From Sockets and RMI to Web Services

Mark A. Holliday, J. Traynham Houston, and E. Matthew Jones
Department of Mathematics and Computer Science
Western Carolina University
Cullowhee, NC 28723
01-828-227-3951

{holliday, thouston, emjones}@email.wcu.edu

ABSTRACT

Traditional coverage of network programming techniques in a computer networking course addresses sockets, remote procedure call, and object-oriented remote procedure call. We propose two innovations to that coverage. The first is to emphasize the historical development of those techniques as a sequence with each technique evolving from the previous one. The second innovation is to extend the historical development and the techniques to the important current technique of web services.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *client-server, distributed applications, distributed databases, network operating systems.*

General Terms

Design, Experimentation, Security, Standardization.

Keywords

Net-Centric Computing, Computer Science Education, Sockets, Java Remote Method Invocation, Web Services.

1. INTRODUCTION

The Final Report on Computer Science Curricula of the Joint Task Force [1] identifies Net-Centric Computing as one of the top-level topics in the Computer Science Body of Knowledge. There are excellent textbooks introducing computer networking available. Consequently, our approach requires a computer networking textbook [6] but to cover network programming and distributed applications by a series of programming projects.

Our projects include detailed handouts to cover the needed concepts. This past year we introduced a significant change in the series of programming projects by adding a project involving web services. How we introduce web services as an evolution from other approaches to network programming is noteworthy and is the subject of this paper. In prior work, Gagne [3] has also reported a course that includes socket and RMI-based projects, but not the historical evolution approach nor the extension to web services. Interestingly, the more recent papers on integrating web services have focused on CS1/CS2 [7] and on a first year graduate course [4] instead of on the computer networking course.

In the next section we illustrate how the course teaches network programming techniques as an historical evolution. Section Two covers the historical sequence taught prior to this semester. Section Three explains the programming projects we have developed that follow that historical development. Section Four explains how web services are a natural next step both in terms of the historical development of network programming and in terms of pedagogy. Section Five discusses the web services project and the software infrastructure it requires. Section Six reports on our experiences with that project. We conclude in Section Seven.

2. HISTORY

The historical development of network programming helps explain the concepts for current approaches. We follow that historical approach in our course and illustrate the highlights of that course content in this section. A natural starting point is the definition of a socket as a communication end-point in an Internet Protocol-based network. The original definition of such a socket for the Advanced Research Project Agency (ARPA) internetwork was written in 1971 [9]. What is most relevant today is the Berkeley Sockets application programmer interface (API) that was introduced as part of the 4.2BSD UNIX operating system in 1983 [6]. Descendants of the Berkeley Sockets API have become the *de facto* standard for operating systems that support access to the Internet.

The original specification of that API required the C programming language. However, versions of that API have been developed for more modern programming languages. For example, the java.net package of Java includes a version of the Berkeley Sockets API. It is our experience that programs written using the Java version of this API are simpler to write, easier to understand, and less likely to have errors than programs written using the C version.

Inter-process communication across the Internet is primarily based on versions of sockets. Therefore, the socket interface lies at the juncture between a course on computer network protocols and a course on using network programming to develop distributed applications. As a reflection of the importance of this transition, our networking textbook does include sections on Java-based socket programming.

However, today's role of socket programming is analogous to that of assembly language programming in understanding the operation of a single computer. Socket programming is important in order to understand how Internet-based inter-process communication works, but it is not the level at which programs are developed typically. Instead, network programming is more

likely to be done at a higher-level that is compiled (using a stub compiler) to a set of socket programs.

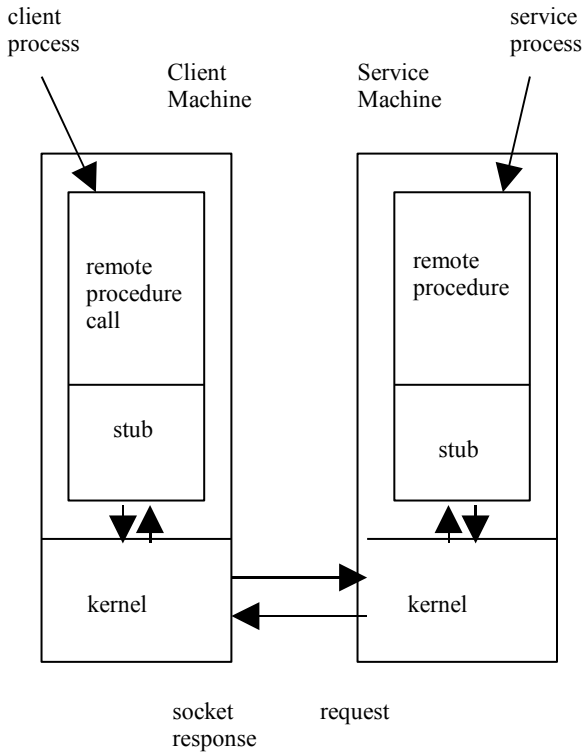

Figure 1: The Steps of a Remote Procedure Call.

That higher level evolved from the idea of a Remote Procedure Call (RPC). A first version of RPC was proposed in 1976 as the Procedure Call Protocol in RFC 707 [8]. Implementations of RPC began appearing in the early and mid 1980s with the version from Sun Microsystems being the most popular. RPC supplied two key and important innovations. One, as the name suggests, is based on the observation that many inter-process communication exchanges form request-response pairs (that is, they are client-server interactions). Such an exchange is analogous to the passing of arguments upon invoking a procedure and the return of the return value when that procedure invocation completes. Thus, if some preliminary steps are taken (interface specification, stub compilation, and linking of client and server stubs) the inter-process communication across the Internet can be made to appear similar to a regular procedure call. The 1984 paper by Birrell and Nelson [2] is widely cited as a reference describing RPC as it is understood today.

Figure 1 illustrates the basic steps in a remote procedure call. The interface of the remote procedure is defined in an interface definition language. A stub compiler generates a client stub and a server stub from that interface. At link-time the linker links the client stub with the client object file to create the client executable and links the server stub with the service object file to create the service executable. At run-time the user-written client code calls the remote procedure which causes a call to the client stub. The client stub translates the procedure call into a request and response on a socket connection. The actions on the socket are done by the client stub trapping to the client machine's operating system kernel. The service machine does the reverse sequence of steps from the kernel, to the service stub to the remote procedures in the user-written service code. The remote procedure executes generating a return value. The return value then retraces the above steps back to the user-written client code.

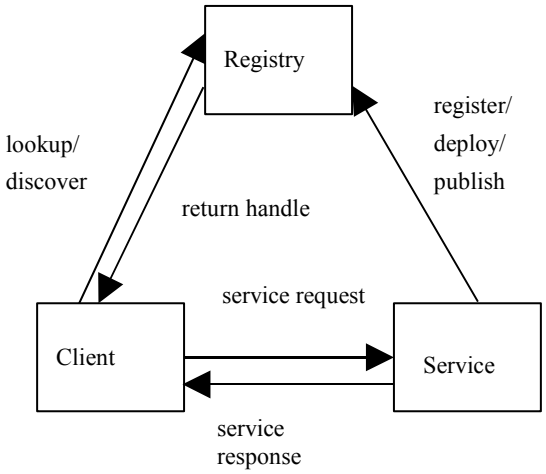

Figure 2: The three parties involved in a remote procedure call.

The second innovation added a level of indirection. Instead of the client having to explicitly specify the location (IP address and port) of the service, the client contacts a third party. The client indicates to the third party the type of service sought; this is called lookup or discovery. That third party returns the location of an entity that provides that service. Thus, RPC forms a triangle between the client, the third party, and the service. The service registers (also called publishing or deploying) with the third party and the client contacts the third party to find the location of the service. This second innovation had been added to RPC by the time of the Birrell and Nelson paper. As shown in Figure 2 this third party was called the registry by Birrell and Nelson with the service registering with the registry and the client doing a "lookup" of the service at the registry.

By the early 1990s object-oriented versions of RPC began to appear. For example, Microsoft's Distributed Component Object Model (DCOM) was based on Microsoft RPC. Java, being an object-oriented language, includes a version of RPC called Remote Method Invocation (RMI) within the java.rmi package. Figure 3 provides a timeline for the key events in the development of network programming techniques described so far. Figure 3 also demonstrates the timeline for the network programming technique of web services to be described later.

This historical review of network programming influenced us in the design of the programming projects we have used prior to this semester. We contend that the students seeing the historical development of network programming helps them in several ways.

- Both the older and lower level socket programming and the more modern object-oriented RPC should be taught because both are in use today. The object-oriented RPC is used explicitly and socket programming is used implicitly, just as high-level languages and assembly languages are used.
- The design choices and features of current network programming techniques are to a large extent a result of

issues with the earlier techniques (For example, the addition of a third party, the registry, as seen in the 1984 Birrell and Nelson paper, but not seen in the original 1976 RFC 707 proposing a Procedure Call Protocol.). Here the historical development aids in understanding the more recent techniques.

- Much of what at first glance appear to be new ideas today are in fact the same or similar ideas from several decades ago.

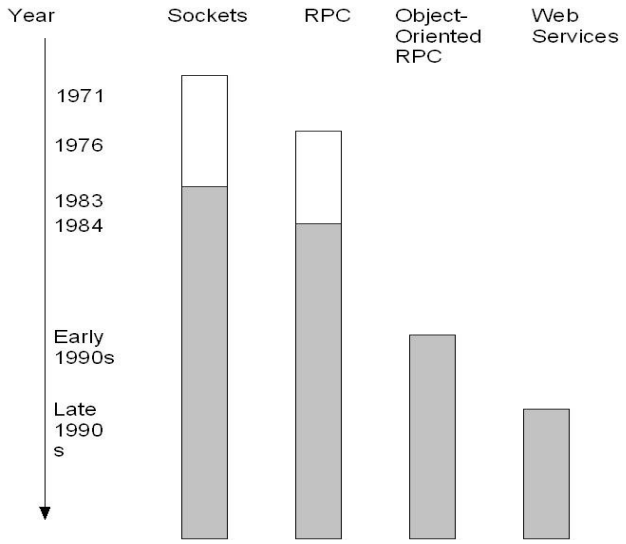

Figure 3: A Timeline showing when each type of network programming became significant. For sockets and RPC we show the time from the original paper to the most influential paper as white rectangles.

3. SOCKETS AND RMI PROJECTS

Prior to last year, two of the four programming projects in our course were designed around the historical development described above. Those two projects in order are:

1. a distributed chat client and server using sockets which access Transmission Control Protocol (TCP) at the transport layer.
2. a distributed chat client and server using Java RMI

The students are given the graphical user interface shown in Figure 4 as the starting point for the chat client. The students augment the graphical interface with two threads: one for the notepad (for the user to enter messages) and one for the log (for all messages entered by any user or generated by the chat server to be displayed in chronological order). The students also must be able to run multiple clients and to have clients running simultaneously on multiple machines.

For the socket version the students also are required to implement a multi-threaded chat server. The master thread of the chat server upon a new connection from a notepad thread of a chat client creates a new thread to service that connection. The new thread in the chat server reads messages from the notepad and writes them into a History data structure in the chat server. A continuously running output thread in the chat server detects when a message is added to the History data structure and writes that new message to the log process in each chat client. The student must prevent race conditions between the input threads and the output thread as they access the History data structure and Users data structure by adding synchronization.

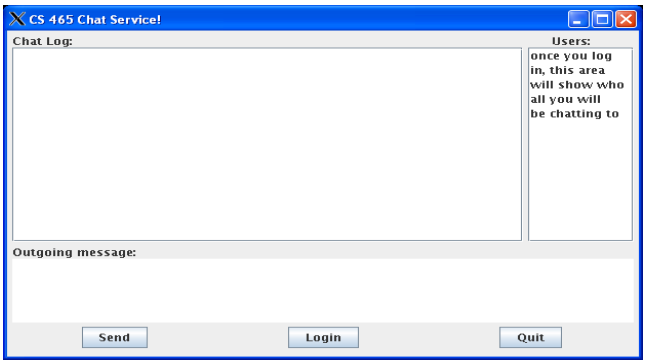

Figure 4: A screenshot of the graphical interface the students can use when developing their Chat projects.

An important feature of both the RMI and socket versions is that the log process of the chat client is a service in the client-server sense. Thus, the chat server is a server for the notepad of the chat clients, but the chat server is also a client of the log of the chat clients. Figure 5 illustrates this fact as well as the sequence of messages exchanged in the RMI version.

The RMI version allows the students to experience for themselves the issues involved in using stub compilation to create client and server stubs, starting a registry, registering a service, and having a client lookup a service in a registry and then invoke the service.

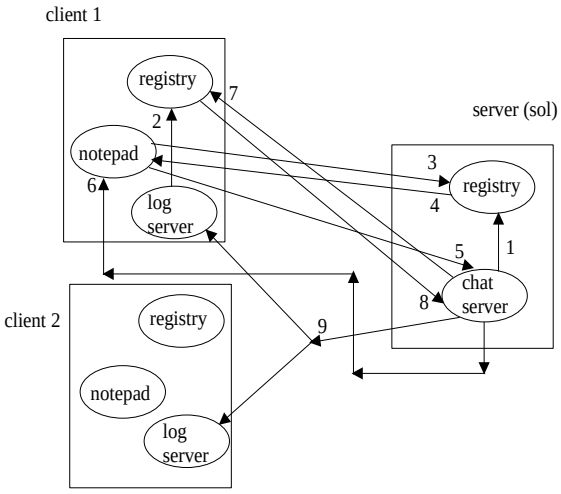

Figure 5: The design and sequence of exchanged messages for the Java RMI implementation of the Chat project.

4. WEB SERVICES

The reader has probably noted that our review of history stopped in the mid-1990s with the widespread use of object-oriented

versions of RPC such as Java RMI. More than a decade has passed and it is time that such a course continue the historical progression by introducing a more current approach to network programming while showing how it relates to the earlier approaches.

The new approach is web services. Web services consist of a broad set of specifications defined by the World Wide Web Consortium [10]. Understanding all of its complexities would take much more time than is available in one segment of one undergraduate course. However, the most important basic aspects of web services can be understood and used by students in such a course if the course follows the historical development we have outlined. As a result, the basic aspects of web services are seen as a natural next step in this historical evolution. Many of the concepts will already have been covered and the students will have implemented software using those concepts in their previous Chat projects.

The key insight in motivating web services occurred in the late 1990s: that object-oriented RPC approaches worked well, but they were too difficult to deploy because they required features that were not already available to everyone. Web services addressed this difficulty in two ways.

- The web services data representation innovation was to adopt XML (eXtensible Markup Language) for data representation. Thus, the service registers itself with the registry using a descriptor called a Web Services Deployment Descriptor (WSDD). The service's WSDD provides a brief description of the service using a special XML schema. Also, the service provides the information about the service needed by the stub compiler to generate the client and server stubs using a descriptor that is written in Web Services Description Language (WSDL). WSDL is also defined using a special XML schema.
- The web services network protocol innovation was to layer the RPC on top of the application layer protocol HTTP (HyperText Transfer Protocol) that is the basis for the web (bindings to other lower layer protocols do also exist). In particular, the RPC requests and responses are contained within SOAP (Simple Object Access Protocol) packets that are encapsulated by HTTP packets. HTTP browsers and servers are ubiquitous. Also, the use of HTTP addresses firewall limitations of earlier RPC approaches because firewalls typically allow HTTP packets through.

Web services fit naturally into our emphasis on historical evolution. Initially, web services appear to be quite novel. This is partly due to their surface appearance using XML syntax. It is also partly due to changes in terminology. However, at a deeper level, the key concept directly descends from the original RPC work in the early 1980s. The concept of a service interface being used to generate client and server stubs via a stub compiler still exists (see Figure 1). Likewise, the concept of the triangle of the service, the registry, and the client still exists (see Figure 2).

5. THE WEB SERVICES PROJECT

As with the earlier approaches to network programming, the course lectures follow the content and format outlined in the above section. Just as we developed versions of the Chat project for the students to practice with socket programming and Java RMI programming, we developed a version of the Chat project for the students to practice web services programming.

The requirements for the Chat project are the same as those for the RMI version except for one significant difference. Recall that the chat client is divided into a notepad and a log. The notepad thread is a client of the chat server in the client-server sense. However, the log, though part of the chat client, is a server in the client-server sense. Thus, the chat server is a server for the notepad, but then acts as a client when interacting with each of the logs.

We decided that the extra complexity of having to deal with two services was too difficult for the web services version. Consequently, for the web services version we allowed the students to implement the log using polling. In other words, the log is implemented as a client in the client-server sense that runs in an infinite loop that calls the chat server for new messages, displays any new messages, and then performs a delay (using the sleep method of the Thread class) before repeating.

After defining the requirements for the project, we introduced what the students need to know to implement the project. Our approach starts with a working example web service and accompanying web client. The web service provides a generalized HelloWorld service with four methods in the service interface including methods that require multiple arguments and return values. We go through the key files in the HelloWorld directory structure. We then review the steps of running the HelloWorld example.

We use the Apache Axis implementation of web services and for that reason we show the directory structure used for a web service in Apache Axis. There are only a few key top-level files:

- build.mappings
- namespace2package.mappings
- build.sh
- Hello.java
- services_HelloWorld.gar

The file build.sh is the shell script for creating a deployable web service. It uses the build.mappings and namespace2package.mappings configuration files and Hello.java which is a Java interface file that satisfies the WSDL (Web Services Description Language) requirements. The file services_HelloWorld.gar is the output, the deployable version of the web service.

There are three relevant subdirectories:

- clients/
  - ChatProgram/HelloWorldClient.java
- schema/
  - HelloWorld/HelloWorld.wsdl
- services/
  - HelloWorld/deploy-server.wsdd
  - HelloWorld/impl/HelloWorld.java

The client Java source file in the clients directory is interesting in part because it contains the actual invocation of the remote methods but also because it has the code for doing discovery (that is, contacting the registry for a reference to the service). The WSDL file in the schema directory is, in essence, the XML equivalent of the Java interface for the service; it holds the detailed information about the service that the stub compiler

needs in order to generate the client and server stubs (as in RMI, the server stub is actually called a skeleton).

The service Java source file in the services directory is interesting for what it does not have. It does have the method bodies for the remote methods. However, unlike RMI, there is not a main method that contains the code for registering the service. Instead in our project the deployment is done manually as described below. The Web Services Deployment Descriptor (WSDD) file in the services directory is used during that deployment to provide the registry with the information the registry needs.

As we go through the directory structure and examine the contents of each file, we relate the approach to how the code appears in RMI. The largest difference is the syntax for describing the brief descriptor (WSDD) for deploying a web service and the more detailed descriptor (WSDL) needed to generate the stubs for the web service. So that is where we focus most of the class time.

The WSDL file serves the analogous role as the Java interface file in RMI: it describes the interface of the remote methods at the level needed by the stub compiler. That the two files are analogous is not immediately apparent partly because of the XML syntax of the WSDL file, but also because the terminology is completely different. We have found especially helpful a table we developed in our handout that maps each of the elements of a WSDL specification to the corresponding syntactic feature of a Java interface. For example, a WSDL PortType tag is equivalent to the name of the complete Java interface.

After the survey of the directory structure, we go through the steps of running the HelloWorld example in a live demonstration in class: compiling the web service and web client, deploying the web service, finding a free TCP port for the container (the term for the registry) process to listen on, starting a container, running multiple web clients concurrently on different machines, stopping the container, and finally undeploying the web service.

At this point, the students are ready to use HelloWorld as a basis for their Chat project. The configuration files need to be edited and the WSDD and WSDL files need to be written using the HelloWorld files as examples. The service source file needs to modify the large amount of code for the RMI version of the chat service to fit the format a web service. Instead of a single command line client, as in HelloWorld, the students will need to write two command line clients (for the notepad and the log) or adapt the graphical user interface version of the RMI client.

6. EXPERIENCES

Based on their comments and questions and their performance on the tests and projects, the students understood the lectures on the historical development of concepts from sockets through web services. The rate at which they successfully completed the web services project was consistent with how well they did on the sockets and the RMI versions of the chat project. The students liked learning the more recent techniques. A number of the students have shown their completed web services projects to prospective employers as examples of their work.

The one concern that our experience has raised is the complexity of the software infrastructure that the instructor needs to develop and maintain in order to offer a project such as the web services Chat project. That is another feature of the historical development that we did not mention earlier: the software environment needed to use the network programming techniques has become increasingly complex. With sockets the only environment needed is the library specific to your high-level language that converts your socket call to a system call. With RMI the code for the client and the service become much simpler, but in return a more substantial environment is required. The trend continues with web services. The environment needed for web services, Apache Axis, is more complex than that needed for RMI.

7. CONCLUSIONS

In distributed applications, one of the most important current approaches is web services. In this paper we have outlined how we are doing that in a current course. The course integrates a computer network protocol course with a course on network programming techniques for developing distributed applications. We argue that organizing the course's content and programming projects around the historical development of those techniques offers a number of advantages. This paper explains how the course does that including the novel extension to web services.

7. REFERENCES

[1] ACM/IEEE-CS, *Final Report of the Joint ACM/IEEE-CS Task Force on Computing Curricula 2001 for Computer Science,* 2001, http://acm.org/education/curric_vols/cc2001.pdf.

[2] Birrell, A.D. And Nelson, B.J., *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, vol. 2, issue 1, February 1984, pp. 39-59.

[3] Gagne, G., "To java.net and Beyond: Teaching Networking Concepts Using the Java Networking API", *Proc. of the Thirty-Third SIGCSE Technical Symposium of Computer Science Education*, Covington, KY, USA, February 2002.

[4] Humphrey, M., "Web Services as the Foundation for Learning Complex Software System Development", *Proc. of the Thirty-Fifth SIGCSE Technical Symposium of Computer Science Education*, Norfolk, VA, USA, February 2004.

[5] Kurose, J. and Ross, K., *Computer Networking: A Top-Down Approach Featuring the Internet, Third Edition,* Addison-Wesley, 2005.

[6] Leffler, S.J., McKusick, M.K., Karels, M.J., and Quarterman, J.S., *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison Wesley, 1989.

[7] Lim, B.B.L., Jon, C., and Mahatanankoon, P., "On Integrating Web Services From the Ground Up Into CS1/CS2," *Proc. of the Thirty-Sixth SIGCSE Technical Symposium of Computer Science Education*, St. Louis, MO, USA, February 2002.

[8] White, J.E., *A High-Level Framework for Network-Based Resource Sharing, Request for Comment (RFC) 707*, 1976, http://tools.ietf.org/html/rfc707.

[9] Winett, J.M., *The Definition of a Socket*, Request for Comment (RFC) 147, 1971, http://tools.ietf.org/html/rfc147.

[10] World Wide Web Consortium Web Services Activity, http://www.w3.org/2002/ws/.