A Document as a Web Service: Two Complementary Frameworks*

Shahram Ghandeharizadeh, Frank Sommers, Kuntal Joisher, Esam Alwagait

Department of Computer Science
University of Southern California
Los Angeles, CA 90089, USA

Abstract. An electronic document may provide more information than its paper-based counter part. For example, in addition to text and images, a scientific document may consist of either the raw data or software that substantiates its hypothesis, ontology of its terms, a detailed comparison with its related work, etc. In this study, we explore: a) an environment that represents a document as a Web Service, and b) two alternative frameworks that facilitate discovery and retrieval of documents, namely, Jini and UDDI. We describe how each framework supports documents that reside on a cluster of nodes. These nodes might be dispersed across the Internet or in geographical proximity using an Intranet setting. In addition to detailing discovery and retrieval of documents, we describe how each framework supports data availability in the presence of load imbalance and node failures. One finding of this study is that these two frameworks are complementary and can be used in conjunction with one another.

1 Introduction

With advances in hardware, software, and computing infrastructure, an electronic document has evolved beyond text and images to include audio, video, software (behavior), etc. It is a self-contained information unit that might reference other documents. As an example consider a scientific manuscript that describes the working characteristics of a magnetic disk drive. In addition to a textual description and images, this document might include the following: a) animations that visualize the mechanical workings of a disk, introducing the reader to terms such as disk seek and rotational delay, b) analytical models that describe and quantify the time required for a disk to perform a seek operation, c) a simulation model (software) that embodies these analytical models, d) empirical data from real magnetic disks, how well the simulation model estimates the seek time of these devices, and software that accumulated the key physical disk parameters for use in the simulation model. This document and its accompanying software might be referenced by other documents that build upon the physical characteristics of magnetic disk drives, e.g., documents in the area of continuous media servers, physical database design, etc.

A Web Service advocates the use of "software and data as a service" paradigm. It is beneficial to conceptualize a document as a Web Service for several reasons. First, as a Web Service, it encapsulates both data and behavior as a self-contained object. This object exposes its methods for use by either humans via a browser (or an Internet application), other documents, or both. Second, it ensures autonomy of documents so that one or more might either reside on a single server or cached (replicated) across many servers. As an autonomous object, a document might migrate from one node to another to prevent formation of hot spots and bottlenecks [16]. Third, it might be accompanied by a proxy object that analyzes the client environment to make intelligent choices. For example, it might install software to enable a user to view a document, e.g., installation of MPEG viewer for a video clip. As another example, the proxy might decide whether a method should execute local to a client or on a remote server. This decision is based on the client's available network bandwidth, data needs of a method, processing capability of the client machine, and the load

* This research has been supported in part by a grant from National Library of Medicine LM07061-1 and unrestricted cash gifts from Microsoft and BMC Software.

on remote servers. While it is unrealistic to expect a scientist to write these proxy objects each time they publish a document, it is possible to introduce templates (and wizards) that tailor proxy objects.

There are several off-the-shelf frameworks in support of a document management system that conceptualizes each document as a Web Service. These include Sun Microsystems' Jini [20], and Universal Description, Discovery and Integration (UDDI) framework. The UDDI framework is a working W3C standard authored by Ariba, IBM, and Microsoft. At the time of this writing, several implementations of UDDI is available from companies such as Microsoft and IBM. (Jini is available from Sun Microsystems.) In this study we describe how these two off-the-shelf frameworks support discovery and retrieval of documents (Web Services). Of course, one may use other frameworks. We focus on Jini and UDDI because they are emerging frameworks. At the same time, they provide complimentary concepts and offer a rich environment to implement a document management system.

The key difference between UDDI and Jini is as follows. While Jini's lookup service is based on a typed system that exchanges both data and behavior with a client, a UDDI registry exchanges only XML formatted data with a client software. This means that Jini is tailor-made for the Java programming language [1]. While UDDI is programming language neutral, the development of a document management system is greatly simplified (and expedited) using programming environments such as Microsoft's Visual Studio .NET (or IBM's Web Sphere).

The rest of this paper is organized as follows. In Section 2, we survey the two complementary frameworks. In addition to discovery and retrieval of documents, we describe how each framework supports availability of data in the presence of failures and scalability of the underlying cluster of nodes. Section 3 describes JUHF, a hybrid framework that utilizes Jini and UDDI. Section 4 offers brief conclusions and our immediate future research directions.

2 Two Frameworks

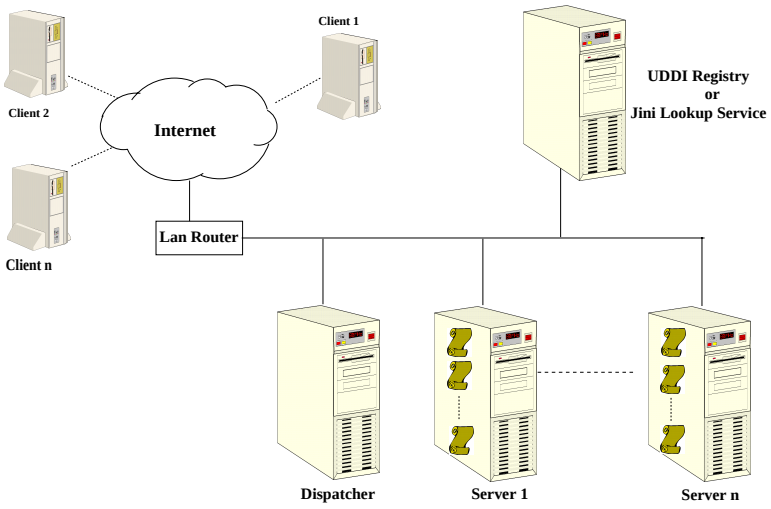

Fig. 1. An Intranet deployment.

There are many ways to setup a document management system. The focus of this paper is on two complementary off-the-shelf frameworks to manage the infrastructure. Each can be deployed for either the Internet or an Intranet deployment. In order to simplify discussion and without loss of generality, we assume an Intranet setting, see Figure 1.

[1] Indeed, it only works with Java class definitions.

In Section 2, we extend this discussion to the Internet. Our assumed Intranet environment consists of a cluster of nodes. Each document is represented as a Web Service that is assigned to a node. Once a Web Service is launched, it registers itself with a lookup service (registry). A dispatcher provides a user friendly interface to the lookup service for document discovery. Subsequent communication between the client and the node containing the referenced data is direct and does not involve the dispatcher. Note that the UDDI registry and the Jini lookup service can be configured in different ways, e.g., a configuration might assign these components to the same machine that hosts the documents.

In addition to discovery and retrieval of documents, this environment must support the following features: 1) availability of documents, and 2) scalability of the cluster to incorporate additional nodes. The first refers to continuous availability of documents in the presence of node failures. Conceptually, there are three failure types [2]: a) dispatcher node failure, b) lookup service (registry) node failure, and c) failure of nodes that contain documents, see Figure 1. Consider each in turn. In this paper, dispatcher node failures are handled as follows. The environment consists of a primary and secondary dispatcher. In the event of a primary dispatcher failure, the secondary dispatcher detects the missing heartbeat message of the primary dispatcher, and employs the IP address of the primary dispatcher to continue operation [7, 17, 15]. The availability of data in the presence of lookup service node failures is realized by maintaining multiple directory servers. The availability of Web Services is supported using a primary and secondary copy of each service that resides on two different nodes. The proxy objects for a Web Service are intelligent enough to switch from the primary copy to secondary copy when the primary copy does not respond. We assume that clients do not update documents (Web Services). In the presence of administrator update to the primary copy, the system must propagate updates to the different copies of a Web Service. We assume the lookup service maintains the location of both primary and secondary copies.

Scalability refers to the cluster's ability to service a higher number of user requests with additional nodes. In order to scale, the system load must be evenly distributed across the available nodes. This depends on the placement of Web Services across nodes. Our assumed framework includes components that migrate Web Services from one node to another in order to a) respond to changing workloads, and b) integrate new nodes by migrating Web Services to these nodes. Our focus is on mechanisms that enable the framework to migrate Web Services. The components that enable the framework to decide what to migrate from a source node to a destination node is an active research topic [22, 10, 21] that is beyond the focus of this study.

In the following, we describe each key functionality of our target document management system and explain how Sun's Jini and Microsoft's UDDI supports this functionality. We use the terminology provided by each framework. This presentation is a bit challenging because each framework uses the same term to refer to a different concept. For example, *discovery* with Jini refers to the process of a Web Service discovering the lookup service. With UDDI, the same term refers to the concept of a user discovering a document. We did not want to address this limitation by introducing yet another set of terminology. Instead, we present the terminology used by each framework in Table 1. The reader should refer to these tables when reading the description of each framework. The concept of a proxy object requires further clarification and is described below.

In addition to exposing its public interfaces, a document may provide a *proxy* object representing that document. This enables a client to download the proxy object and invoke the document's public interfaces using that proxy. In turn, the proxy initiates access to the remote document that it is representing. How that communication takes place, or what other resources are accessed, are encapsulated in the proxy, and remain hidden from the client making the method calls on the proxy. With Jini [20], a proxy object might consists of code and data. With Microsoft's .NET framework, a Web Service might be published by exposing its WSDL, i.e., XML description of the Web Service. One may employ

[2] This study ignores network link failures.

Concept	Jini	UDDI
A directory or index of available services.	Lookup service	registry
Services and clients discovering available lookup services. Jini framework supports both Unicast and Multicast.	Discover	N/A
A service registering with the lookup service	Join	Publish
The downloadable component that enables a client to communicate with the Web Service.	Proxy	WSDL
A Web Service removing its entry from the lookup service (UDDI registry)	de-register	delete
The period of time that a service joins the network and is available to process requests. Before the time expires, the service may re-negotiate its lease. If the time expires, its entry in the lookup service is removed automatically.	Lease	N/A
The process of a client querying the UDDI registry for a service using a search criteria provided by the client.	Inquire/ Lookup	Discovery

Table 1. Terminology used by Jini and UDDI frameworks.

tools to convert this description into code and data (dynamic link libraries, Java class definitions, etc.,) for download and use by the client.

In the rest of this section we survey Jini and UDDI from a functionality perspective. We focus on the following key functionalities: 1) Registration of a Web Service with a directory, 2) Discovery and retrieval of documents, 3)Migration of Web Services, 4) Node failure, and 5) Internet deployment. We describe each in turn

Registration of a Web Service with Directories: The **Jini** framework consists of one or more lookup services. A lookup service maintains a flat collection of service items. Each item represents an instance of a Web Service, i.e., a document. The item contains: a) the service proxy that clients use to access the service, and b) an extensible collection of attributes that describe the service or provide secondary interfaces to the service. The later is important because it can maintain the primary and secondary copies of a Web Service. In addition, the implementer may provide sufficient descriptive attributes to allow clients to identify relevant documents.

A Web Service discovers the lookup services using three protocols: Unicast, Multicast request, and Multicast announcement. With a Multicast announcement, the lookup service announces its availability. In response, the Web Services register with the lookup service. With Multicast request, a Web Service requests the available lookup services to respond. Once a lookup service responds, the Web Service registers itself with each lookup service. The Unicast discovery protocol consists of the following two steps: first, the lookup service listens for incoming connections and, when a connection is made by a Web Service, decodes the request. If the request is correct, it responds with a marshalled object that implements the net.jini.core.lookup.ServiceRegistrar interface. Second, a Web Service that wishes to contact a particular lookup service uses the known host and port information to establish a connection to that service. It sends a discovery request and listens for a marshalled response object. Third, when the Web Service receives the marshalled object, it employs this object to *join* the Jini lookup service. The details of this step are as follows. A Web Service might be implemented to register with many lookup services. If a lookup service fails to respond, the implementer may choose to either retry or give up. A Web Service registers with, termed *joins*, a lookup service by providing the following: 1) a Service object which is also termed a proxy, and 2) a set of Service attributes where one or more might be an object. A service registers with a lookup service for a fixed amount of time, termed *lease* period.

It must periodically renew its lease with the lookup service. Otherwise, once its lease period expires, it is removed from the lookup service.

The **UDDI** framework consists of a registry containing the Web Service's descriptive attributes. A user registers a Web Service by publishing it into the UDDI registry. The World-Wide-Web consortium (W3C) provides an API for publishing and finding Web Services using UDDI. The publish API used for registration consists of four save_xx functions and four delete_xx functions. There is one for each key UDDI data structures, namely:

- businessEntity: supports "yellow pages" taxonomies so that searches can be performed to locate documents that fall in a certain category. For example, the field of computer science can be broken into multiple disciplines such as: Artificial Intelligence, Databases, Operating Systems, etc. Each discipline can be further subdivided. For example Databases can be divided into continuous media servers, data warehouses, parallel database management systems, etc. Each of these can be further divided into finer-grain disciplines. The businessEntity structure serves as the top-level information manager for all of the information about a particular set of information related to a group of documents.
- businessService: is a descriptive container used to group a collection of related Web Services related to either a process or a category of services. For example, one may specify a businessService for a group of documents related to a specific topic, e.g., data mining, etc.
- bindingTemplate: consists of technical information about a service entry point, e.g., URLs, and construction specifications. It provides technical data to enable a client to both discover and use a Web Service. It generally stores references to tModels.
- tModel: is keyed metadata, i.e., data about data. Its primary role is to represent a technical specification. It provides a reference system based on abstraction. There are two conventions for using tModels: a) as sources for determining compatibility, and b) as keyed namespace references. The tModel specification for a Web Service is the WSDL file. It consists of related information that facilitates communication between a client and a Web Service. Visual Studio .NET automates the authoring of this file.

Once authorized, an individual party may register any number of information sets, modify previously published information, and delete complete structures.

Similar to Jini, an environment may consist of multiple UDDI registries. It is the responsibility of the implementer to specify how a Web Service discovers a UDDI registry. Of course, the implementer may use the mechanisms described by the Jini framework. There is no concept of a lease with UDDI registry.

One difference between UDDI's WSDL and Jini's proxy object is as follows: A proxy object contains both data and code. This enables the programmer to implement behavior once and distribute to as many clients as possible. UDDI's WSDL provides data and it is the responsibility of the client software to implement behavior around this data. In Section 2, we see how these two concepts complement one another in the context of data availability.

Discovery and Retrieval of Relevant Documents: The Inquiry API provided by UDDI consists of two types of calls that let a program quickly locate documents. First, find_xx enables a client to pose a variety of search criteria on the registration data. Once a Web Service is identified as relevant, the get_xx APIs are used to retrieve a particular structure (businessEntity, businessService, bindingTemplate, tModel) from the UDDI registry.

While Jini does not provide an Inquiry API similar to UDDI, it is possible for an implementer to design and implement functionality similar to UDDI's Inquiry API using the lookup service and its extensible attributes. However, we believe this is not an appropriate use of Jini's lookup services. It is more appropriate to use UDDI registry for the discovery of relevant documents.

Web Service Migration: In order to respond to the changing workloads, the cluster may decide to migrate a Web Service from one node to another in order to distribute system load evenly across the available nodes. This is

accomplished by a Web Service de-registering itself with the lookup services (UDDI registries), migrating to a new node, and re-registering with the lookup services (UDDI registries). This raises two challenges. First, this invalidates the entry in a lookup service (registry). Second, with Jini, the proxy obtained by the client will point to an invalid location. With UDDI, the WSDL contained by a client will be invalid for communication. We start by describing how Jini's framework addresses these two challenges.

With Jini, it is the implementer's responsibility to author the proxy object for a Web Service in a manner that: a) throws an exception when one of its method invocations fails, b) contact the lookup service to obtain a new location (or proxy) for the object, and c) re-invoke the failed method. The proxy provides this flexibility because it may include software in support of this behavior.

With UDDI, the client software must detect that its bindingTemplate references a stale URL. In particular, a client (or a dispatcher) caches the bindingTemplate information of a Web Service from the registry upon its first reference to the Web Service. Its subsequent references are resolved from its cache and do not involve the UDDI registry. UDDI provides a retry on failure strategy where, once a client encounters a failure, it invokes a refreshFromCache() method that queries the UDDI registry to obtain the correct bindingTemplate (and cache it for subsequent use). It is the responsibility of the implementer to author the client software to detect stale bindingTemplates and obtain the new ones.

Node Failures: With a node failure, one or more Web Services may disappear. To continue operation the client software must locate and use the secondary copy of the Web Service to process requests. With Jini, there are several ways to accomplishing this functionality. One approach is for the proxy to cache the location of secondary copies and use these copies once it encounters an exception (either a timeout or server not found exception). An alternative approach would require the proxy to contact the lookup service to retrieve the URL of a secondary copy. This would require the use of extensible attributes for an item in the Jini lookup service.

It is natural for the UDDI registry to contain the URL of secondary copies of a Web Service. Similar to Jini, it is the responsibility of the client software to detect a failure (timeout or server not found exception) and access the UDDI registry for alternative URLs for the referenced Web Service.

Internet Deployment: The Internet deployment impacts Jini and UDDI frameworks as follows. First, one must be careful with the use of Multicast messages and its impact on the network traffic in an Internet setting. With a Jini deployment, it is more appropriate to use a Unicast protocol to facilitate discovery of lookup services. Second, it might be more appropriate to empower users to publish their documents by registering them with a UDDI registry (Jini lookup service). The document (Web Service) might reside on the server that is owned by the document publisher. It might be strategically cached by one or more servers to expedite access to the Web Service. A number of studies have investigated the impact of hosting data in close proximity to clients [23, 1]. These studies are orthogonal and can be applied to both Jini and UDDI.

3 Jini-UDDI Hybrid Framework (JUHF)

In Section 2, we surveyed Jini and UDDI as two complementary off-the-shelf frameworks for discovery and retrieval of documents in a distributed document management system. While we described each framework independent of one another, we believe these two frameworks provide complementary concepts. This section describes a hybrid framework that includes both frameworks. This framework employs UDDI to enable clients to discover documents. It utilizes Jini lookup servers to maintain the location of the UDDI registries and documents Web Services. This means that a UDDI registry itself is a Web Service registered with Jini lookup service. Proxy objects obtained from Jini lookup service

facilitate direct communication between the clients accessing UDDI registry and individual documents, and enables invocation of methods implemented by the document object.

3.1 JUHF Infrastructure

As discussed briefly above, the hybrid framework includes one or many Jini lookup services, with multiple UDDI registries acting as services and registering with Jini Lookup service. A document, represented as a service, registers with both UDDI Registry and Jini Lookup service. With multiple Jini lookup services, there is no need to modify the existing framework because the current specifications requires services to discover Lookup Services and register with them simultaneously. Of course, with a Unicast protocol, the identity of the lookup service must be provided to each Web Service.

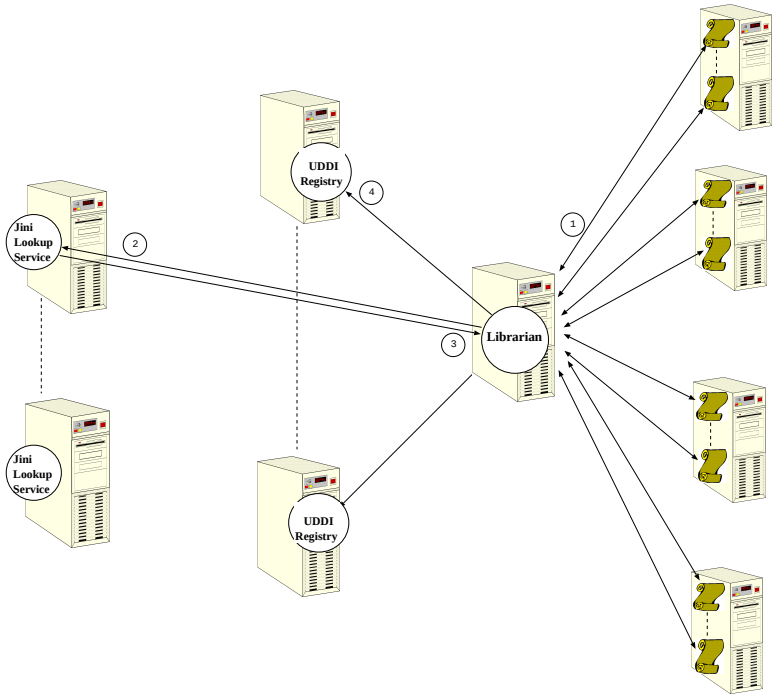

Fig. 2. A librarian process populates UDDI registries.

To facilitate discovery of documents, a document must register its descriptive information with UDDI registries. JUHF employs a ”librarian” service that is triggered every time a new document object is created(Step 1 in Figure 2). The ”librarian” registers the descriptive information of the document with the available UDDI registries. To obtain the locations of UDDI registries, librarian daemon queries the Jini lookup service for the available UDDI registry services (Steps 2 and 3) and contacts them to register the documents (Step 4). Figure 2 shows the librarian interaction with both document objects and UDDI registries.

In order to employ UDDI, we developed a mapping between UDDI concepts and Document Management concepts, see Table 2. Note that UDDI concepts are not designed to be extensible. It is specifically targeted towards businesses and a yellow page of their Web Services.

UDDI	Document Management	Comments
Business Entity	Document title	
BusinessService	Methods implemented in Document object, e.g., getPDF()	
Discovery URL	Location of the Document Object	In UDDI, the default discoveryURL is assigned by the UDDI registry and is used to retrieve the XML document for the businessEntity and everything contained within it. Additional discoveryURLs are assigned by the submitter, and provide links to external content that provides information about the businessEntity. In JUHF, we provide two discoveryURLs, one for Jini Jini lookup services and the other for the location of the document Web Service.
contact	Author	The role of the *Contact* in the business field and *Author* in the document management are identical.

Table 2. Mapping between UDDI concepts and those of a document management system.

3.2 Access Scenario

With JUHF, a client employs the following steps to discover and reference a document, see Figure 3. First the client contacts Jini lookup services and locates a UDDI registry, Step 1 and 2 in Figure 3. Next, the client contacts UDDI registry, submits and retrieves the identity of relevant documents. Subsequently, the client contacts Jini lookup services with the document identity and downloads the documents' proxy, Steps 5 and 6 of Figure 3. Finally, the client uses the proxy to remotely invoke methods on the objects, and obtain results, Steps 7 and 8 of Figure 3.

3.3 Handling Failure in JUHF

JUHF is built using two off-the-shelf frameworks, namely UDDI and Jini. These frameworks' specifications provide a mechanism for handling failure, namely, mirroring of UDDI registries and Jini lookup services. The UDDI specification [5, 4] requires registries to communicate in the background to exchange entries with each other to propagate updates. This means if a service joins a registry, it will automatically join other registries. The same applies to Jini lookup services.

3.4 JUHF Implementation and Deployment

We implement and deploy this hybrid environment for a collection of scientific papers at the USC database laboratory (http://dblab.usc.edu/). We implemented JUHF in a small environment with one UDDI registry (WASP UDDI provided by Systinet co.) and one Jini lookup service (Reggie lookup service implementation by Sun Microsystems) with 50 Document objects representing the scientific papers published by the USC database laboratory. We implemented a simple client, that enables the user to type in keywords and use them to query the UDDI registries. We used Java XML package provided by Sun Microsystems (http://java.sun.com/xml) to facilitate the communication between our client and the UDDI Registry. When the user issues a query, the client contacts the UDDI registry, obtains results and displays them for the user. When the user chooses one of the results, the interface obtains the Proxy object of that document from the Jini lookup service and display the methods provided by it, allowing the user to invoke any method by simply clicking on it.

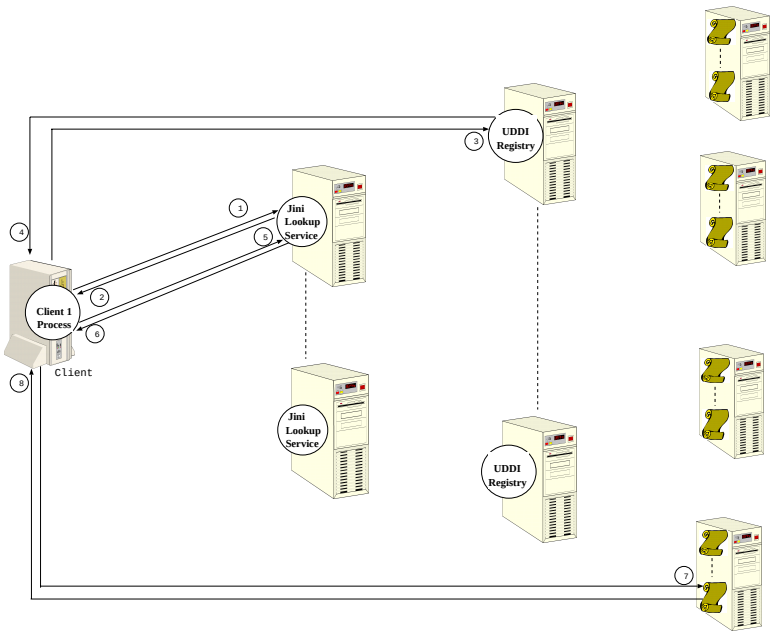

Fig. 3. Client communicating with components of a JUHF deployment.

4 Conclusion and Future Research Directions

In this paper, we described two complementary, off-the-shelf frameworks for discovery and retrieval of documents in a distributed document management system. While Section 2 describes each framework independent of one another, Section 3 describes a hybrid environment that utilizes the complementary concepts provided by each framework. This environment employs UDDI to enable clients to discover documents. It utilizes Jini lookup servers to maintain the location of the UDDI registries and Web Services. This means that a UDDI registry is a Web Service. Proxy objects facilitate direct communication between the clients accessing a UDDI registry and individual documents. We have a functioning prototype of this environment populated with scientific papers at the USC database laboratory (http://dblab.usc.edu/).

We are pursuing two immediate research directions. First, the UDDI registry is designed as a yellow-pages service for businesses. While we populated its schema by mapping UDDI's conceptual design to that of a document management system, it would be more appropriate to develop a tailor-made registry to facilitate document discovery. This registry would be a component similar to UDDI and use the existing standards specified for a document management system, e.g., the W3C standard on Document Object Model, DOM [2, 11]. It would provide functionality such as free-text search on a title of a document. For example, with UDDI, given a paper entitled "Continuous Display Using Multi-Zone Disk Drives", a query that searches using a keyword other than "Continuous" (or its subset, e.g., "Cont") would not retrieve this document.

Second, we are exploring extensions that enable a client program to launch the appropriate viewer for a component of a document. This viewer might either be general purpose, e.g., Adobe's acrobat reader, or specific to that document, e.g., a viewer for MRI scans. The client would launch acrobat viewer when the user invokes the *GetPDF* method of a document. Similarly, the client would employ Java's virtual machine to execute sample Java code published with a document. These viewers might be standardized by a community, e.g., a rat brain atlas viewer for neuroscientists. In this case, it would be more appropriate to maintain one copy of these standards (instead of replicating them with each document). We are currently supported by NIH to further investigate this issue for the Neuroscience community.

References

1. Y. Amir, A. Peterson, and D. Shaw. Seamlessly Selecting the Best Copy from Internet-Wide Replicated Web Servers. In *International Symposium on Distributed Computing*, 1998.
2. V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacbos, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. *Document Object Model Level 1*. World Wide Web Consortium, 1998.
3. N Borenstein and N Freed. *RFC 1341: MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies*. Bellcore/Innosoft, 1992.
4. UDDI Community. *UDDI Version 2.0 Data Structure Specification*. UDDI Organization, June 2001.
5. UDDI Community. *UDDI Version 2.0 Programmer's API Specification*. UDDI Organization, June 2001.
6. Microsoft Corporation. *Active Server Pages Guide*. Microsoft Corporation, 2001.
7. O. P. Damani, P. E. Chung, Y. Huang, C. Kintala, and Y. Wang. ONE-IP: Techniques for Hosting a Service on a Cluster of Machines. *Computer Networks and ISDN Systems*, 29(8–13):1019–1027, 1997.
8. L Masinter E Nebel. *RFC 1867: Form-based File Upload in HTML*. Xerox Corporation, 1995.
9. R Fielding, J Gettys, J Mogul, H Frystyk, L Msinter, P Leach, and T Berners-Lee. *RFC 2616: Hyptertext Transfer Protocol – HTTP/1.1*. W3C/MIT, 1999.
10. S. Ghandeharizadeh, S. Gao, C. Gahagan, and R. Krauss. An On-Line Reorganization Framework for Embedded SAN File Systems. In *Submitted for publication*, 2001.
11. A. Le Hors, P. Le Hegaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. *Document Object Model Level 2 Core*. World Wide Web Consortium, 2000.
12. Java Community Process. *JSR-000053 Java Servlet 2.3 and JavaServer Pages 1.2 Specifications*, 2001.
13. D.R.T Robinson E*Trade UK Ltd. K A Coar, IBM Corporation. *The Common Gateway Interface, Version 1.1*. W3C/MIT, 1999.
14. Jun Lang and David B. Stewart. A study of the applicability of existing exception-handling techniques to component-base real-time software technology. *ACM Transactions on Programming Languages and Systems*, 20(2):274–301, March 1998.
15. T. Schroeder, S. Goddard, and B. Ramamurthy. Scalable Web Sever Clustering Technologies. *IEEE Network*, pages 38–45, May 2000.
16. F. Sommers, S. Ghandeharizadeh, and S. Gao. Cluster-based computing with active, persistent objects on the web. In *IEEE 3rd International Conference on Cluster Computing*, October 2001.
17. W. Vogels, D. Dimitriu, K. Birman, R. Gamache, R. Short, J. Vert, J. Barrera, and J. Gray. The Design and Architecture of the Microsoft Cluster Service. In *IEEE FTCS*, 1998.
18. W3C/MIT. *HTML 4.0 Specification*, 1997.
19. W3C/MIT. *Simple Object Access Protocol (SOAP).*, May 2000.
20. Jim Waldo et al. *The Jini Specification*. Addison Wesley, 1999.
21. G. Weikum. The Web in 2010: Challenges and Opportunities for Database Research. In *Informatics*, pages 1–23, 2001.
22. G. Weikum, C. Hasse, A. Moenkeberg, and P. Zabback. The COMFORT automatic tuning project, invited project review. *Information Systems*, 19(5):381–432, 1994.
23. E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee. Application Layer Anycasting: A Server Selection Architecture and Use in a Replicated Web Service. *ACM/IEEE Transactions on Networking*, 8(4):455–466, 2000.