JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 1, No. 2, July-August 2002

Objects are for Wimps: Real Developers Need S.O.S.

Mahesh H. Dodani, IBM Global Services, U.S.A.

1 WHAT'S HAPPENING TO SOFTWARE DEVELOPMENT?

The shift from business-to-consumer (B2C) applications to business-to-business (B2B) integration applications has posed new requirements on software development. To address these requirements many new development paradigms have emerged. Real developers are faced with a wide array of technologies, tools, methods, and techniques, and have to decide which ones are important. The primary indicators point to "specifications" and (web) "services" as the new wave of approaches that need to be the foundation of every developer along with the well established Object Oriented approach. This article discusses these new requirements, and provides real developers guidance on making Specifications, Objects and Services part of their skill repertoire.

The first wave of e-business was all about B2C applications. This wave was characterized by a move from heavy GUI client applications to the "thin-client" model, and the move of application and business logic from the client to the server. Object Oriented principles (including encapsulation, polymorphism, and inheritance), programming model (especially the separation of interface from implementation), best practices (incuding design patterns and Model-View-Controller framework), and methods (use case driven and agile methods) provided the foundation for developing these applications. In addition, many "standards" were born during this time to support the need for portability, interoperability, and maintainability. Many touted e-business as being driven by these "standards", including TCP/IP for sharing resources across a network, HTML and HTTP for standard presentation and communication over the internet, XML for portable structured documents and data on the web, and Java for "write once, run everywhere" portable code.

The maturity of the OO approach along with this "specification" movement is best embodied in the Java2 Enterprise Edition (J2EE) component model which defines a set of specifications (http://java.sun.com/j2ee) for implementing and deploying enterprise applications. J2EE defines both client side (applets, application clients) and server side (servlets, Java Server Pages (JSPs) and Enterprise Java Beans (EJBs)) components as Java objects. It defines common distributed services include naming and directory

(JNDI), connecting to data stores (JDBC), security, and transactions (JTA and JTS), along with remote communication between components (RMI/IIOP) and messaging (JMS.) Finally, J2EE facilitates its components to be used in the MVC framework where EJBs embody the business logic (model), JSPs deal with the presentation (view), and servlets handle the interaction (controller). Note however that this J2EE model is very oriented towards developing B2C applications focusing on interfaces, tight client-server "conversation" coupling, and using a mostly synchronous communication.

The bursting of the "dot com" bubble has led us to the "second wave" of e-business. The basic realization in this second wave is that e-business is all about integration – integrating applications in the distributed enterprise, and business to business integration. The requirements of these integration applications include:

- Loose coupling between applications.
- Applications able to communicate regardless of endpoint implementation (i.e. platform, operating system, programming language, object model.)
- Mostly asynchronous communication between applications.
- Just-in-time application integration within and across enterprises.
- Interfaces for applications published and accessible.

Are distributed OO technologies and principles capable of handling these requirements? The handling of integration requirements through another set of specifications on top of the distributed OO technologies (e.g. J2C and JMS) and the focus on extending B2C applications is not sufficient. What is needed is a completely different approach, that tackles the need in an integral fashion. Enter web services and the service oriented architecture for integrating applications.

Web Services: Web Services are self-describing, self-contained, modular applications that have to be described, published, found, bound, invoked and composed. Web services conform to a service oriented architecture as summarized in Figure 1.

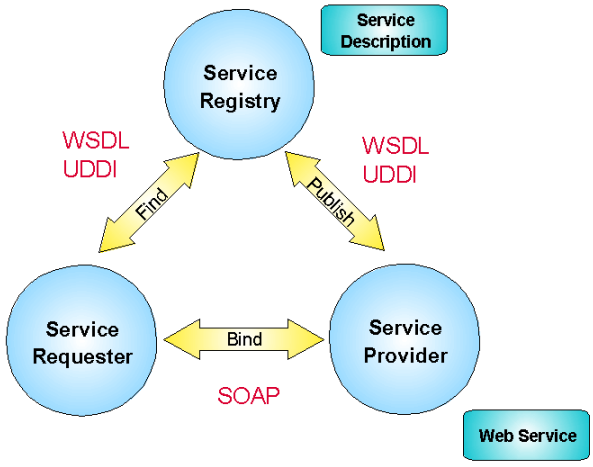


Fig. 1: The Service Oriented Architecture

As shown in the figure, service providers create application functions that are available on disparate implementation platforms as web services and describe them using a standard definition language. These services are published in a service registry**.** Service requestors who need a particular type of service search the service registry and find the desired service. Once a service is found the requestor and the provider of the service negotiate to access and invoke the service.

The underlying technologies supporting web services need to be platform and implementation neutral, and are therefore specified in XML. The core technologies include

- Simple Object Access Protocol (SOAP) is the XML based messaging specification that defines the message envelope content, the encoding rules for the data types, and conventions for defining requests and responses. SOAP is vendor neutral, and can support any language, programming model, and platform. Implementations of SOAP exist in Java, Perl, C/C++, and C#.
- Web Services Definition Language (WSDL) is the XML based description of a web service as a group of ports which are in turn defined by associating a network address with a reusable binding. Each binding defines a group of operations (port type) that is associated with a protocol. Each operation in turn is defined in terms of messages and types. WSDL facilitates the abstract definition of web services to be separated from their concrete implementations.
- The Universal Description, Discovery and Integration (UDDI) specifications define a way to publish and discover information about Web services. The core component of the UDDI is the UDDI business registration, an XML file used to describe a business entity and its Web services, and which are used to provide white pages (service provider), yellow pages (business categories), and green pages (technical binding information.)

What kind of applications will be built using web services? Primarily, the focus will be on integration – starting from integrating applications within the enterprise (happening now), moving to integrating the entire supply chain (in the next 6 months to a year), to a complete dynamic e-business (in 2-3 years.) However, with web services several new opportunities emerge, including the "pay as you go" subscription based pricing model (e.g. IBM's e-business on demand e-utility model http://www-1.ibm.com/services/ondemand/index_flash.html) and the emerging area of grid computing (http://www.globalgridforum.org/)

It is important to note that web services are at their infancy – specifications are still evolving, (competing) standards are being defined, critical enterprise issues (including scalability, performance, security, etc.) have to be defined and implemented, and best practices, frameworks, and design patterns have to be developed and used. Follow their evolution through the standards bodies involved including W3C (http://www.w3.org/2002/ws/), OASIS (http://www.oasis-open.org/committees/wsia/), WS-I (http://www.ws-i.org/), and the community portal http://www.webservices.org/

2 SKILLS IN THE BRAVE NEW WORLD

What skills are needed in this brave new world? Here is a summary of the core skills needed for each participant in the service oriented architecture:

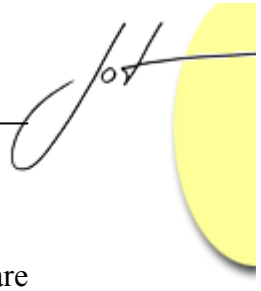
For the Service Provider:

- Discover and access an existing web services for use in building new web services using UDDI to find the web service, WSDL to understand how to access the web service, and SOAP to actually use the web service.
- Create a new web service which includes a WSDL based definition and the actual implementation of the web service. There are several approaches including a "green field" approach where the web service application and web service interface are built from "scratch", a bottom-up approach where a web service interface is built on an existing application, and a top-down approach where an implementation to an existing service interface is built.
- Deploy a web service to an application server which includes configuring the server to run the web service and installing the web service on the application server.
- Test a web service to validate its behavior which requires the development of a client proxy for the web service, and test clients that can use the proxy to invoke the web service. Testing should also validate that the web service can be searched for in the registry, the binding information is sufficient to allow a client to access the web service, and that the web service can be invoked from various clients.
- Publish a web service by registering it (along with the WSDL) into a UDDI service registry.

For the Service Requestor:

- Discover existing web services using the UDDI based services provided by a service registry.
- Access existing web services in many ways including via static binding (i.e. no run-time registry lookup), build-time dynamic binding (i.e. by generating a service proxy and allowing the proxy to locate a specific implementation), and run-time dynamic binding (i.e. the client environment will dynamically generate, build, and execute a specific implementation).

For the Service Broker:



- Establish a UDDI registry for web services. Both private and public registries are feasible. Public registeries have additional stringent requirements for ensuring that all the information in the public registries are in synch. Visit http://www.uddi.org for more details of how to set up a registry.
- Allow clients to perform basic operations on the UDDI registry, including publishing to a UDDI registry, searching a UDDI registry, and deleting from a UDDI registry.

3 SO HOW DO YOU GET S.O.S?

Almost all of the training organizations (both academic and industrial) are still focused in the distributed OO "camp", and very few have started to address SOS skills. Most training organizations do provide orthogonal tracks in developing XML skills, and some even have made the connection in combining XML and Java skills. However, the focus of the XML skills is primarily at addressing "presentation", i.e. the use of XML to represent information and using XSL to transform the information so it can be presented in a style that is appropriate to the device (e.g. web browsers, wireless phones, and voice.) There exists training targeted at message oriented middleware, which usually cover XML as the basis for the messages that need to be transported. So, how do you get SOS? Well, the answer depends on where you are as shown below.

Distributed Object Developers: You should have solid object skills, and need to concentrate on strengthening your specification skills and building your services skills. You should have a good foundation in specification skills as you are required to use well defined interfaces to build distributed applications. The skill that you need to build on top of this foundation is the ability to design a (document based) specification, and then build an implementation for the defined interface. The first step in developing these skills is to get a solid foundation in XML and its related technologies, and programming using XML and Java (or your language of choice.) After acquiring these core skills, you can move to building your web services skills. The primary focus of your skills will be on understanding the underlying technologies (SOAP, WSDL, and UDDI) and using these technologies in the context of the tasks that you need to perform (as defined above.) Note that building the programming skills will be very dependent on the development environment (including language, component model, and tools) that you choose to build your skills on. For example, a J2EE based development environment would require that you understand how to use the Java API for XML (JAX) based specifications for integrating web services with J2EE applications (http://www.jcp.org/aboutJava/communityprocess/review/jsr109). The main problem that you will face in building up your skills will be on how to design distributed objects so that they can be used as or in conjunction with web services. The main concern is that web services are not "objects", so an initial approach of simply mapping a distributed object into a web

service will not lead to a well-designed web service. Furthermore, the current OO methods and best practices (patterns, frameworks, etc.) do not apply very well to building web services. Unfortunately, since web service technology is still in its infancy, a lot of the work in building appropriate methods and best practices is ongoing and not well established yet. Note, however, that distributed objects and web services are complementary. So, you need to develop a good understanding of both worlds, and know how to use them in conjunction with each other. So, for example, when you have designed a web service, you will implement it using distributed objects.

Document-Oriented or Message-Oriented Developers: You should have solid skills in designing and manipulating documents, messages, and specifications primarily using XML and its related technologies. You can start building your web services skills by mastering the underlying technologies (SOAP, WSDL, and UDDI) and using these technologies in the context of the tasks that you need to perform (as defined above.) Most likely, you will rely on your core XML skills to build interfaces to and wrap existing applications so that they can be used as web services. The main problem you will face is in composing and integrating various applications together into a web service. You will need to get very good support from the underlying tools to help you in the "wrappering" and integration process, e.g. develop skills in using the technologies and tools for supporting web service development from http://www.alphaworks.ibm.com/webservices which includes support for wrappering and integration of web services.

Academic Institutions: The most recent computer science curriculum proposal from IEEE and ACM http://www.computer.org/education/cc2001/final/index.htm paves the way for finally establishing an OO based curriculum. The key for academic institutions adopting SOS is to evolve the proposed model to incorporate "document-oriented" XML based concepts and skills at the same introductory core level as "object-oriented" Java based concepts and skills are handled. The next step is to extend the coverage within each of the applicable intermediary courses (especially in the compressed and web-based approach) from component-based distributed object applications to service-based distributed integration applications, and from B2C to B2B applications. The advanced courses can support coverage of handling dynamic e-businesses, managed business processes, and the standards/specification process.

About the author

Mahesh Dodani is an e-business architect with IBM Global Services. His primary interests are in helping enterprises transfrom themselves from their current environment into being an e-business. He can be reached at dodani@us.ibm.com.