Implementation of Online Book Searching Using Web Services

San San Nwe, Nay Zar Chi Htoo
University of Computer Studies, Yangon, Myanmar
sansannwe27@gmail.com, htoo.nayzarchi@gmail.com;

Abstract

Universal Description, Discovery and Integration (UDDI) is a platform-independent, Extensible Makeup Language (XML)-based registry for businesses worldwide to list themselves on the Internet [2], [5]. UDDI enables businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet [2]. It is designed to be interrogated by Simple Object Access Protocol (SOAP) messages and to provide access to Web Services Description Language (WSDL) documents describing the protocol bindings and message formats required to interact with the Web Services listed in its directory [6]. UDDI is also a publicly accessible set of implementations of the specification that allow businesses to register information about the Web Services they offer so that other businesses can find them[1], [6]. UDDI registries are used to promote and discover these distributed Web services. This system describes the capabilities that these registries add to the World Wide Web with the demonstration of a client-server application. These shows how to run book add service and book search service by a Web Service Provider. And then, a UDDI Registry hosts searchable book entries. A Book Add Interface contacts the Book Add Service. A Web Server consumes Book Search Services and provides Book Search Web Interface to the Web Clients.

1. Introduction

The Universal Description, Discovery and Integration (UDDI) specifications define a way to publish and discover information about Web services. The term "Web service" describes specific business functionality exposed by a company, usually through an Internet connection, for the purpose of providing a way for another company or software program to use the service [1], [4].

Web services are becoming the programmatic backbone for electronic commerce [2]. For example, one company calls another's service to send a purchase order directly via an Internet connection. At first glance, it would seem simple to manage the process of Web service *discovery*. After all, if a known business partner has a known electronic commerce gateway, what's left to discover? The tacit assumption, however, is that all of the information is already known. When you want to find out which business partners have which services, the ability to discover the answers can quickly become difficult. One option is to call each partner on the phone, and then try to find the right person to talk with. For a business that is exposing Web services, having to staff enough highly technical people to satisfy random discovery demand is difficult to justify [2].

Another way to solve this problem is through an approach that uses a Web services description file on each company's Web site. After all, Web crawlers work by accessing a registered URL and are able to discover and index text found on nests of Web pages. The "robots.txt" approach, however, is dependent on the ability for a crawler to locate each Web site and the location of the service description file on that Web site. This distributed approach is potentially scalable but lacks a mechanism to insure consistency in service description formats and for the easy tracking of changes as they occur [5].

UDDI takes an approach that relies upon a distributed registry of businesses and their service descriptions implemented in a common XML format [4].

2. Background

The number of ways that companies are using the World Wide Web varies considerably. Many companies are starting to define ways to allow their internal applications to interact with the business systems at other companies using the emerging Web infrastructure. Left alone, each company invents a unique approach based on the experiences of designers, available technologies, and project budgets. The proliferation of integration approaches and unique solutions have spawned an entire sub-industry focused on bridging incompatible service layers within and across company boundaries [3].

Recent work within the W3C starts to raise hopes that Extensible Markup Language (XML) will play a role in simplifying the exchange of business data between companies [10]. Further, collaboration between computer industry giants and small companies alike have outlined a framework called SOAP that allows one program to invoke service interfaces across the Internet, without the need to share a common programming language or distributed object infrastructure. All of this is good news for companies feeling the cost pressures associated with electronic commerce because the foundations for common interoperability standards are being laid. Because of these foundation technologies and emerging standards, some of the

intractable problems of the past are becoming easier to approach [10].

From XML and SOAP, one can observe that the integration and interoperability problem has been simplified in layers. XML provides a cross-platform approach to data encoding and formatting [10]. SOAP, which is built on XML, defines a simple way to package information for exchange across system boundaries. SOAP bindings for HTTP are built on this packaging protocol and define a way to make remote procedure calls between systems in a manner that is independent of the programming language or operating system choices made by individual companies. Prior approaches involved complex distributed object standards or technology bridging software. Neither of these approaches has proven to be cost effective in the long run. Using XML and SOAP, this cross-language, cross-platform approach simplifies the problem of making systems at two companies compatible with each other [2], [3].

Even when one considers XML and SOAP, though, there are still vast gaps through which any two companies can fall in implementing a communications infrastructure. Clearly, there is more work to do to achieve this goal. The UDDI specifications borrow the lesson learned from XML and SOAP to define a next-layer-up that lets two companies share a way to query each other's capabilities and to describe their own capabilities.
The following diagram depicts this layered view:

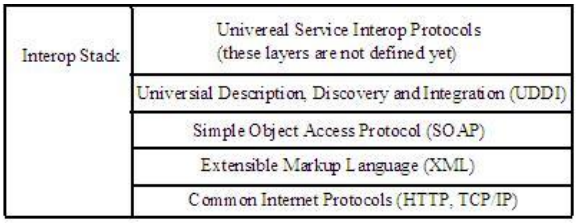


Figure 1. UDDI is a "next layer" in an emerging *stack e*nabling rich Web services. UDDI uses standards-based technologies such as TCP/IP, HTTP, XML and SOAP to create a uniform service description format and service discovery protocol.

3. UDDI and SOA

3.1. UDDI Business Registrations

The core component of the UDDI is the UDDI business registration, an XML file used to describe a business entity and its Web services. Conceptually, the information provided in a UDDI business registration consists of three components: "white pages" including address, contact, and known identifiers; "yellow pages" including industrial categorizations based on standard taxonomies; and "green pages", the technical information about services that are exposed by the business [1].

3.2. UDDI Usage

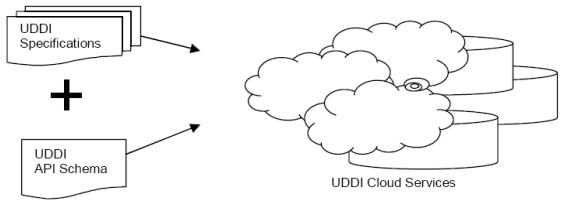
UDDI includes the shared operation of a business registry on the Web. For the most part, programs and programmers use the UDDI Business Registry to locate information about services and, in the case of programmers, to prepare systems that are compatible with advertised Web services or to describe their own Web services for others to call. The UDDI Business Registry can be used at a business level to check whether a given partner has particular Web service interfaces, to find companies in a given industry with a given type of service, and to locate information about how a partner or intended partner has exposed a Web service in order to learn the technical details required to interact with that service.

After reading this paper, the reader will have a clearer understanding of the capabilities defined in the UDDI specifications and have a clearer understanding of the role of Web service registries that implement these specifications.

3.3. UDDI as a Discovery Layer

The Universal Description, Discovery and Integration (UDDI) specification describes a conceptual cloud of Web services and a programmatic interface that define a simple framework for describing any kind of Web service. The specification consists of several related documents and an XML schema that defines a SOAP-based programming protocol for registering and discovering Web services.

The following diagram shows the relationship between the specifications, the XML schema and the UDDI business registry cloud that provides "register once, published everywhere" access to information about Web services.



\
Figure 2. UDDI Cloud Services

Using the UDDI discovery services, businesses individually register information about the Web services that they expose for use by other businesses. This information can be added to the UDDI business registry either via a Web site or by using tools that make use of the programmatic service interfaces described in the UDDI Programmer's API Specification. The UDDI business registry is a logically centralized, physically distributed service with multiple root nodes that replicate data with each other on a regular basis. Once a business registers

with a single instance of the business registry service, the data is automatically shared with other UDDI root nodes and becomes freely available to anyone who needs to discover what Web services are exposed by a given business.

3.4. SOAP and API

The Simple Object Access Protocol (SOAP) is a W3C draft note describing a way to use XML and HTTP to create an information delivery and remote procedure mechanisms. In its current state, the draft note describes a specification that is useful for describing a Web service. The companies that collaborated on UDDI decided to base the UDDI APIs on this SOAP specification. The specifics of how SOAP and XML are used by UDDI registry *Operators* are defined in the appendices in the API specification itself.

All of the API calls defined by the UDDI Programmer's API Specification behave synchronously [7].

3.5. Publication API

The Publication API consists of four *save_xx* functions and four delete_xx functions, one each for the four key UDDI data structures (businessEntity, businessService, bindingTemplate, tModel). Once authorized, an individual party can register any number of businessEntity or tModel information sets, and can alter information previously published. The API design model is simple – changes to specific related information can be made and new information be saved using save. Complete structure deletion is accommodated by the delete calls [8], [9].

4. System Architecture

The system contains a Web Service Provider, a UDDI registry and a Web Server. A Web Service Provider runs the Book Add Service and the Book Search Service. A UDDI Registry hosts the searchable book entries. A Web Server consumes the Book Search Services and provides the Book Search Web Interface to Web Clients. A Book Add Interface contacts the Book Add Service.

4.1. Components

The system accompanying this paper mainly consists of four components. They are:

1. UDDI Registry which storing a database of content listings submitted by providers.
2. Providers who provide actual web services and register their services on UDDI.
3. Upload Client which uploads contents from providers to UDDI and add new data to providers and UDDI.
4. Consumer that is a web client consuming the services of the providers with the help of UDDI.

4.2. UDDI Registry

UDDI Registry works by accepting entries from web services. In this program, web services are book providers registering available books on UDDI. When clients browse for books, UDDI looks for entries matching search criteria and points the clients to appropriate services.

4.3. Providers

Providers hold the actual book data. Stored Books have their respective attributes and when books are registered on UDDI, these attributes are reformatted ad appropriate for a data format required by UDDI specification. When the book uploading clients contact the providers for the first time, providers ask the client if they wish to upload the existing book information to the UDDI server. When new books are added to the providers, the information is also automatically uploaded to UDDI.

4.4. Upload Client

Upload clients are used as a front end to operate the providers. Since providers are just services waiting to be invoked by interested parties, a GUI in some form is necessary for success operation of providers. In our Upload Client, on first login, the user is presented with existing book information on providers and asked if each entry should be added to UDDI. The providers upload the entry to UDDI upon receiving upload request from Upload Client. Upload Client is also used to add new books to each provider.

4.5. Consumer

Consumers are programs that make use of the UDDI and Book Provider services to find required books. The consumer client first searches the entries on UDDI for a required book and if the UDDI returns a list of provider services, it contacts these services to make actual searches. When no UDDI entries are found for a specific search, the consumer client does not look for the book anymore, informs the user of the absence of the book.

4.6. UDDI as Filter

Because the UDDI has to be first contacted before contacting book providers, only the providers which hold the entry to the required search will be contacted at the second stage. No book providers will be contacted if no matching entries are found on UDDI.

5. Experimental Results

5.1. Set Up

To compute the performance gains made by the UDDI based SOA approach, a traditional client server system is used as a comparison. First, five computers are set up to consume services running on one computer. HTTP Analyzer of NetBeans and JDeveloper Java IDEs are used to capture performance statistics. Secondly, a similar configuration is used to test the performance of the UDDI-based web services. A UDDI server needs to reside between the five client computers and the server - with a total of seven computers.

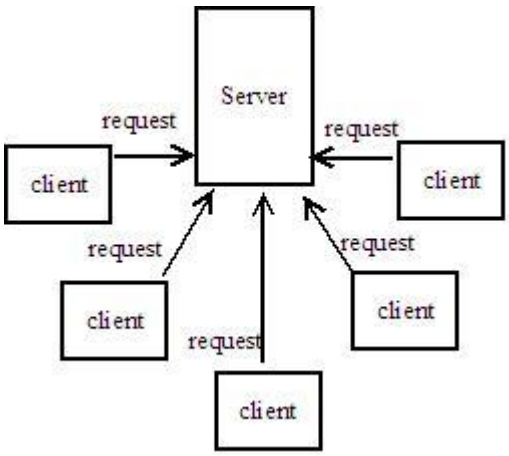


Figure 3. Traditional Approach

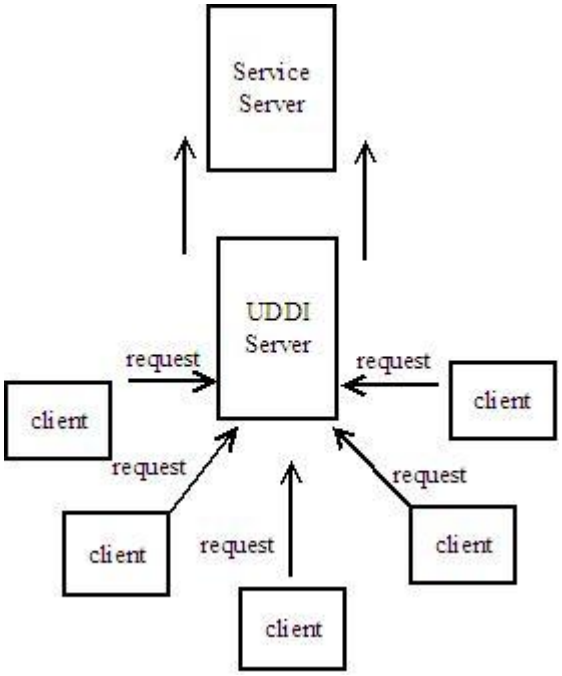


Figure 4. UDDI-based SOA Approach

5.2. Traditional Setup Results

When automated requests are sent to the server, with increases in the number of clients, the traffic increases proportionally. The results are graphed below.

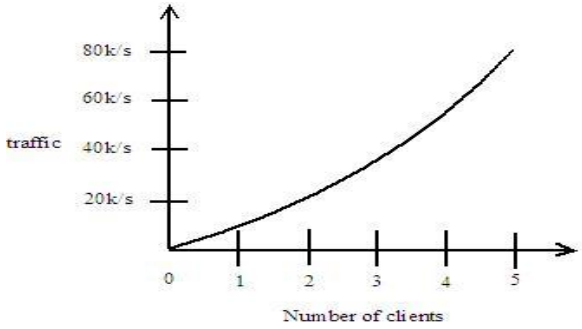


Figure 5. Traffic consumption of Traditional System

The traffic is expected to increase in this pattern with more clients connecting to the server. In real world situations where thousands of clients (and possibly millions of clients) are connecting to one server, the resultant load can be overwhelming. The effect of this on memory consumption is charted as follows.

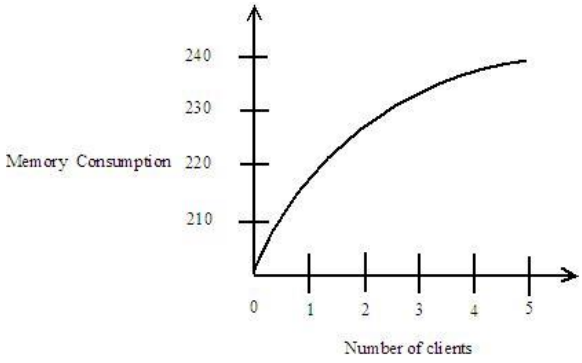


Figure 6. Memory consumption of Traditional System

5.3. UDDI-based SOA Results: UDDI

As expected, the UDDI approach has XML overhead compared with traditional approach and the UDDI server receives more traffic.

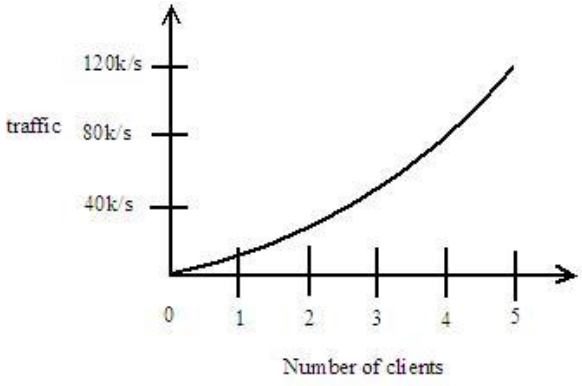


Figure 7. Traffic consumption of UDDI-based SOA system (UDDI)

However, with the efficiency of XML processing, the effect on memory is similar to that of traditional approach.

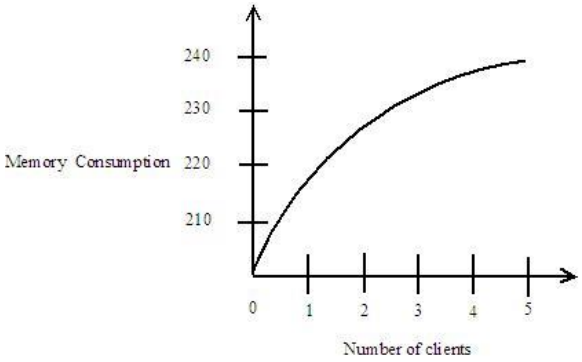


Figure 8. Memory consumption of UDDI-based SOA system (UDDI)

5.4. UDDI-based SOA Results: Providers

The real performance improvement is in the services themselves. As the UDDI server serves as a filter to non-qualifying requests, the number of searches reaching the actual providers is lowered.

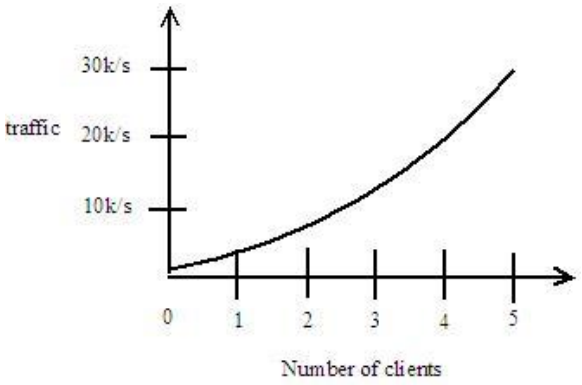


Figure 9. Traffic consumption of UDDI-based SOA system (Providers)

This results in better memory consumption of the providers.

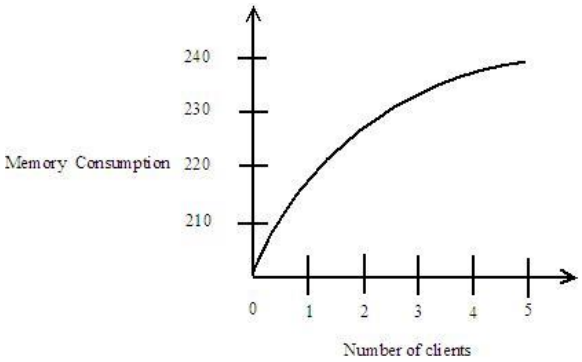


Figure 10. Memory consumption of UDDI-based SOA system (Providers)

6. Conclusion

From the above facts and data, several benefits of the UDDI-based SOA web service can be drawn. First of all, the use of a centralized registry means that applications are distributed and accessibility is better guaranteed with this approach. Secondly, for high traffic sites, using a registry filter can result in better performance for the service provider. Considering that, UDDI searches are more efficient than index searches; this can result in substantially higher throughput for data service providers. Finally, in addition to technical benefits, users of web services also enjoy the choice of web services registered on the same UDDI server.

Hence, it can be safely claimed that under situations similar to the one developed here, the use of web services and the UDDI layer can highly improve the infrastructure and performance of web-based systems.

References

[1] David Chappell, Tyler Jewell "*Java Web Services*" First Edition March 2002, ISBN: 0-596-00269-6, 276 pages

[2] Eric Newcomer "*Understanding Web Services"* May 13, 2002, ISBN: 0-201-75081-3, 368 pages

[3] G Wiederhold, "Mediators in the Architecture of Future Information Systems", IEEE Computer

[4] JianJun Yu, "Dynamic Web service invocation based on UDDI", E-Commerce Technology for Dynamic E-Business, 2004. IEEE International Conference

[5] Justin R. Erenkrantz, "Web Services: SOAP, UDDI, and Semantic Web", University of California, Irvine

[6] Richard Monson Haefel *"J2EE™ Web Services"* October 24, 2003, ISBN: 0-321-14618-2, 928 pages

[7] SOAP 1.1 W3C note: http://www.w3.org/TR/#Notes

[8] UDDI Programmer's API Specification: http://www.uddi.org/pubs/UDDI_Programmers_API_Specification.pdf

[9] UDDI Data Structure Reference: http://www.uddi.org/pubs/UDDI_XML_Structure_Reference.pdf

[10] W3C XML and related recommendations: http://www.w3.org/TR