Table of Contents

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

Chapter 1. Introduction

In the history of software development, new approaches frequently bring discarded ideas back into the mainstream of common practice. Each time an idea is revisited, prior successes and failures become invaluable aides in improving the concept and making its implementation better, or at least more usable. Now I'm not saying that we keep reinventing the wheel; rather, we keep going back and improving the wheel. And doing so can often be the catalyst for new ideas and new technologies that were not possible with the old wheel.

We've seen centralized computing with mainframes and their associated terminals come back disguised as application servers and thin clients. We've seen the concept of P-Code return in the form of interpreted languages like Java and Visual Basic. The universe of software development seems to expand and contract like, well, the cosmic Universe. If you wait around long enough, you may just be able to use the work you're doing today at some time in the future.

1.1. RPC and Message-Oriented Distributed Systems

Distributed systems exist, for the most part, as loosely coupled entities that communicate with each other to accomplish some task. One of the most common models used in distributed software is the remote procedure call (RPC). One reason for the popularity of RPC systems is that they closely resemble the function/method call syntax and semantics that we as programmers are so familiar with. Technologies like Java RMI, Microsoft's COM, and CORBA all use this kind of model. Of course, you have to jump through many hoops before making the ever-familiar method call to a remote system, but even with all that it still feels remarkably like making a local method call. Often, once the method call returns we don't care how it happened.[1] Much of the work in providing that abstraction to programmers at the API level is what makes up the majority of the distributed systems implementations.

[1] I'm not suggesting that you turn a blind eye to the fact that you're making calls to remote systems. Imagine, for instance, the ramifications of iterating over a remote array of ten thousand objects by using an array accessor method that goes out to the remote system for every array element. Not exactly efficient!

Another popular model for distributed computing is message passing. Unlike the RPC model, messaging does not emulate the syntax of programming language function calls. Instead, structured data messages are passed between parties. These messages can serve to decouple the nodes of the distributed system somewhat, and message-based systems

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

often prove to be more flexible than RPC-based systems. However, that flexibility can sometimes be just enough rope for programmers to hang themselves.

1.2. Self-Describing Data

In programming parlance, the term *self-describing data* is itself self-describing. Put another way, if the question is, "What is self-describing data?" then the answer is, "Data that describes itself." Not a very useful definition. But that's the result of designing a flexible data format to be used by many, many people.

Let's look at a very simple example. Let's say we were designing a message-style distributed system for delivering stock quotes. We could design the response message format to be something like this:

1.3. XML

We've all been staring a form of self-describing data in the face every time we use a web browser. HTML is a good example of a standard data format that is quite flexible due to its provision for self-describing elements. For example, the color and font to be applied to a particular section of text are described right along with the text itself. This kind of self-describing data is commonly referred to as a *markup language*. The content is "marked-up" with instructions for its own presentation. This is very nice, and it obviously has gained an incredible level of industry acceptance. But HTML is not flexible enough to accommodate content that was not anticipated by its designers. That's not a knock on HTML; it's just the truth. HTML is not extensible.

The Extensible Markup Language (XML) is just what we're looking for. XML is a hierarchical, tag-based language much like HTML. The important difference for us is that it is fully extensible. It allows us to describe content that is specific to our own applications in a standard way, without the designers of the language having anticipated that content. For example, XML would allow me to create content to represent the stock quote response message from the previous section. It defines the rules that I must follow in order to accomplish that task, without dictating a specific format.

1.4. API Specs Versus Wire-Level Specs

Java programmers are used to dealing with API-level specifications, where classes, interfaces, methods, and so on are clearly defined for the purpose of addressing a specific need. These specifications are designed to be independent of any specific implementation, focusing instead on the abstractions that must be implemented.

Consider the Java Message Service (JMS) specification. It fully describes the API that Java applications can use to access the features of message-oriented middleware (MOM) products. The motivation for a standard API is simple: if MOM vendors adopt the API, it becomes that much easier for programmers to work with the various product offerings. In

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

theory, you could swap one JMS implementation for another without impacting the rest of your code. In practice, it means that product vendors might be somewhat handcuffed, unable to provide alternative APIs that leverage features and capabilities of their own products without sacrificing compliance. Nevertheless, API specifications have been around a long time, and they do achieve most of what they're intended to do.

1.5. Overview of SOAP

One of the more recent forays into the world of distributed computing resulted in a wire-level specification called the Simple Object Access Protocol, or SOAP. The protocol is relatively lightweight, is based on XML, and is designed for the exchange of information in a distributed computing environment. There is no concept of a central server in SOAP; all nodes can be considered equals, or even peers.

The protocol is made up of a number of distinct parts. The first is the *envelope*, used to describe the content of a message and some clues on how to go about processing it. The second part consists of the rules for encoding instances of custom data types. This is one of the most critical parts of SOAP: its extensibility. The last part describes the application of the envelope and the data encoding rules for representing RPC calls and responses, including the use of HTTP as the underlying transport.

1.6. SOAP Implementations

As I write this book, there are dozens of SOAP implementations, and new ones emerge all the time. Some are implemented in Java, some aren't. Some are free, some aren't. And inevitably some are good, and some aren't. It would be impractical to do a side-by-side comparison of every available implementation or even to give equal coverage to them all. On the other hand, it wouldn't be wise to focus on a single implementation, since that would present a bias that I don't intend. A reasonable compromise, and the one I've elected to use, is to select two interesting Java SOAP implementations and use them both extensively throughout the book. This gives you an opportunity to see different APIs and programming strategies. In Chapter 9 and Chapter 11, I'll break this rule and look briefly at a couple of other important SOAP technologies.

1.6.1. Apache SOAP

The Apache Software Foundation has an ongoing project known as Apache SOAP. This is a Java implementation of the SOAP specification that can be hosted by servers that support Java servlets. The examples in this book are based on Apache SOAP Version 2.2, which is available at http://xml.apache.org/soap/index.html. Four very important factors led me to choose this implementation: it supports a good deal of the specification, it has a reasonably large user base, the source code is available, and it's free.

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01

1.7. The Approach

This book certainly does not cover every aspect of the SOAP technologies. My goal is to give you a good understanding of the major aspects of SOAP in the context of Java software development. You'll find that many of the examples are presented not only in Java source code, but also in the SOAP XML that is generated through the execution of the Java code. This will give you a sense of what the various APIs are actually accomplishing. Learning SOAP this way will allow you to go beyond the scope of this book with confidence, exploring the features and capabilities of the implementations I have covered as well those I have not.

1.7.1. No Security

One particular area is not covered in this book: security. How can you talk about a distributed computing technology without talking about security? The answer is actually quite simple: the SOAP specification does not deal with security. The current implementations rely on the security features of the hosting technology. Be it SSL, basic HTTP authentication, proxy authentication, or some other mechanism, all security is a function of the hosting technology and not part of SOAP itself. It's expected that either a future version of the SOAP spec or a separate SOAP Security spec will address that issue, but for now you'll have to rely on whatever your hosting technology supports.

1.8. Getting Started

If you plan to work with the examples, you'll need to install all of the necessary software components, including the JDK, the SOAP implementations described earlier, and a number of other supporting technologies. All of these packages are reasonably well documented, so you should have no trouble getting them installed properly.

The chapters in this book are arranged so that each one builds on concepts of the preceding chapters. I suggest that you follow along in order, but of course that's entirely up to you. If you are already comfortable with XML or (even better) with some aspects of web services, you may find that you can jump around a bit.

Java and SOAP By Robert Englander ISBN: 0-596-00175-4 Publisher: O'Reilly
Print Publication Date: 2002/05/01