be deployed by a wide variety of organisations to meet the diverse needs of many distributed applications. Therefore, traders must be flexible, capable of being adapted to suit the needs of the organisation and its IT strategy, rather than the reverse.

A service offer is a description of a service. However, a service can evolve over time and so the property values of the service offer might change over time. Some property values will never change, some will change slowly, some will change rapidly. Traders are capable of supporting a wide range of volatility of property values held in service offers:

- *readonly* properties

If a property has a permanent value, then that property can be flagged as being a readonly property. Readonly properties reassure the clients that those characteristics of a server will not alter during the lifetime of the service offer. For example, in Figure 2, the `Bank_Name` and `Expiry_Date` could both be readonly properties.

- *modifiable* properties

If a property has a value that might change, but only infrequently (e.g. every week), then that property is a modifiable property which means that the server can advise the trader to update that property value in its service offer stored within the trader. Modifiable properties are the default in the trader. In Figure 2, `Cheque_Book` and `Cheque_Clearance` would be modifiable properties, as these characteristics might change but only on an infrequent basis.

- *dynamic* properties

The very nature of certain properties, e.g. queue length of a printer, dictates that they will be continually changing and requiring frequent updates. Instead, a server with such properties can choose to export a dynamic property value. A dynamic property value is an indirect reference which instructs the trader to obtain the property value when the value is required for evaluation by an import request. Typically, the indirect reference is a callback to the server to obtain its current property value. The use of dynamic properties ensures that a client's import request is evaluated with the most up-to-date information. The presence of dynamic properties is transparent to the importing client as the trader will only return actual values and not indirect references. Although dynamic properties are a powerful mechanism, the callbacks can degrade the performance of the import requests. Therefore, their use tends to be restricted to applications with a critical need for the most up-to-date information. In Figure 2, it is quite possible that none of the properties would have dynamic values. However, if a bank wishes to offer savings accounts linked to the international bond market, then maybe the `Interest Rate` might be a dynamic property.

## **Integration with legacy systems**

Some organisations deploying trader technology might already have information about services held by existing systems, e.g. in a systems management information base. Duplicating such information can cause inefficiency and inconsistency, so the trader has mechanisms to interwork with legacy systems.

A *proxy offer* is a specialised service offer that enables the support of legacy systems. An import-only interface is created as a front-end to the legacy system, and proxy offers are exported to the trader to describe, in general, the services available through the legacy system. When an import request matches the proxy offer, the trader forwards the import request to the associated legacy system to search for the exact services that match the import request. The legacy system can be viewed as a “secondary trader” hiding behind its proxy offer.

Through the trader’s ability to interwork with legacy systems, the organisation can leverage their existing IT investments.

### **Suitable for the novice and the expert**

Successful middleware components must be suitable for users at different levels of learning and sophistication. The trader is designed to be easy to use by the novice, and to provide a gradual learning curve as additional control is required.

The expert user can interact with the trader to solve complex problems and to fine-tune trader performance. Dynamic properties and proxy offers are examples of the extra flexibility available to the server. For the client, the trader has many *import policies* to provide fine-grained control over the searching and matching algorithms used within the trader when processing an import request.

For example, a client can specify that an import request should only be propagated through the federated trading graph if the request cannot be satisfied by service offers held in the local trader. The client can also set a *hop count* on a request which determines the maximum depth of propagation in the trading graph, to control the extent of the search and hence the time and cost involved.

Import policies are not an “all-or-nothing” situation. For a given request, even a sophisticated client need only concern itself with only a few of the import policies; any import policy not specified by the client will take a default value set by the trader. The novice user need not be aware of any of the import policies; the trader behaviour will be determined by the default values.

The trader is suitable for the full range of users, from the novice to the expert.

### **A safe investment**

In the rapidly changing IT world, there are too many passing fads and too much marketing hype. Organisations need to minimise their risks when selecting middleware technology, and need to be satisfied that the chosen technology will work and that it can be a long-term component of their IT architecture.

Traders can be used to discover services implemented in a variety of technologies ranging from CORBA and OLE objects to World Wide Web pages and Java applets. The open

design of trader even enables discovery of services implemented in future technologies. Indeed, a trader can be used whenever services need to be discovered using their type and characteristics.

Trader technology has had a decade of R&D in universities, research institutes, and industry. Much of this R&D was carried out collaboratively and has resulted in a *single* specification being adopted by

- the Object Management Group as a CORBA Object Service
- the International Standards Organisation as an International Standard
- the International Telecommunications Union (formerly CCITT) as their X.950 Recommendation

Traders conforming to these joint standards are currently in development and a number of products are expected to be available in early 1997.

### **Management Conclusion**

Trader is the middleware component that provides the Yellow Pages service. It enables the discovery of services based on their type and characteristics and the automatic reconfiguration of distributed applications in an evolving environment.

In conclusion, by incorporating trader technology as part of its IT infrastructure, an organisation can enjoy:

- scalability to accommodate the expansion of services, both within and beyond the organisation
- support for users of varying degrees of experience
- minimal risk due to widespread adoption and legacy support.