An Architecture for the UniFrame Resource Discovery Service[1]

Granditha N. Siram, Rajeev R. Raje, Andrew M. Olson
Department of Computer and Information Science
Indiana University Purdue University Indianapolis
723 W. Michigan Street, SL 280
Indianapolis, IN 46202-5132, USA
Email: {nnayani, rraje, aolson}@cs.iupui.edu

Barrett R. Bryant, Carol C. Burt
Department of Computer and Information Sciences
The University of Alabama at Birmingham
115A Campbell Hall, 1300 University Boulevard
Birmingham, AL 35294-1170, USA
Email: {bryant, cburt}@cis.uab.edu

Mikhail Auguston
Department of Computer Science
New Mexico State University
PO Box 30001, MCS CS
Las Cruces, NM 8803
Email: mikau@cs.nmsu.edu

Abstract

Frequently, the software development for large-scale distributed systems requires combining components that adhere to different object models. One solution for the integration of distributed and heterogeneous software components is the *UniFrame* approach. It provides a comprehensive framework unifying existing and emerging distributed component models under a common meta-model that enables the discovery, interoperability, and collaboration of components via generative software techniques. This paper presents the architecture for the resource discovery aspect of this framework, called the UniFrame Resource Discovery Service (URDS). The proposed architecture addresses the following issues: a) dynamic discovery of heterogeneous components, and b) selection

[1] This research is supported by the U. S. Office of Naval Research under the award number N00014-01-1-0746.

of components meeting the necessary requirements, including desired levels of QoS (Quality of Service). This paper also compares the URDS architecture with other Resource Discovery Protocols, outlining the gaps that URDS is trying to bridge.

1. Introduction

Software realizations of distributed-computing systems (DCS) are currently being based on the notions of independently created and deployed components, with public interfaces and private implementations, loosely integrating with one another to form a coalition of distributed software components. Assembling such systems requires either automatic or semi-automatic integration of software components, taking into account the quality of service (QoS) constraints advertised by each component and the collection of components. The UniFrame Approach (UA) [12][13] provides a framework that allows an interoperation of heterogeneous and distributed software components and incorporates the following key concepts: a) a meta-component model (the Unified Meta Model – UMM [11]), b) an integration of QoS at the individual component and distributed system levels, c) the validation and assurance of QoS, based on the concept of event grammars, and e) generative rules, along with their formal specifications, for assembling an ensemble of components out of available choices. The UniFrame approach depends on the discovery of independently deployed software components in a networked environment. This paper describes an architecture, URDS (UniFrame Resource Discovery Service), for the resource discovery aspect of UniFrame. The URDS architecture provides services for an automated discovery and selection of components meeting the necessary QoS requirements. URDS is designed as a Discovery Service wherein new services are dynamically discovered while providing clients with a Directory style access to services. The result of using URDS, the UA and its associated tools is a semi-automatic construction of a distributed system.

The rest of the paper is organized as follows. Section 2 discusses related resource discovery protocols. Section 3 discusses the UniFrame approach and the URDS architecture. An example is presented in section 4. A brief comparison of URDS and other approaches is presented in section 5. Details of an initial prototype and experimentations are indicated in section 6 and the paper concludes in section 7.

2. Related Work

The protocols for resource discovery can be broadly categorized into: a) ***Lookup (Directory) Services and Static Registries*** *and b)* ***Discovery Services.*** A few prominent approaches are briefly discussed below.

Universal Description, Discovery and Integration (UDDI) Registry: UDDI [16] specifications provide for distributed Web-based information registries wherein Web services can be published and discovered. Web Services in UDDI are described using Web Services Description Language (WSDL) [4] -- an XML grammar for describing the capabilities and technical details of Simple Object Access Protocol (SOAP) [1] based web services.

CORBA Trader Services: The CORBA Trader Service [10] facilitates 'matchmaking' between service providers (*Exporters*) and service consumers (*Importers*). The exporters register their services with the trader and the importers query the trader. The trader will find a match for the client based on the search criteria. Traders can be linked to form a *federation of traders,* thus making the offer spaces of other traders implicitly available to its own clients.

Service Location Protocol (SLP): SLP [6] architecture comprises of *User Agents (UA), Service Agents (SA),* and *Directory Agents (DA).* UAs perform service discovery on behalf of clients, SAs advertise the location and characteristics of services and DAs act as directories which aggregate service information received from SAs in their database and respond to service requests from UAs. Service requests may match according to service type or by attributes.

JINI: JINI [15] is a Java-based framework for spontaneous discovery. The main components of a JINI system are *Services*, *Clients* and *Lookup Services*. A service registers a "service proxy" with the Lookup Service and clients requesting services get a handle to the "service proxy" from the Lookup Service.

Ninja Secure Service Discovery Service (SSDS): The main components of the SSDS [5], [9] are: Service Discovery Servers (SDS), Services and Clients. SSDS shares similarities with other discovery protocols, with significant improvements in reliability, scalability, and security.

3. UniFrame and UniFrame Resource Discovery Service (URDS)

The Directory and Discovery Services, described earlier, mostly do not take advantage of the heterogeneity, local autonomy and the open architecture that are characteristics of DCS. Also, a majority of these systems operate in one-model environment (e.g., CORBA Trader service assumes only the presence of CORBA components). In contrast, a software realization of a DCS will most certainly require a combination of heterogeneous components – i.e., components developed under different models. In such a scenario, there is a need for a discovery system that exploits the open nature, heterogeneity and local autonomy inherent in DCS. The URDS architecture is one such solution for the discovery of heterogeneous and distributed software components.

3.1. UniFrame Approach

3.1.1. Components, Services and QoS

Components in UniFrame are autonomous entities, whose implementations are non-uniform. Each component has a state, an identity, a behavior, well-defined public interfaces and private implementation. In addition, each component has three aspects: a) ***Computational Aspect****:* it reflects the task(s) carried out by each component, b) ***Cooperative Aspect****:* it indicates the interaction with other components, and c) ***Auxiliary Aspect****:* this addresses other important features of a component such as security and fault tolerance.

Services, offered by a component in UniFrame, could be an intensive computational effort or an access to underlying resources. The ***QoS*** is an indication given by a software component about its confidence to carry out the required services in spite of the constantly changing execution environment and a possibility of partial failures.

3.1.2. Service Types

Components in UniFrame are specified informally in XML using a standard format. XML [3] is selected as it is general enough to express the required concepts, it is rigorously specified, and it is universally accepted and deployed. The UniFrame service type, which represents the information needed to describe a service, comprises of:

ID**:** A unique identifier comprising of the host name on which the component is running and the name with which this component binds itself to a registry will identify each service.

ComponentName**:** The name with which the service component identifies itself.

Description**:** A brief description of the purpose of this service component.

Function Descriptions**:** A brief description of each of the functions supported by the service component.

Syntactic Contracts**:** A definition of the computational signature of the service interface.

Function**:** Overall function of the service component.

Algorithm**:** The algorithms implemented by this component.

Complexity**:** The overall order of complexity of the algorithms implemented by this component.

Technology**:** The technology used to implement this component (e.g., CORBA, Java RMI, etc.).

QoS Metrics**:** Zero or more *Quality Of Service (QoS) types*. The *QoS type* defines the QoS value type. Associated with a QoS type is the triple of <*QoS-type-name, measure, value*> where *QoS-type-name* specifies the QoS metric, for example, *throughput*, *capacity*, *end-to-end delay,* etc. *Measure* indicates the quantification parameter for this type-name like *methods completed/sec*, *number of concurrent requests handled*, *time,* etc. *Value* indicates a numeric/string/boolean value for this parameter. We have established a catalog of Quality of Service metrics that are used in UniFrame specifications [2].

Figure 1 illustrates a sample UniFrame specification. This example is for a bank account management system with services for deposit, withdraw, and check balance. This example assumes the presence of a Java RMI server program and a CORBA server program, which are available to interact with the client requesting their services. We will return to this example in detail when we describe the resource discovery service.

```
<UniFrame>

    <ComponentName> AccountServer </ComponentName>
    <Description> Provides an Account Management System </Description>

    <FunctionDescription>
        <Function> javaDeposit </Function>
        <Function>javaWithdraw </Function>
        <Function> javaBalance </Function>
    </FunctionDescription>

    <ComputationalAttributes>
        <InherentAttributes>
            <ID> intrepid.cs.iupui.edu/AccountServer </ID>
        </InherentAttributes>
    </ComputationalAttributes>

    <FunctionalAttributes>
        <Function> Acts as Account Server </Function>
        <Algorithm> Simple Addition/Subtraction </Algorithm>
        <Complexity> O(1) </Complexity>
        <SyntacticContract>
            <Contract> void javaDeposit(float ip) </Contract>
            <Contract> void javaWithdraw throws OverDrawException</Contract>
            <Contract> float javaBalance() </Contract>
        </SyntacticContract>
        <Technology> Java-RMI </Technology>
    </FunctionalAttributes>

    <CooperatingAttributes>
        <PreprocessingCollaborators> AccountClient </PreprocessingCollaborators>
    </CooperatingAttributes>

    <AuxillaryAttributes>
        <Mobility> No </Mobility>
    </AuxillaryAttributes>

    <QOSMetrics>
        <Availability measure=”%”> 90 </Availability>
        <End2EndDelay measure=”ms” > 10 </End2EndDelay>
    </QOSMetrics>

</UniFrame>
```

Figure 1: Sample UniFrame Specification in XML

3.2 URDS

The main components of the URDS architecture (illustrated in Figure 2) are: i) Internet Component Broker (ICB), ii) Headhunters (HHs), iii) Meta-Repositories, iv) Active-Registries, v) Services, and vi) Clients. Other details in the figure will be explained in the following sections. The numbers indicate the flow of activities in the URDS. These are explained, in detail, in the context of an example in section 3.2.7. The URDS architecture is organized as a federation in order to achieve scalability. Figure 3 illustrates the federation aspect of URDS.

Every ICB has zero or more headhunters attached to it. The ICBs in turn are linked together with unidirectional links to form a directed graph. The URDS discovery process is “administratively scoped”, i.e., it locates services within an administratively defined logical domain. ‘*Domain’* in UniFrame refers to industry specific markets such as Financial Services, Health Care Services, Manufacturing Services, etc.

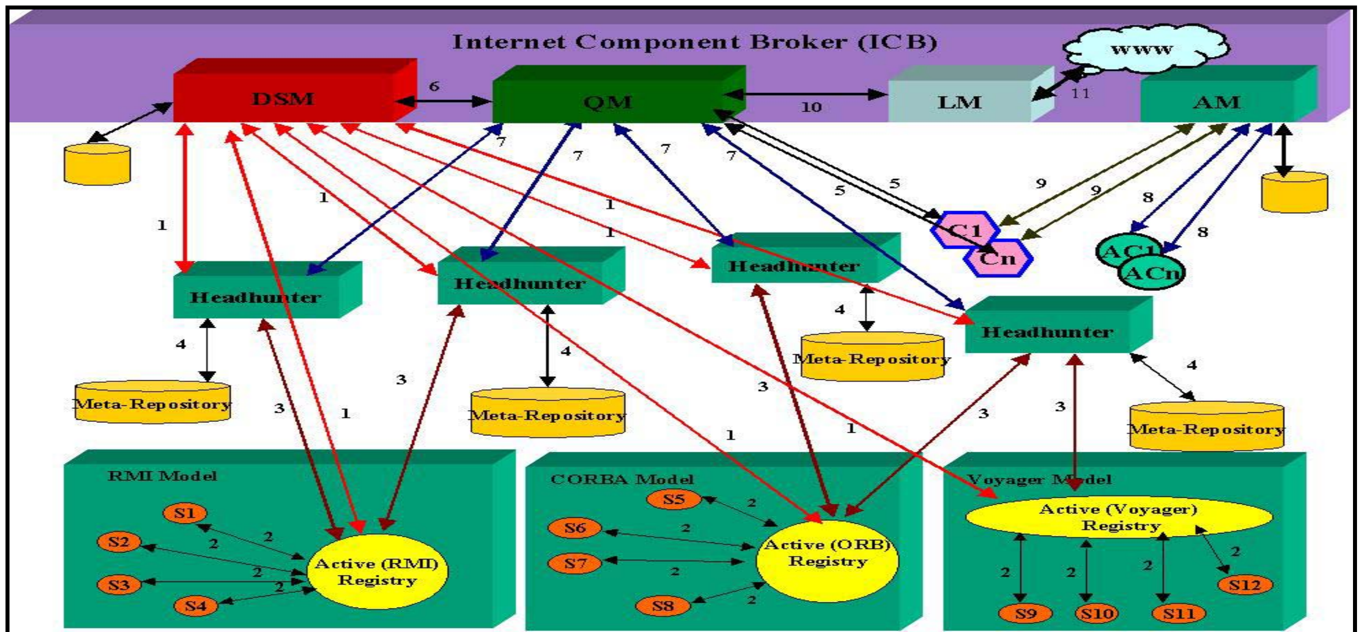

Figure 2: URDS Architecture

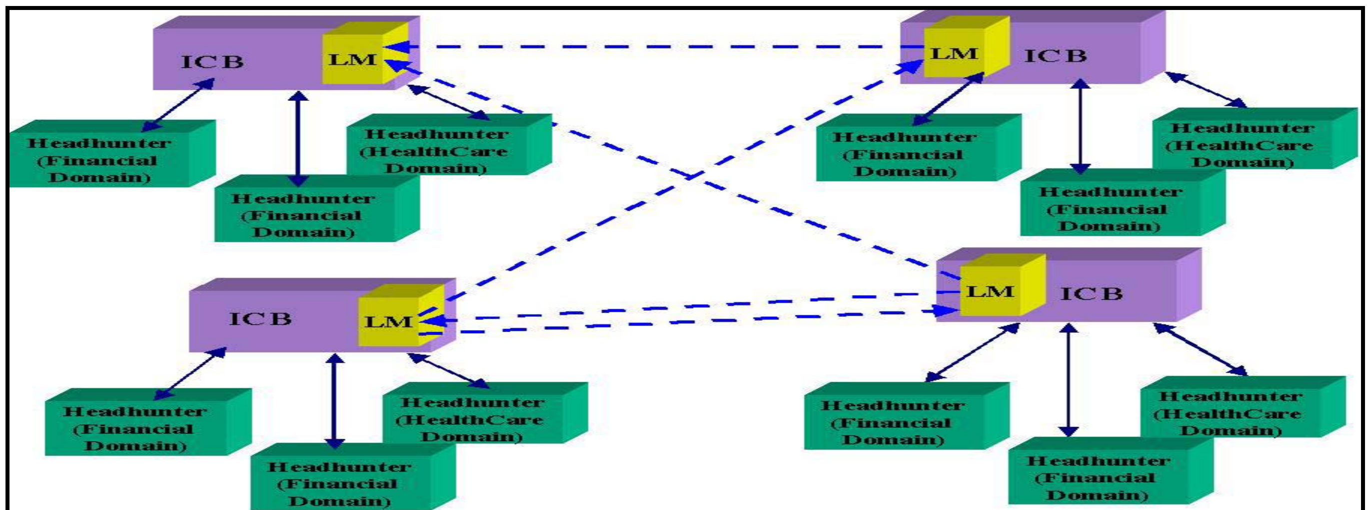

Figure 3: Federated Organization in URDS

3.2.1 Internet Component Broker (ICB)

The ICB acts as an all-pervasive component broker in the interconnected environment providing a platform for the discovery and seamless integration of disparate components. The ICB is not a single component but is a collection of services comprising of the Query Manager (QM), the Domain Security Manager (DSM), Adapter Manager (AM), and the Link Manager (LM). It is envisioned that there will be a fixed number of ICBs deployed at well-known locations hosted by corporations or organizations supporting this initiative.

The functionality of the ICB is similar to that of an Object Request Broker. However, the ICB has certain key features that are unique. It provides component mappings and component model adapters. The ICB, in conjunction with headhunters, provides the infrastructure necessary for scalable, reliable, and secure collaborative business using the interconnected infrastructure. The functionalities of the ICB are:

- Authenticate the users (Headhunters and Active Registries) in the system and enforce access control over the multicast address resources for a domain with the help of the Domain Security Manager (DSM).
- Attempt at matchmaking between service producers and consumers with the help of the Headhunters and Query Manager. ICBs may cooperate with each other in order to increase the search space for matchmaking. The cooperation techniques of ICBs are facilitated through the Link Manager (LM).
- Act as a mediator between two components adhering to different component models. The mediation capabilities of the ICB are facilitated through the Adapter Manager (AM).

Domain Security Manager (DSM)

The DSM handles secret key generation and distribution and enforces the group membership and access control to multicast resources through authentication and use of access control lists (ACL). The resources being guarded are the multicast addresses allocated to a particular *domain*. The DSM serves as an authorized third party, which maintains an inclusion list of Principals (headhunters or registries), corresponding to a domain. DSM has an associated repository (database) of valid principals, passwords, multicast address resources and domains. Every Headhunter or Active Registry is associated with a domain. The Active Registries associated with a domain have components registered with them, which belong to that domain. The Headhunter in turn detects Registries, which belong to the same domain as itself, and hence the service components detected by the headhunter will belong to a particular domain. The Principal (authenticated user), is allowed access only to the multicast address mapped to the domain with which it is associated. A Principal that wishes to participate in the discovery process contacts the DSM with its credentials (id, password, domain). The DSM authenticates the principal and checks its authorizations against the domain ACL. The DSM returns a secret key and a multicast address mapped to the corresponding domain to a valid principal. In case the principal is a Headhunter the DSM registers the contact information of the headhunter with itself. The QM to propagate queries uses this information.

Query Manager (QM)

The QM uses a natural language parser [7] to translate a service consumer's natural language-like query into an XML based query. The QM parses the XML based query to generate a structured query language statement and dispatches this query to the 'appropriate' Headhunters. The QM obtains the list of registered Headhunters from the DSM. The HH returns the list of matching service providers. The QM in conjunction with

the LM is also responsible for propagating the queries to other linked ICBs. The functions performed by the QM are:

- Parse a service consumer's natural language-like query and extract the keywords and phrases pertaining to various attributes of the components UniFrame specification.
- Extract the consumer-specified constraints, preferences and policies to be applied to the various attributes.
- Compose the extracted information into an XML based query.
- Translate the XML based query to a structured query language statement.
- Dispatch this structured query to all the headhunters associated with the domain on which the search is being performed and also forward the query to the Link Manager, which will propagate the query to other ICBs.
- The headhunters will query the Meta-Repository and return a list of components matching the search criteria to the QM.
- QM will wait for a specified time period for results to be returned from the headhunters/other ICBs before timing out.
- The client has the option to specify search-scoping policies to affect the time spent on the search process.

Link Manager (LM)

ICBs are linked to form a *Federation of Brokers* (see Figure 3) in order to allow for an effective utilization of the *distributed offer space.* ICBs propagate the search query issued by the Clients to other ICBs to which they are linked apart from the headhunters with which they are associated. The LM performs the functions of the ICB associated with establishing links and propagating the queries. *Links* represent paths for propagation of queries from a source ICB to a target ICB. The LM supports the following operations:

- *Register:* LMs register with each other to create unidirectional links from the Source LM to the Target LM. The registration information comprises of the location information of the LM.
- *Query:* The query operation is responsible for propagating the query from the source LM to the list of Target LMs with which the Source LM is registered.
- *Failure Detection:* This involves keeping track of LMs that may no longer be active due to failures. Periodically each LM sends a unicast message to all other LMs that are registered with it. LMs receiving the message maintain a cache of the pairs <*Sender LM address, Time-stamp of receipt*>. At regular time intervals the receiving LMs note the 'freshness' of the information they hold and purge the Sender's information, which they deem to be 'stale'. Staleness is determined by the time elapsed between the receipt of the LM address through the unicast communication and the current time.

- *Link Traversal Control:* The Link Traversal Control mechanism used in the LM is similar to that of CORBA Trader Services. The necessity for Link Traversal Control arises due to the nature of LM linkage, which allows arbitrary, directed graphs of LMs to be produced. This can introduce two problems: i) a single LM can be visited more than once, and ii) loops can occur. To ensure that a search does not enter into an infinite loop, a **hop count** is used to limit the depth of links to propagate a search. The hop count is decremented by one before propagating a query to other LMs. The search propagation terminates at the LM when the hop count reaches zero.

Adapter Manager (AM)

The AM serves as a registry/lookup service for clients seeking adapter components. The adapter components register with the AM and while doing so they indicate their specialization (i.e., which heterogeneous component models they can bridge efficiently). Clients contact the AM to search for adapter components matching their needs. The AM utilizes adapter technology, each adapter component providing translation capabilities for specific component architectures. The adapter components achieve interoperability using the principles of wrap and glue technology [8].

3.2.2 Headhunters

Another critical component of URDS is a headhunter. The headhunters perform the following tasks: a) Service Discovery: detect the presence of service providers (Exporters), b) register the functionality of these service providers, and c) return a list of service providers to the ICB that matches the requirements of the consumer (Importers) requests forwarded by the QM.

The service discovery process utilizes a search technique based on multicasting. Once deployed in the system, the headhunters periodically multicast their presence to a multicast group. The multicast group address is obtained from the DSM. The active registries, that also obtain a multicast group address from the DSM, listen for these multicast messages. The active registries maintain a cache of the pairs <*headhunter address, time-stamp of receipt*> and periodically send response messages to all the headhunters in their cache. The headhunter in turn maintains a cache of the pairs <*registry address, time-stamp of receipt*>. The Headhunter intermittently queries the Registries for the component information of service providers they contain. During the registration, the headhunter stores into the meta-repository all the details of the service providers, including the UniFrame specifications. The headhunter uses this information during matching. A component may be registered with multiple headhunters. The functionality of headhunters makes it necessary for them to communicate with Active Registries belonging to any model, implying that the cooperative aspect of headhunters be universal. The headhunters need to also address the issues of failures and security.

- ***Failure Detection:*** Failure detection involves keeping track of service exporters that may no longer be active in the system for various reasons. Headhunters achieve failure detection at the level of detecting failures of the active registries, which hold the service exporters. The headhunter keeps track of the time at which it

obtains registry location information from various active registries. At regular time intervals the headhunter notes the 'freshness' of the information it holds and purges the registry information, which it deems to be 'stale'. 'Fresh' or 'Stale' are determined based on the time elapsed between the receipt of the registry address through unicast communication and the current time. This process is based on the principle that if a registry is still active in the system, it will respond to the headhunter with its location information and thus have a recent timestamp. A registry which for whatever reason is unable to contact the headhunter with its information will hold a 'stale' timestamp and it will be assumed that all service exporter components held by this registry are no longer available for rendering their services.

- ***Multicast Security:*** This involves securing the multicast data transmission mechanism from security threats such as eavesdropping, and masquerading. The headhunter uses Secret Key Encryption to ensure security of transmitted data. The secret key used is a symmetric key wherein the sender and receiver use the same key for purposes of encryption and decryption.

3.2.3 Meta-Repository

The Meta-Repository is a data store that serves to hold service information of exporters adhering to different models. The service information stored by the Meta-repository consists of: a) Service type name, b) Details of its informal specification, and c) Zero or more QoS values for that service for each of the components. The implementation of a Meta-Repository is database oriented. A Meta-Repository is a *passive component,* i.e., a headhunter brings information to the meta-repository.

3.2.4 Active Registry

The native registries (e.g., RMI Registry or CORBA registry) are extended to have the following features:

- *Activeness:* The registries are modified to be able to listen to multicast messages from the headhunter and respond with their host IP Address.

- *Introspection Capabilities:* The registries are extended to not only keep a list of component URLs of those components registered with them but also their detailed UniFrame specifications. This is achieved by querying the components (using principles of introspection) to obtain the URL of their XML based specifications. The registries parse the specification and maintain the details in a memory resident table, which is returned to the headhunter upon request.

- *Failure Detection Of Headhunters:* Failure detection involves keeping track of headhunters, which may no longer be active in the system for reasons such as network or node failure. The active registries keep track of the time at which it obtains headhunter location information from various headhunters through multicast. At regular intervals the active registries note the 'freshness' of the headhunter information they hold and purge the headhunter information, which they deem to be 'stale'. 'Fresh' or 'stale' are determined based on the time elapsed

between the receipt of the headhunter address through multicast communication and the current time.

3.2.5 Service Exporter Components

Service Exporter Components are implemented in different models, e.g., Java RMI, CORBA, EJB, etc. The components are identified by their *Service Offers* comprising of service type name, b) informal UniFrame specification, and c) zero or more QoS values for that service. The component registers its interfaces with its local registry. The component interface contains a method, which returns the URL of its informal specification. The informal specification is stored as a XML file adhering to certain syntactic contracts to facilitate parsing. These service exporter components will be tailored for specific domains, such as Financial Services, and will adhere to the relevant standards in those domains.

3.2.6 Clients

Clients are Service Requesters searching for services matching certain functional and non-functional requirements.

4. An Example

Table 1 outlines the interactions between the URDS components in servicing a client query for assembling an account management system. The rows of the table are numbered corresponding to the flow of control shown in Figure 2. The result of this interaction will be an ensemble of components, which may be assembled into a complete system as described in [12].

Table 1: Interactions between URDS components

1	This indicates the interactions between the principals (Headhunters/Active registries) and the DSM.
	• The principals contact the DSM with their authentication credentials in order to obtain the secret key and multicast address for group communication (many to one interaction).
	`<name="headhunter1", password="secret1", domain="financial">`
	`<name="registry2", password="secret2", domain="financial">`
	• The DSM authenticates the principals and returns a secret key and multicast address to a valid principal (one to many interaction).
	`<secretkey = key.dat, multicast_address="224.2.2.2">`

2	This indicates the interactions between Service Exporter Components and active registries. • Service exporter components register with their respective registries (many to one interaction) -- `<id="intrepid.cs.iupui.edu/AccountServer">`

	• These registries in turn query these components for their UniFrame Specification (one to many interaction).
	`<introspect property = "uniFrameSpecURL">`
	• The components respond with the URL at which the specification is located (any to one interaction).
	`<url="C:\Account System\AccountServerSpec.xml">`

3	This indicates the interactions between Headhunters and Active Registries.
	• Headhunters periodically multicast their presence to a multicast group addresses (one to many interaction).
	`<headhunterlocation = phoenix.cs.iupui.edu/headhunter1>`
	• Active Registries, which are listening at this group address, respond to Headhunters' messages by passing their information to Headhunters (many to many interaction).
	`<registrylocation = magellan.cs.iupui.edu/registry2>`
	• Headhunters intermittently query the active registries to which they hold a reference for the information of all the components registered with them (one to many interaction). The active registries respond by passing the list of components registered with them and the detailed UniFrame specification of these components (many to many interaction).

4	This indicates the interactions between a Headhunter and a Meta-Repository. • Headhunters persist the component information obtain ed from the active registries onto the Meta-Repository (one to one interaction). • Headhunters query Meta-Repository to retrieve component information (one to one interaction). `<query="SELECT * FROM componentTable A, functionTable B WHERE (A.ID = B.ID) AND ((description LIKE%account%) OR (description LIKE %system%)) AND (end2endDelay<10) AND (availability>90)">` • Meta-Repository returns search results to headhunter (one to one interaction).
5	This indicates the interactions between the QM and clients. • Clients contact the QM and specify the functional and non-functional search criteria (many to one interaction).

<table>
<tr><td></td><td>• The natural language-like client query is as follows:
"Create an account management system that has end-to-end delay < 10 ms and availability > 90% preference maximum availability".
• Figure 4 shows the translated XML based query.
<Query>
<Description> Account System </Description>
<Domain> Financial </Domain>
<End2EndDelay constraint="<" >10 </End2EndDelay>
<Availability constraint = ">" preference ="max"> 90 </Availability>
</Query>
Figure 4: Processed XML query
• The QM returns the search results to the clients (one to many interaction).
< component 1: id="..", description="…", availability="…",…;
component 2: id="..", description="…", availability="…"…;
component 3: id="..", description="…", availability="…",…;></td></tr>
<tr><td>6</td><td>This indicates the interaction between the QM and DSM.
• QM contacts DSM for contact information of registered headhunters belonging to the domain of client query (one to one interaction).
• DSM responds with list of registered headhunters (one to one interaction).
<phoenix.cs.iupui.edu/headhunter1
magellan.cs.iupui.edu/headhunter2></td></tr>
<tr><td>7</td><td>This indicates the interactions between the QM and headhunters.
• The QM propagates Client's query to all headhunters registered with it, which fall in the domain of the Client's search request (one to many interaction).
• The headhunters respond to the QM query with search results matching the criteria (many to one interaction).</td></tr>
<tr><td>8</td><td>This indicates the interactions between adapter components and AM.
• Adapter components register with the AM, which is running at a well-known location (many to one interaction).</td></tr>
</table>

9	This shows the interactions between the clients and the AM. • Clients contact the AM at the well-known location at which it is running with requests for specific adapter components (many to one interaction). • The AM checks against its repository for matches and returns the results to the clients (one to many interaction).
10	This shows the interactions between QM and LM. • The QM propagates the query to the LM (one to one interaction). • LM returns search results to QM (one to one interaction).
11	This shows the interactions between the LM of one ICB and target LMs of other ICBs with which this LM is registered. • The LM propagates the search query issued by the QM to the target LMs (one to many interaction). • The source LM receives the result responses from these target LMs (many to one interaction).

5. Comparison between URDS and Other Resource Discovery Protocols

A brief comparison between URDS and other approaches is provided below.

- *Interoperability:* The other resource discovery protocols provide services for specific models and interoperations can be achieved only through proxies. URDS addresses the issue of non-uniformity by providing for discovery and coordination between components implemented using diverse models.
- *Network Usage:* Unlike other protocols, URDS clients and services do not participate in active discovery thus cutting down on the periodic communication required for the process of discovery. Instead, the active nature of the extended native registries allows the discovery process and removes the additional burden of developing ‘active’ components.
- *Query Processing and Matchmaking:* Unlike other approaches, which rely on Java-based or XML-based matching, the URDS supports a natural language-like query mechanism. This provides flexibility in formatting queries and during the matchmaking process.
- *Domain of Discovery:* In URDS the contextualization of the search space is logical and dictated by the industry specific markets. In other discovery protocols the

notion of “administrative scope” is associated with the topology of the network domain.

- *Security:* The URDS security model addresses many of the common threats, which may occur during the discovery process. SSDS is another service notable for its robust security model.
- *QoS:* UniFrame incorporates of the notion of QoS as applied to software components and integrates this aspect into the service specification and the matchmaking process.

6. Prototype and Experimentation

A preliminary prototype [14] of the URDS has been implemented using the J2EE version 1.4 software environment. The core architectural components (domain security manager, query manager, link manager, headhunters and active registries) have been implemented as Java-RMI based services.

The repositories (domain security manager’s repository and meta repository) have been implemented using Oracle version 8.0. The Web-based components (JSPs), which service client interactions, are placed in a Tomcat 4.0 Servlet/JSP container.

The unicast communication between the core architectural components is achieved through JRMP (Java Remote Method Protocol) and the multicast communication between the headhunters and the active registries is achieved through Multicast sockets based on UDP/IP. The database connections are established using the JDBC (Java Database Connectivity) APIs and the user interaction is achieved through a browser front-end using the HTTP protocol. The security infrastructure, of URDS, is implemented by the security and cryptography APIs that form a part of Java Cryptography Architecture and Java Cryptographic Extension frameworks.

Preliminary experiments were carried out on this prototype to observe the performance of URDS. The experimental setup consisted of Sun SPARC machines connected by an Ethernet. The experiments contained one ICB, one headhunter, and one active registry (enhanced version of Java RMI registry). A single client was used to issue query requests, which consisted of different QoS constraints. The measurements were averaged over one hundred trials. The following times were measured:

- Average Authentication Time: It is the average time taken by the domain security manager to authenticate a principal (i.e., headhunter and active registry).
- Average Query Service Time: It is the average time taken to service a query.
- Average Registry Discovery Time: It is the average time taken by a headhunter to discover an active registry.
- Average Component Information Retrieval Time: It is the average time taken by the headhunter to retrieve component information from an active registry.

These initial experiments showed a value of 690 ms for the average authentication time. The average query time and the registry discovery time showed a marginal increase with an increasing number of registered components; while the average component retrieval information time increased linearly with the number of components (as expected).

The current prototype is able to discover only Java-RMI components, thus making it homogeneous. Efforts are underway to make it heterogeneous, i.e., able to discover components created using other models (such as CORBA, .NET, etc.) also. The current prototype also does not include the federation aspect.

7. Conclusion

The paper has presented an architecture that facilitates the semi-automatic construction of a distributed system by providing for the dynamic discovery of heterogeneous components and selection of components meeting the necessary requirements, including desired levels of QoS. The URDS architecture addresses issues such as interoperability, QoS of software components, scalability, fault tolerance, security and network usage. Interoperability is achieved by discovering components developed in several different component models. The discovery mechanism uses multicasting to detect native registries/lookup services of various component models that are extended to possess 'active' and 'introspective' capabilities. The component specification captures their computational, functional, co-operational, auxiliary attributes and QoS metrics. Flexibility in query formatting is achieved by providing support for natural language-like client requests. As a scalability mechanism URDS is organized in a federated hierarchical structure. Failure tolerance is handled through periodic announcements by entities and through information caching. Security is provided through authentication of the principals involved, access control to multicast address resources, and encryption of data transmitted. URDS provides a directory based discovery service which is scalable secure and fault tolerant. Although, the current prototype does not address all the features of the URDS architecture, it has created a basis for validating the concepts behind URDS. Efforts are underway to extend the current prototype that will enable a validation of all the features presented in this paper.

References

[1] Box, D., et al., "Simple Object Access Protocol (SOAP) 1.1", W3C, May 2000, http://www.w3.org/TR/SOAP.

[2] Brahmnath, G., Raje, R. R., Olson, A. M., Auguston, M., Bryant, B. R., Burt, C. C., "A Quality of Service Catalog for Software Components," to appear in Proceedings of the 2002 Southeastern Software Engineering Conference, 2002.

[3] Bray, T., Paoli, J., Sperberg-McQueen, C. M. "Extensible Markup Language (XML) 1.0 (Second Edition)," W3C, October 2000, http: //www.w3c.org/xml.

[4] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., "Web Services Description Language (WSDL) 1.1," W3C, March 2001 http://www.w3.org/TR/wsdl.

[5] Czerwinski, S. E., Zhao, B. Y., Hodes, T. D., Joseph, A. D., Katz, R. H., “An Architecture for a Secure Service Discovery Service,” Proceedings of Mobicom '99, 1999. http://ninja.cs.berkeley.edu/dist/papers/sds-mobicom.pdf

[6] Guttman, E., “Service Location Protocol: Automatic Discovery of IP Network Services,” IEEE Internet Computing, vol. 3, no. 4, 1999, pp. 71-80.

[7] Lee, B.-S., and Bryant, Barrett R., “Automated Conversion from Requirements Documentation to an Object-Oriented Formal Specification Language,” Proceedings of SAC 2002, the ACM Symposium on Applied Computing, 2002, pp. 932-936.

[8] Luqi, Berzins, V., Ge, J., Shing, M., Auguston, M., Bryant, B. R., Kin, B. K., “DCAPS – Architecture for Distributed Computer Aided Prototyping System,” Proceedings of RSP 2001, the 12th Rapid Systems Prototyping Workshop, 2001, pp. 103-108.

[9] Ninja, “The Ninja Project,” http://ninja.cs.berkeley.edu, 2002.

[10] Object Management Group, “Trading Object Service Specification,” Object Management Group 2000. ftp://ftp.omg.org/pub/docs/formal/00-06-27.pdf.

[11] Raje, R. R., “UMM: Unified Meta-object Model for Open Distributed Systems”, Proceedings of ICA3PP 2000, 4th IEEE Int. Conf. Algorithms and Architecture for Parallel Processing", 2000, pp. 454-465.

[12] Raje, R., Auguston, M., Bryant, B. R., Olson, A., Burt, C., “A Unified Approach for the Integration of Distributed Heterogeneous Software Components”, Proceedings of the Monterey Workshop on Engineering Automation for Software Intensive System Integration, 2001, pp. 109-119.

[13] Raje, R., Auguston, M., Bryant, B. R., Olson, A., Burt, C., “A Quality of Service-based Framework for Creating Distributed Heterogeneous Software Components”, Technical Report, Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 2002.

[14] Siram, N. N., “An Architecture for the UniFrame Resource Discovery Service”, MS Thesis, Indiana University Purdue University Indianapolis, Spring 2002.

[15] Sun Microsystems, “Jini Architecture Specification, Version 1.2,” Sun Microsystems, December 2001, http://www.sun.com/jini/.

[16] uddi.org, “UDDI Technical White Paper”, September 2000, http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf