Will Web Services Choke on their Own Complexity Due to a Shortage of Capable Developers?

by XXXXXX XXXXXX

1. Introduction

As the business community becomes familiar with an incipient e-business technology, the IT industry produces implementations that make it easier for programmers to think about that technology in light of implementations familiar to them, particularly the design and coding conventions those implementations induce. Abstract knowledge about the fundamentals of the technology moves into the background of day-to-day, production-oriented concerns, especially when the developer uses the same tools, simply to varying effect depending on his current task. Additionally, implementation-specific lexicons can develop. But in order to make judgments concerning Web Services *per se* we must understand them generically. It will suit our purposes to define Web Services as

> standardized ways of integrating Web-based applications using the XML, SOAP and WSDL and UDDI open standards over an Internet protocol backbone [1].

Since we wish to discuss the newest Web Services technologies, we reject standards-independent definitions as inadequate to distinguish current trends from unsuccessful predecessors and early custom jobs that predate these standards' widespread acceptance.

We assert that the complexity of Web Services technologies will not prevent them from predominating in the world of e-business applications due to a too-small pool of capable developers. First we discuss the complexity intrinsic to raw Web Services technology and suggest means by which this is being and can be overcome. To further support our claim, we draw historical parallels to the integration of hard-to-understand technologies into the IT mainstream, for instance by wrapping those technologies so as to hide the details from the programmer, by refining the technology itself, and by changing programmer education to fill the ranks with enough competent programmers to spread Web Services programming around the IT world. Finally we discuss how the problem of complexity is ill-posed in the sense that multiple persons can contribute to the success of Web Services projects, avoiding the requirement that a large number of individuals be well-versed in all of the smaller technologies involved. Throughout the discussion we show how common misgivings about the viability of Web Services technologies are really not barriers if we are flexible in our search for solutions.

2. The Complexity of Web Services

Web Services are sophisticated on several levels because they are ambitious, designed to achieve interoperability, reusability, and ubiquity across e-business applications. Whereas traditional applications development can be as simple as facility with one high-level programming language, an IDE, and basic debugging and testing techniques, Web Services demand considerable sophistication on the part of the developer. This much is clear even if we disregard the technologies of XML and HTTP in

our analysis since they can be made invisible to the programmer with a judicious choice of development tools currently available. Later it will become clear that the obstacles to effective development are not as formidable as a breakdown of Web Services technologies would suggest, but let us investigate the breadth of knowledge a single programmer working alone would have to master in order to become an expert in his field. After all, it is this admittedly daunting body of supposed requisite knowledge that gives rise to a large share of current skepticism regarding the viability of Web Services.

2a. Remote service invocation through XML-RPC

To begin with, the developer would have to be conversant with the client-server programming model as well as the specific mechanisms that his third-party tools employ to invoke a remote service's methods. This is not as simple as a conventional remote procedure call or dynamic loading and linking. Apache's SOAP documentation [2] delineates the steps in sending a request to an RPC-based SOAP service:

1. Obtain the interface description of the SOAP service.
2. Make sure that there are serializers registered for all parameters that you will be sending, and deserializers for all information that you will be receiving back.
3. Create the org.apache.soap.rpc.RPCMessage.Call object.
4. Set the target URI into the Call object using the setTargetObjectURI(...) method.
5. Set the method name that you wish to invoke into the Call object using setMethodName(...).
6. Create any Parameter objects necessary for the RPC call and set them into the Call object using the setParams(...) method.
7. Execute the Call object's invoke(...) method and capture the Response object which is returned from invoke(...).
8. Check the Response object to see if a fault was generated using the generatedFault() method.
9. If a fault was returned, retrieve it using the getFault(...) method, otherwise extract any result or returned parameters using the getReturnValue() and getParams() methods respectively.
10. For every subsequent method call, repeat steps 3 through 9.

Web Service critics such as [3] claim that XML-RPC using SOAP is too complicated, but we will show that it is easier than this list suggests and that it is not a significant barrier to successful Web Services programming.

2b. Database programming

Since much of Web Services – indeed, much of client-server programming in general – revolves around remote access to a centralized database, the developer should have a working knowledge of database technology. Usually this means familiarity with database theory and fluency in SQL, the building-block language of programmatic interaction with databases. The programmer must also be able to establish a connection to the database, which requires locating the database on the server and starting up a database driver. The difficulty of locating the database varies with the database management tool the programmer is using; for instance, MySQL automatically finds any database residing in a specific location inside the MySQL installation, whereas Microsoft Access requires the user to register a Data Source Name object with Windows to make the database visible to applications. Hence there are not just development but also administrative tasks involved in setting up a Web Service.

2c. Interface Development

It is possible to implement Web Service clients as command-line executables that return simple response strings, but frequently a more user-friendly interface such as dynamically-generated web content is desirable. There are a number of technologies for web page generation including servlets, EJBs, JSPs and PHP, which are essentially either sizable, specialized Java packages or their own languages; thus using them effectively takes specialized knowledge. Additionally, to understand what it is they're creating developers should be familiar with building simple static web pages through HTML.

2d. Other Issues

Finally, a number of other technologies are important in creating Web Services. Configuring a programmer's basic development environment such as Apache Tomcat or Microsoft's .Net and installing SOAP and XML-RPC tools is a nontrivial administrative task. Additional programming issues include the need for proper synchronization of transactions just as in other client-server database applications, and security is a concern especially since the service is passing messages over the Internet. Testing and debugging Web Services is much different from debugging conventional applications. At some point after the developer has mapped out the intended functionality of his service, he must craft a WSDL descriptor file. Finally, when his service is ready for deployment he must package an interface for distribution, and enable access to his server from the outside world.

From this summary of the technical issues of Web Service creation, it is not hard to see why some express doubt as to the viability of the technology. We now present reasons for optimism in the face of these challenges.

3. Software-based solutions: Ameliorating the technical complexities

Before we explain why Web Services are not as prohibitively complex as many believe them to be, it is worth recalling our claim precisely: that Web Services programming is not so complex that there will be insufficient programmers with the requisite knowledge to make Web Services the predominant infrastructure for e-business. Some of the following suggestions might not be ideal from a software engineering point of view, but we are attempting to set the record straight on how a new programming specialty can be made more accessible to mainstream developers when certain practices are followed, regardless of the design issues involved and the possible tradeoff of thorough understanding for expediency of implementation.

It is possible to work around much of the complexity in Web Services programming, and members of the freeware and corporate software communities are quickly plugging in the gaps to make service development, deployment and discovery as easy as possible. The Apache Software Foundation was among the first organizations to wrap certain components of low-level Web Services technology in easy-to-use interfaces

that give the appearance of ordinary remote procedure calls. Their development tool, Apache SOAP [4] was the result; this tool hides the details of XML-RPC messaging and HTTP transport in Java wrapper procedures that appear to the developer as ordinary remote procedure calls; the two are in fact often confused. [5,12] Other providers have quickly following suit by creating wizards for designing WSDL, skeletal server-side logic, client stub code, and complex type mappings [6], and full-fledged Web Services IDE's are now available [7]. As a consequence, it is likely that in the future, mastering the general procedure of Web Services construction will be more important than understanding the detailed configuration settings and obscure features of these technologies, which have their roots in low-level networking and message-passing issues. Later we will discuss means by which developers can acquire the necessary knowledge at minimal cost and effort.

Even if we do not make full use of third-party development tools, much utility can be derived from Web Services by simply copying the more arcane configuration code as-is and fleshing out the client- and server-side stubs with the right functionality. In other words, much of the complexity in Web Services is not of great concern if we are willing to trade off thorough comprehension of every line of code for the ability to intuitively discriminate between configuration settings and arcane details we can take for granted, and implementation details we will have to adjust and flesh out for our purposes. Abundant walkthroughs and tutorials on the Internet make this approach feasible. Shortly we will discuss how this approach applies to the previously cited example of invoking an XML-RPC procedure in a Java client. But generally speaking, we simply apply intuition and experimentation, fine-tuning our code through trial and error perhaps with the help of more experienced persons, tutorials and online developers forums, until we have a working body of code; and to expand functionality in the future, we will copy what we've done and modify some names and function signatures, then compose the new functions. Perhaps this is a drastic simplification of *ad hoc* software development techniques, but these techniques are nothing new, and taking into account the sizable body of older technologies that Web Services inherits, the new territory that Web Services represents (and where things can theoretically go wrong) is really not as vast as it might look.

To illustrate this point with an example, let us recall the ten steps involved in invoking an XML-RPC procedure from a Java client. The first two steps are potentially the most difficult. The first step, obtaining the interface description (and making sense of it) takes little effort if the developer has a WSDL utility to generate client stubs. Even if not, third-party services frequently supply documentation of their interface as a list of method signatures; furthermore, tools to view WSDL files in Java-like pseudocode are available [8]. It is not likely that our sole source of information is a WSDL file and we have no tools to work with; we would then have to decipher the structure of the service's interface from the WSDL text by locating the keywords of interest to us – those which identify the service name and location (URI), the operations comprising the interface and their signatures (*i.e.* their inputs and outputs), and any custom data types the service uses. Then we can begin coding. In this case, quick tutorials and references are available at [9, 10] The developer only needs to decipher one WSDL file once to work with the service, although admittedly, WSDL files for sophisticated services can be quite complicated. The

second step in writing the client, creating the necessary serializers and deserializers to map to the service's custom data types, is perhaps the most complex step; see for instance [11]. But not all services trade in such structures, and in any case steps 3-9 are much easier and more intuitive. Step 3 is easily copied from sample code, steps 4 and 5 call for reading a string from the documentation and steps 6-10 are workaday exercises in object populating, function calling and error checking. In many cases, all ten steps can be copied and pasted and a few identifiers and function parameters changed. The Apache SOAP documentation [4] explains occasional variations to watch out for in such areas as the message-encoding and -passing style, which correspond to explicit tags we can look up in the WSDL.

So programming XML-RPC clients can be done without a thorough knowledge of the transport protocol, message encoding or other details. Now we move to the many technologies involved in Web Services programming which are neither new nor arcane. These include SQL, HTML, a general-purpose language suitable for implementing business logic, transaction synchronization, and basic networking. We claim that present knowledge of these technologies is widespread enough that that they pose no substantial barrier to the universalizing of Web Services. For we observe that no single programmer need be fluent in all these technologies to further the Web Services cause; rather, an intelligent division of labor within a development team can make up for the lack of a "renaissance man" on the team. We discuss organizational solutions further in Section 4.

Finally, we will touch briefly on the challenges of debugging Web Services. To test the business logic of a Web Service it is not necessary to write scores of XML-RPC calls and display the results; the programmer can simply add a temporary driver method to his service that calls the functions in turn, prints out the affected data after each function call to verify the correctness of both general-purpose code and SQL. The major challenges arise from the distributed nature of Web Services, where interoperability problems and version mismatch between client and server can arise [12]. Fortunately the SOAPscope tool [8] watches the communications wire of a service and can display messages in human-friendly formats to determine whether client and server are expecting the same behavior from each other.

In short, the knowledge barrier to the widespread success of Web Service technology is not as formidable as a raw list of component technologies would suggest. The conventional developer's shortcut of cutting, pasting and tweaking is still widely applicable; a great deal of Web Services technology is already becoming easier and details made invisible thanks to the emergence of sophisticated development tools; and tutorials are available to explain the details when necessary. We have briefly mentioned the managerial solution of pooling developers with different skill sets, which suggests that the problem of Web Services' technical complexity is ill-posed in a sense; that is, no single developer needs to know everything. We will now leave the technical aspects of Web Services programming to investigate some key facts from education, managerial practice, psychology, the history of computer technologies, economics and the labor market which shed light on the viability of Web Services as a ubiquitous e-business technology.

4. Human and Organizational Solutions: Putting Web Services into Perspective

4a. Education and Learning

Jones [13] identified ten major channels through which software personnel acquire new information: in-house, commercial, vendor, and university education; self-study through workbooks and CD-ROMs; conferences; internet resources; books; and journals. He also classified several forms of employee training common among major software employers, including in-house training for both new and long-term hires, seminars, and financial assistance to attend conferences and university classes. Among these, in-house training rated the most effective, and the largest software employers usually sported the most extensive training programs. Commercial education and university education also ranked high in impact. Consequently, we can infer that these companies' and institutions' own educational priorities will exert a great deal of influence on the skills profile of the software development labor market of the future.

Sullivan [14] asserts, "[A]ll of the major software vendors back web services" and lists many tools they have produced to spread Web Services technologies throughout the world. By implication, these vendors cultivate Web Services talent among their own staff. Software vendors [15] and traditional commercial educators alike are beginning to offer instruction in Web Services technology to the programming public, but this phenomenon has yet to catch up with the abundance of Web Service providers offering services and products. This is understandable given the newness of Web Services and the latency in creating formal instructional curricula around new technologies. But the strong support that Web Services is receiving from important software vendors ensures a market for Web Services training for a long time to come.

Perhaps the most interesting possibility is the integration of Web Services technology into computer science education. In theory, Web Services presents the software industry with a new version of an old problem: that of experienced workers' aversion to broad technological change. But if we consider the industry's long and celebrated history of rapid change and its self-selection of curious and experimentally inclined people ("perfection-oriented learners" [16]), this is likely a very small problem indeed. Nonetheless, if the educational community introduces Web Services programming into standard computer science curricula, the field will benefit as up-and-coming developers gain exposure to it. We now present a hypothetical two-semester program for junior- and senior-level undergraduates that begins with web programming and culminates in full-fledged Web Services development.

4b. Suggested University Curriculum

We suggest a three-semester sequence in Web development, comprising two higher-level undergraduate courses followed by one class at the graduate level. The first course would focus on elementary web design and web programming with a high-level "teaser" introduction to Web Services, and the follow-up course would concentrate on

more sophisticated Web Services programming. Both classes would emphasize implementation in order to attract an undergraduate audience. To minimize costs our development tools would be Web Services freeware available from the Apache Software Foundation and Sun Microsystems [17, 18]. We assume a semester comprised of fifteen weeks of classroom instruction, and students who are (at least) juniors with two years' background in basic programming, preferably in Java although C/C++ would be an acceptable substitute. In the first course, Introduction to Web Development, students would learn HTML, JavaScript, servlets, the client-server programming model and basic service description and deployment. The students would be provided with a fully functional development environment and stub code to work with so they can focus on program functionality. The finer points of networking and service configuration would be beyond the scope of this first class.

The second course, Advanced Web Development / Web Services, would aim toward more sophisticated web programming, and familiarity with database tools and networks would be pre- or co-requisites. The class would incorporate database programming, Java Server Pages (JSP), object serialization for XML-RPC, networking issues including Web Services debugging, and touch on some high-level issues in security and software engineering.

Finally, at the graduate level students would analyze and synthesize ongoing research as well as case studies from industry, and engage in independent speculative investigation. A key component of the class would be constructing a working Web Service from scratch, to adhere to principles of good software engineering and to be interoperable with the others' web services. A brief exploration of the economics of Web Services technology would wind out the course.

4c. Managerial Issues

Web Services development groups can be more effective if managers treat Web Services like other complicated technologies such as commercial software applications consisting of three million lines of code -- or even the automobile. The manager must keep in mind that the whole technology might be a bit much for any one person to master, and consequently he divides the work of Web Services development among, for instance, a database specialist; a conventional web programmer well-versed in at least HTML, servlets and JSPs, with perhaps some experience in DCOM or CORBA; a network programmer well-versed in client-server architecture; and conventional applications programmers. We suggest a new type of specialist would be helpful, a Web Services administrator who can quickly create WSDL based on requirements specs and who can work with the programmers to diagnose problems with message encodings, URIs, database drivers and other aspects of Web Service configuration.

4d. Historical perspective

Finally, in assessing Web Services' viability in terms of labor supply, it is helpful to examine the history of technological innovations in the computer industry so that we

can compare other phenomena to the emergence of Web Services. Incrementalism is essential to the success of complex technologies. The important point is to realize that no one needs to be a master at all the details of Web Services to program in the field. For instance, from the standpoint of an assembly language programmer of the 1960's, most modern computer applications would seem impossible to implement or to use. Yet as the computer industry has evolved, software has evolved with it and so have the knowledge of programmers and the general public (and programs are seldom written in assembly language anymore).

Web Services are a new technology that stands to pick up where DCOM and CORBA failed. The latter helped familiarize the e-business world with the advantages of distributed computing In the case of Web Services, the goal of interoperability carries over from CORBA and DCOM, technologies which failed due to subtle problems with interoperability in the naming of communication endpoints, differing object interfaces, and the encoding of RPC procedure calls on the wire. Web Services' reliance on open standards such as URL/URI naming, WSDL and XML will obviate many of those interoperability problems [18].

5. Conclusion

We have seen that organizations and individual developers can successfully transition to Web Services technology; the technology's own complexity will not cause it to fail for lack of competent practitioners. While the technology is complex, developers can learn by imitation of examples, by self-teaching and by formal instruction. Online help is already abundant and conventional resources are becoming more plentiful. Furthermore, while a large supply of "renaissance man" programmers is not necessary to the success of Web Services technology so long as managers pool their team members' talents effectively, university computer science departments can train students to meet the future needs of the Web Services job market at little cost to their own computer science departments by establishing Web Services curricula consisting of two or three courses. Certainly the early practitioners of Web Services need an intensive jack-of-all-trades training, but as the technology matures, less specialized persons will be able to find a niche in the field, and neither the labor market nor the technology itself will not suffer for want of talented workers.

References

[1] "What Is Web Services?" The Webopedia Online Computer Dictionary. http://www.pcwebopedia.com/TERM/W/Web_services.html.

[2] "Writing RPC Clients," Apache SOAP v2.3.1 Documentation, http://ws.apache.org/soap/docs/index.html.

[3] DeJong, Irmen, M. Henning, M. Woyna *et al.* "Code Complexity When using SOAP." Section 7, *Web Services/SOAP and CORBA.* http://www.xs4all.nl/~irmen/comp/CORBA_vs_SOAP.html, April 2002.

[4] “Web Services: SOAP”, http://ws.apache.org/soap.

[5] Gitman, Mitch. “Part 1: Untangling the terminology”, in “Keep up with the Web service styles (and uses).” http://www.javaworld.com/javaworld/jw-10-2003/jw-1003-wsstyles.html.

[6] Cape Science Web Services Developer Community, http://www.capescience.com. For details see the Tutorials Page, http://www.capescience.com/education/tutorials/. See also Cape Clear Software, http://www.capeclear.com.

[7] “Web Services.Org: The Web Services Industry Portal: Vendor List”, http://www.webservices.org/index.php/article/vendorlist provides a sample. KnowledgeStorm.com gives a more extensive but less focused list.

[8] “WSDL and SOAP Message Analysis,” Demo linked to www.mindreef.com. Summary on www.webservices.org/index.php/article/articleview/1171/1/15/.

[9] “Structure of a WSDL Document.” http://download-east.oracle.com/otn_hosted_doc/jdeveloper/904preview/web_services/ws_wsdlstructure.html

[10] Robinson, Lance. “Basic WSDL and how to use WSDL docs to Access a SOAP Service.” http://www.devhood.com/tutorials/tutorial_details.aspx?tutorial_id=332. Further tutorials can be found by searching for “Basic WSDL” on Google.com.

[11] Bong, Gavin. “Apache SOAP type mapping, Parts 1 and 2.” http://www-106.ibm.com/developerworks/webservices/library/ws-soapmap[12] where [12] is a regular expression we use to denote a pair of URLs.

[12] Vogels, Werner. “All Things Distributed: Web Services are not Distributed Objects [etc.]” weblogs.cs.cornell.edu/AllThingsDistributed/archives/000120.html, 26 August 2003.

[13] Jones, Capers. “How Programmers Learn New Skills.” http://www.spr.com/news/NewSkillsArticle.pdf, 4 January 2002.

[14] Sullivan, Tom. “News: Web Services”, *Infoworld Online*, http://archive.infoworld.com/articles/hn/xml/01/03/12/010312hnwebserv.xml

[15] IBM Global Services, “IBM IT Education Services.” http://www-1.ibm.com/services/strategy/files2/D3FL-2013-00.pdf

[16] Strandh, Robert. “The Psychology of Learning.” www.labri.fr/Perso/~strandh/Teaching/Langages-Enchasses/Common/Strandh-Tutorial/psychology.html

[17] “The Web Services Project @ Apache.” http://ws.apache.org/.

[18] “Java Technology”, http://java.sun.com/downloads/index.html.

[19] Gisolfi, Dan. “Web Services Architect, Part 3: Is Web Services the Reincarnation of CORBA?” http://www-106.ibm.com/developerworks/webservices/library/ws-arc2.html. IBM Developer Works, 1 April 2001.