A GENERAL RESOURCE DISCOVERY SYSTEM FOR OPEN DISTRIBUTED PROCESSING

QINZHENG KONG AND ANDREW BERRY[1]

Abstract—**The networks of today support a wealth of resources which can aid the daily tasks of a diverse user base. Resource discovery is a relatively new field that deals with the problems of finding, organising and accessing these resources in a global network. The architecture model proposed by the DSTC's Architecture Unit provides a way to specify open distributed application systems. A resource discovery system is an example of such an application. This paper uses the DSTC architecture model to specify a general resource discovery system.**

Keyword Codes—**C.2.4; H; H.2.c**

Keywords—**Distributed System; Information Systems; General Model**

Source of Publication—**Open Distributed Processing—Experiences with distributed environments, IFIP, Raymond and Armstrong (eds.), Chapman & Hall, 1995**

1 INTRODUCTION

The distributed systems of today support a wealth of resources which can aid the daily tasks of a diverse user base, ranging from school children to business professionals. The types of resources available include network services, documents, software, video, sounds and images. Resource discovery is a relatively new field that deals with the problems of finding, organising and accessing these resources in an open distributed system.

The Architecture Model [1] proposed by the DSTC's Architecture Unit provides a way to specify open, distributed application systems. A resource discovery system in a distributed environment is an example of such an application. This paper uses the DSTC Architecture Model to specify the architecture of a general Resource Discovery System.

In the remainder of the document, section 2 gives a brief description of the concepts defined in the DSTC Architecture Model. Section 3 gives a general architecture for resource discovery systems. Sections 4 and 5 describe how the concepts defined in the architecture model can be used to specify a general resource discovery system. Section 6 describes some fundamental requirements of a distributed environment for general distributed applications. Finally, section 7 discusses future research directions in the distributed resource discovery system area.

2 BASIC CONCEPT OF THE ARCHITECTURE MODEL

Distributed applications in the DSTC architecture model are specified in terms of objects, their interface types and bindings between the interfaces of objects.

2.1 OBJECTS

In the DSTC architecture model, an object is an entity that encapsulates state, behaviour and activity (i.e. the ability to take independent action). Objects interact by exchanging strongly typed messages over a binding. Objects connect to a binding through their interfaces.

2.2 BINDINGS

A binding is an association between a set of objects that allows the objects to interact. Bindings are strongly typed—the binding type defines the roles of objects in a binding and the interaction that can occur between objects fulfilling those roles.

1. Andrew Berry, CRC for Distributed Systems Technology, The University of Queensland, Australia, 4072 <berry@dstc.edu.au>

Roles of a binding are filled by the objects participating in the binding. For example, a binding to support RPCs must have a "client" role and a "server" role. Each role of a binding specifies an interface type that must be satisfied by objects fulfilling that role—objects participating in the binding must therefore instantiate an interface that is a compatible with their role in the binding.

The fundamental unit of interaction in a binding is a single, strongly typed message, although more complex, high-level interactions can be defined, for example, RPC or multicast.

2.3 INTERFACES

An interface is instantiated to fulfil the role of an object in a particular binding. Interfaces are strongly typed—an interface type describes the structure and semantics for interactions of an object during a binding. In other words, an interface type describes the structure of messages and the object behaviour associated with messages sent and received by that object during a binding.

An object can offer many interface types and the set of interface types offered can vary over time.

3 AN ARCHITECTURE FOR GENERAL RESOURCE DISCOVERY SYSTEMS

A resource discovery system in today's distributed environment is a distributed application in general. On one hand, it shares many common features with all the other distributed applications. For example, its infrastructure supports naming and directory services, security facilities, underlying protocol transparency, alternative messaging routing, etc. On the other hand, it has its own characteristics in terms of the services provided to its user. Resource discovery services mainly include:

- identifying the possible location(s) of user interested information;
- selecting the information required by providing the searching and browsing facilities;
- presenting the information
- delivering the required information in user specified form to user specified destination.

In general a distributed resource discovery system consists of many sites. Each site can be described as having a *Server* and one or more *Clients*.

The *Server* is responsible for accessing local resources, communicating with other remote servers to access their resources, and providing common services required by its clients. The *Client* provides a mechanism for users to discover and access resources. The *Client* can be further decomposed into two major components: a *User* part which provides an interface specifically designed for the end-user environment; and an *Agent* part, which provides user specific services and communicates with the local server.

Each server can support more than one agent and each agent can support more than one user.

The architecture of such a system is illustrated in Figure 1:

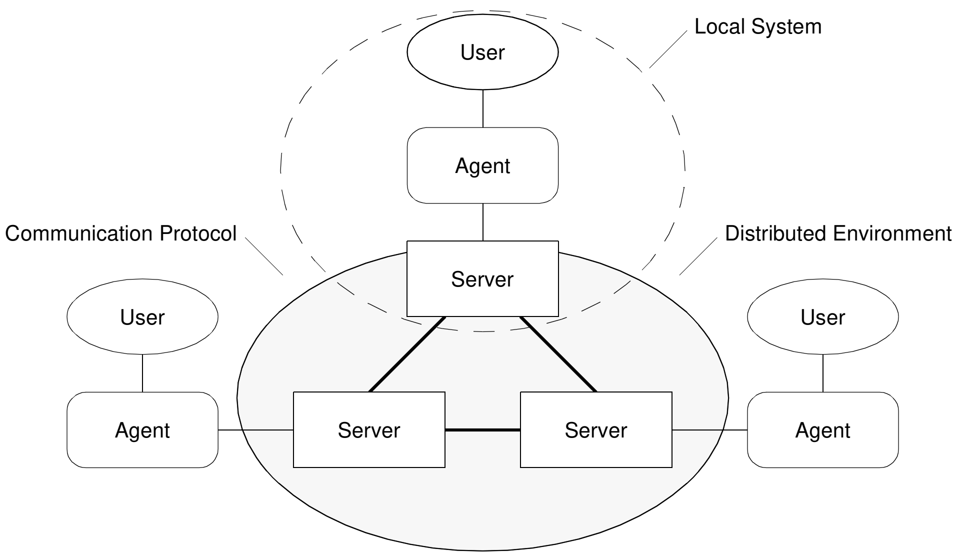

Figure 1: General Architecture of Resource Discovery System

Each local system can be refined as in Figure 2:

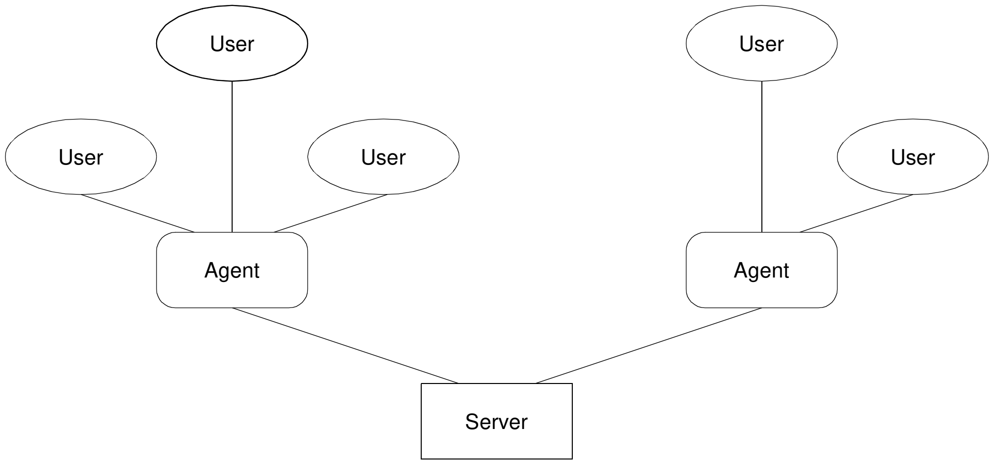

Figure 2: An Example of a Local System

4 THE SPECIFICATION OF A GENERAL RESOURCE DISCOVERY SYSTEM

As mentioned before, a resource discovery system in a distributed environment is only a particular case of a general distributed application. The DSTC architecture model can be used to specify a general resource discovery system.

The first step of specifying a distributed system using the model is to identify a set of basic *Objects*. Based on the architecture described in section3, there are three top level objects: a *User* an *Agent* and a *Server*.

The second step is to describe the interactions between these objects with a set of *Binding Types*. In our example of resource discovery system, three binding types are required, *User-Agent Binding Type*, *Agent-Server Binding Type* and *Server-Server Binding Type*. These binding types define the communication and interaction protocols used to support resource discovery.

From the binding types, we can identify and derive a set of *Interface Types* to be supported by these objects.

The interface types supported by the *User* object are *End-user Interface Type* and *Agent-request Interface Type*. The *End-user Interface Type* is used to specify the end-user interface, which might be a GUI interface. The *Agent-request Interface Type* defines the interactions a *User* can expect to have with an *Agent*.

The interface types supported by the *Agent* object are the *Agent-service Interface Type* and *Resource-user Interface Type*. The *Agent-service Interface Type* is used to specify the local services provided to clients of the agent and the parameters required for those services. The *Resource-user Interface Type* defines the interactions an *Agent* can expect to have with a *Server*.

The *Server* object supports two interface types, *Resource-service Interface Type* and *Remote-service Interface Type*. The *Resource-service Interface Type* is used to specify the services provided to an *Agent,* and the *Remote-service Interface Type* is used to specify the interactions that a *Server* can expect to have with other servers.

The relationship between the objects, interface type and binding types of a Resource Discovery System is illustrated in Figure 3:

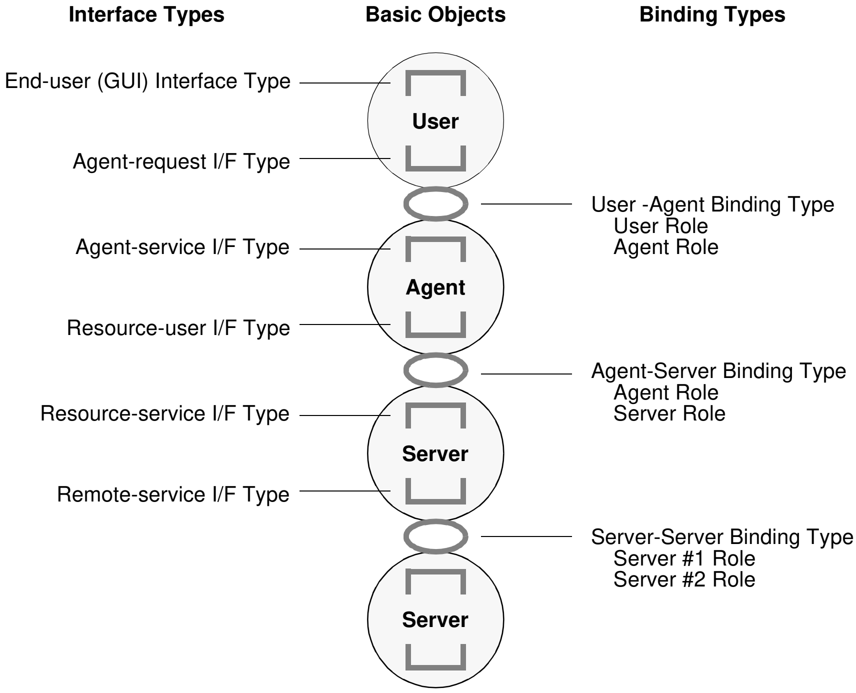

Figure 3: Object, Interface Types and Binding Types

Notice that separate interfaces are defined for the *User* and the *Agent* in the *User-Agent* binding type and similar separations in the other binding types. This is to allow for differences in what the user expects, and what the agent provides. An agent might be accessed by more than one user, and each user can have different expectations. Provided the expectations of the user are a subtype of the service provided by the agent, a binding between the user and agent is meaningful.

The following sub-sections describe the binding types in more detail. Detailed interface types are not described, to allow maximum flexibility in the definition of systems based on this architecture.

4.1 USER-AGENT BINDING TYPE

The *User-Agent Binding Type* specifies the possible interactions between the user and the agent objects. It has two roles, *user* and *agent*. A number of interactions can take place between a user and an agent. For example, the user can issue a request that is delivered to the agent, and it should always be followed by a response from the agent delivered to the user. (The handling of the request is a local issue of the agent and is not discussed here.) Another example of the interaction might be that the agent can issue a message to a number of users when a certain event occurred.

This binding type can be expressed using an arbitrary syntax as follows:

```
User-Agent Binding type
    Roles are: user, agent
    Interactions are:
        query:  user.request -> agent.request;
                agent.response -> user.response
        message:agent.sending -> user.receiving
```

As mentioned before, a binding type can be used to derive minimal interface types required for different roles. Each action taken by different roles should be specified as an action in the corresponding interface type. For example, the *Agent-request Interface Type* must support the action of sending requests and the *Agent-service Interface Type* must support the action of sending responses.

The binding type for a general resource discovery system specified above can be used as the basis for defining binding types in a specific system. A binding type for a system (RDS) based on Z39.50 [2], could be based on the general binding type as demonstrated below:

```
RDS-User-Agent Binding Type
Derived from User-Agent Binding Type
    Roles are: user, agent
        user has RDS User-Agent Interface using Z39.50
        agent has RDS Agent Interface using Z39.50
    Interactions are:
        query:init:... ...
        query:search:... ...
        query:present:... ...
        query:delete:... ...
        query:scan:... ...
        query:sort:... ...
        ... ...
        segment:... ...
        ... ...
```

In which, the *init*, *search*, etc. are specific queries derived from the *query* of the general *User-Agent Binding Type* and *segment* is a new interaction. Detailed descriptions of each interaction can be specified but are not given in this paper.

4.2 AGENT-SERVER BINDING TYPE

The *Agent-Server Binding Type* may be expressed as:

```
Agent-Server Binding Type
    Roles are: agent, server
    Interactions are:
        query: ... ...
        service: ... ...
        message: ... ...
```

It has the similar form as the *User-Agent Binding Type*. When the server object is refined to capture more detailed services supported by the server (Section 5), the binding type will have more details.

4.3 SERVER-SERVER BINDING TYPE

The binding type for interaction between servers is more complicated. The binding type can be specified either as a multi-party binding with many servers, or a set of 2-party bindings between servers.

In the case of multi-party binding, an instance of a binding type is a single binding which contains two or more servers. Each server in the binding has an equivalent role (i.e they are peers). Servers can choose to join and leave existing bindings. The following figure illustrates a multi-party binding:

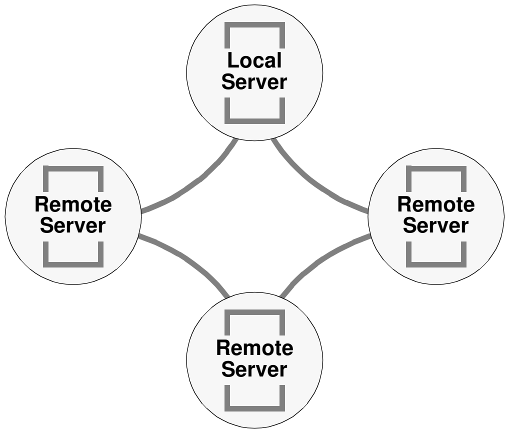

Figure 4: Multi-Party Binding Type

The *Multi-Party Binding Type* is often required in a real distributed system. A key problem of such a binding type is to define the policy of finding, then joining and leaving a binding. In a distributed environment, it might not be possible to have a centralised system to keep the information of all the existing bindings, so the policy should include how to find existing bindings in a distributed environment.

It is assumed that when a server wishes to join a binding, it should have the knowledge of at least one server which is already in the binding. With a join request, a set of conditions can be specified. When the server in the binding receives the join request, it is its responsibility to notify all the servers already in the binding that a new server has joined. If the server which receives the join request is a stand-alone server (not in any existing binding) then a new instance of the binding type is created which contains only two servers initially.

The *Multi-Party Binding Type* can be expressed as:

```
Multi-Party Server Binding Type
    Roles are: servset - a set of servers
    Interactions are:
        query:  ∃s:servset,
                 s.sending -> ∀r:servset-{s}: r.receive;
                ∀r:servset-{s},
                 r.response -> s.response
        join:   ∃s1:servset, s2: server,
                 s1.join(s2) -> ∀r:servset-{s1}:r.newserv(s2)
        leave:  ∃s: servset,
                 s.leave(s2) -> ∀r:servset-{s}:r.leaving(s2)
        ... ...
```

A number of interactions can be added to the above specification, such as, combine two existing bindings, create a new binding, etc.

In the case of a *2-Party Binding Type* (it is sometimes also referred to as peer-to-peer binding), each binding contains only two servers, one is referred as *Local Server* and the other as *Remote Server*. The following figure illustrates *2-Party* bindings.

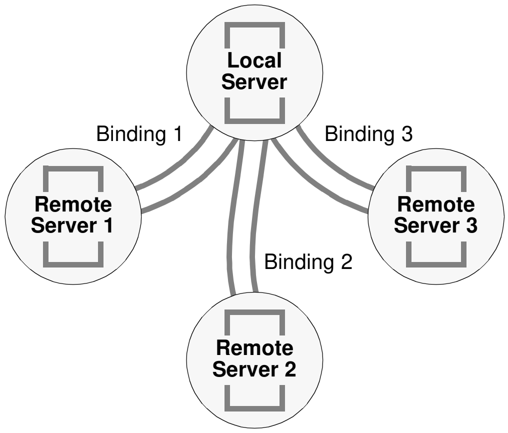

Figure 5: 2-Party (Peer-To-Peer) Bindings

Each binding instance in the above figure is a simple binding between a pair of servers. Although several servers are connected through indirectly by the *local server*, there are no direct bindings between these servers. If direct interactions between all servers are required, specific bindings have to be set up.

The *2-Party Binding Type* can be expressed as:

```
2-Party Server Binding Type
    Roles are: server1, server2
    Interactions are:
        query:  s:server1 or server2, r: not s,
                    s.sending -> r.receive
                r.response -> s.response
        create: s:server1 or server2, r: not s,
                    s.create -> r.create
        delete: s:server1 or server 2, r: not s,
                    s.delete -> r.delete
        ... ...
```

It is possible that within one system, both the multi-party binding and the 2-party binding are used at the same time. From the descriptions above, it is apparent that multi-party bindings provide significantly more flexibility in describing the interactions at the expense of complexity.

5 REFINEMENT OF TOP LEVEL OBJECTS

The top level objects of the general resource discovery system only give an abstract view of the system. These objects can be refined to capture additional detail. For example, apart from normal query access, an agent object can provide some intelligent services. An example of intelligent agent service might be to register the interests of a user whenever that user issues a request. This data can be used by the agent to intelligently gather information for delivery to the user.

Another example of an additional service might for the server and the agents to provide an administrative interface for configuration and administration.

Hence, based on these object refinements, more binding types are required which can either be derived from existing binding types or be newly defined types. Some examples of these binding types are given in the following sections.

5.1 INTELLIGENT AGENT BINDING TYPE

The intelligent agent binding type can be derived from the general *User-Agent Binding Type* with an extra operation performed by the agent to register user's interests:

```
Intelligent-Agent Binding type
    Derived from User-Agent Binding Type
    Roles are: user, agent
    Interactions are:
        query:  user.request -> agent.request and
                    agent.register;
                agent.response -> user.response
        notify: agent.notify -> user.notify
```

5.2 ADMIN-SERVER BINDING TYPE

An administrator is a special user to the system. Apart from normal access through local agent interface, the administrator might have direct access to the Server for administration. A new binding type is required to capture these abilities.

A template of *Admin-Server Binding Type* can be given as:

```
Admin-Server Binding type
    roles are: admin, sever
    interactions are:
        insert:
        delete:
        update:
        backup:
        verify:
        security:
        management:
        ... ...
```

5.3 REFINEMENT OF THE SERVER

The refinement of the server object can be achieved by specifying a set of *Service* objects embedded in the *Server* object. That is, each service provided by the server can be defined as an object. For a general resource discovery system, the following service objects might be defined:

- Security Service Object — This performs the service level security checking, including the checking of access made by local users or remote servers.
- Directory Service Object — This provides the X.500 Directory Services. The information in the directory may include the information about other servers in the system and information about existing Bindings.
- Trader Object — This provides a “yellow pages” service, for example, an ODP Trader [3,4,5].
- Register Object — This is an alternative to the Trader Object. It provides a simple mechanism for information providers to register themselves with the distributed system.
- Type Management Object — This object provides a service to determine the compatibility of types [6].
- Search and Retrieve Object — This is the search engine of the system. The search and retrieve are based on the system knowledge maintained and managed by other objects, such as Directory Object, Trader Object, Register Object and Type Management Object. User required information/resource can be discovered by the co-operation among these objects.
- Local Database Management Service — This might be any kind of persistent data storage management system. This service may be used by other Service objects such as the Directory Service objects, the Type Management objects and the Trader objects.
- Accounting Service Object — This provides resource control service for the user.
- Logging Service Object — This object records events of interest. A general logging service can be used for many different purposes, e.g. security logging, search event logging, etc.

- Federation Object — This object is responsible for the federation among different Servers. Its activities include joining and leaving the federation, communication agreement to other servers, security agreements and many other issues.

Some of the above services may be defined as common services (e.g. the *Accounting Service*, the *Logging Service* and the *Security Service)* shared with other applications running on the same node. Other services might be specific to the resource discovery system. Each of these services can be specified as an object. Figure 6 illustrates the relationship between these service objects. Different interface types and binding types among these objects are also illustrated:

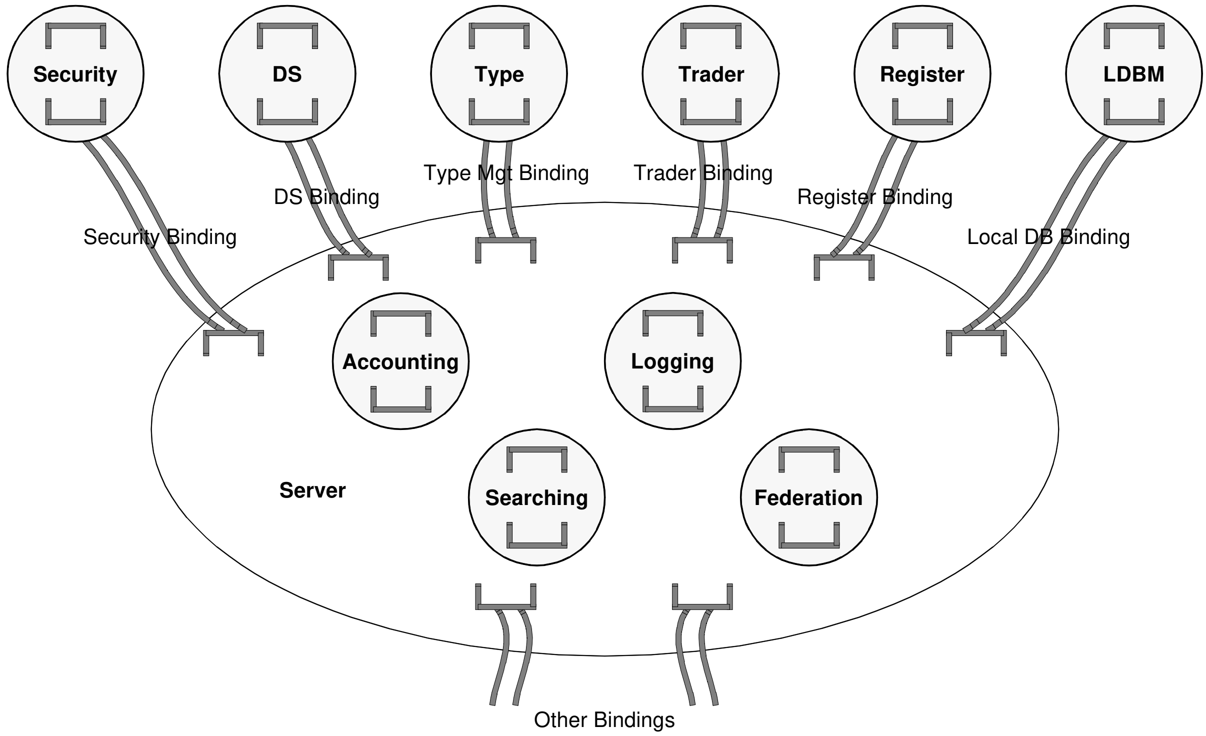

Figure 6: Service Objects and the Binding Types

6 DISTRIBUTED ENVIRONMENT

Ideally, a resource discovery system would be built on top of a standard distributed environment. Such an environment must provide location transparency and protocol independence. Each component of the resource discovery service can be defined as services supported by the distributed environment and can be dynamically added in the environment when needed. The abstract view of such an environment is illustrated in the following figure:

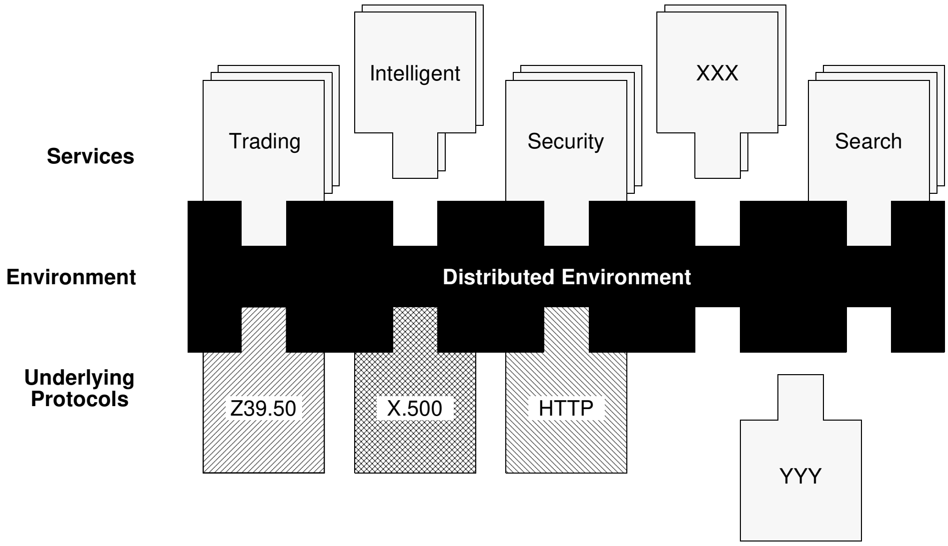

Figure 7: Resource Discovery Services in a Distributed Environment

It can be seen from Figure 6 that the resource discovery services can be built together with many other distributed applications in a sufficiently rich distributed environment.

7 DISCUSSION

A brief specification of a general resource discovery system using the DSTC Architecture Model has been given. It demonstrated that a resource discovery system is only a special example of a distributed application, and it can be built on top of an open distributed environment with other distributed applications.

Further refinement of this specification of a general resource discovery system will include the following tasks:

- Each binding type will be refined in order to capture all the possible interactions between different roles.
- Interactions in each binding type will be refined to specify the details of the actions and to give the allowed parameters in each action
- More detailed interface types will be derived from the binding types.
- Formal specification languages, such as LOTOS and/or Object-Z should be used to give more accurate definition of the objects, interface types and binding types.

8 ACKNOWLEDGEMENTS

Thanks to numerous research staff of the DSTC Resource Discovery Unit, the DSTC Architecture Unit and the University of Queensland Computer Science Department for useful discussions and suggestions. The work reported in this paper is funded in part by the Cooperative Research Centres Program through the department of the Prime Minister and the Cabinet of the Commonwealth Government of Australia.

REFERENCES

[1] A. Berry and K. Raymond, *The A1✓ Architecture Model*, submitted to International Conference on Open Distributed Processing (ICODP) 1995.

[2] ANSI/NISO Z39.50-1994 Information Retrieval: Application Service Definition and Protocol Specification, Draft 1994.

[3] ISO/IEC JTC1/SC21, *Draft Recommendation X.903: Basic Reference Model of Open Distributed Processing - Part 3: Prescriptive Model* (ISO/IEC DIS 10746-3.1), February, 1994.

[4] ISO/IEC JTC1/SC21: *ODP Trader*, 1994.

[5] M. Bearman, *ODP-Trader*, Proceedings of the International Conference on Open Distributed Processing 93 (ICODP'93), Berlin, September 1993.

[6] J. Indulska, M. Bearman and K. Raymond, *A Type Management System for an ODP Trader*, Proceedings of the International Conference on Open Distributed Processing 93 (ICODP'93), Berlin, September 1993.