IBM Certified SOA Solution Designer certification prep, Part 1: SOA best practices

Skill Level: Intermediate

Mark Lorenz (mlorenz@nc.rr.com)
Senior Application Architect
Hatteras Software, Inc.

10 Oct 2006

Service-Oriented Architecture (SOA) is the next step in software development, leveraging XML technologies and Web services (WS) that went before. This best practices tutorial teaches you how to use SOA techniques in system design effectively. Use this tutorial, along with the other educational resources, to help prepare for IBM Certified SOA Solution Designer certification.

Section 1. Before you start

About this series

This tutorial will help you prepare for IBM certification Test 665: Architectural Design of SOA Solutions to attain IBM Certified SOA Solution Designer certification. This intermediate-level certification is intended for consultants and architects with experience designing enterprise application components, enterprise business integration solutions, and those who are part of an SOA project team responsible for architecting the end-to-end design of an SOA solution. SOA is the next step in software development, leveraging XML technologies and Web services (WS) that went before. This tutorial teaches the reader how to use SOA techniques in system design effectively.

About this tutorial
This tutorial takes you through key aspects of using SOA technologies for your software projects. We will cover best practices (what designs to use in which situations.) It’s written for software architects and business analysts who have a basic understanding of XML and Web services and who have advanced skills and experience. You should have a general familiarity with business analysis and software development methodologies.

This tutorial is also intended for developers who have a background in XML technologies and Web services. You should also be familiar with Internet standards and concepts such as Web browser, client-server, documenting, formatting, e-commerce, and Web applications. Experience designing and implementing .NET or J2EE-based computer applications, and working with relational databases is also recommended.

Objectives

After completing this tutorial, you will know how to recognize SOA best practices and their benefits.

Prerequisites

Experience in SOA, Web services, and XML.

System requirements

You need a system with an up-to-date browser.

You need JavaScript enabled in your browser.

Section 2. SOA Best Practices

A lot of hype surrounds SOA. And where there is hype, there is obfuscation. We will use buzzwords, but we will ground them with clear explanations and correlations so we can understand both their intent and meaning.

Not long ago, the world of software development began to see products supporting
model-driven development (MDD). SOA is arguably taking this direction further with what is called business-driven development (BDD), (see the Resources section for Mitra). See Figure 1.

**Figure 1. Business-driven development**

MDD moved the initial focus to software modeling and to the role of the software architect. BDD takes the initiative up another level to business modeling and the business analyst (BA) role.

Because SOA is still relatively new in the world of software development, best practices are still emerging and evolving. Therefore, for now, we sometimes refer to them as "guidelines".

Before we delve into the details of SOA best practices, we need to make sure we have some common language and definitions. So, let's take a brief look at XML, Web services, and SOA.

**XML and Web services**

As a brief refresher, XML is a lingua franca of the first level. It is an extensible tag language, understood across platforms and languages. It is used in many Web services standards. The contents of the tags are validated, or parsed, by a schema that defines the grammar.
Web services are functionality building blocks that can be reused. They must be published, found (discovered), and invoked by provider systems using standard protocols and semantics. This happens using XML with different grammars and related structures such as the following:

- WSDL
- UDDI
- SOAP

Figure 2 shows the relationship between these standards.

**Figure 2. Use of Web service standards**

Since Web services are standards-based, they provide a lingua franca of a second, higher level.

**WSDL**

Web services description language (WSDL) is an XML instance document that complies with a W3C standard XML grammar for communication between service requesters and service providers. It describes how a Web service works. It is because of WSDL files that Web services are called "self-describing" because
SOAP messages can be generated from WSDL files. In fact, many tools will create client code from a WSDL file.

A WSDL file contains the following elements:

- **Type**: Data-type definitions (string, int) using some grammar (such as an XML Schema)
- **Message**: The data being communicated
- **Part**: A message parameter
- **Operation**: Abstract description of an action supported by the service
- **Port Type / Interface**: An abstract set of operations supported by one or more endpoints. This name has changed, so you may see either one
- **Binding**: A concrete protocol and data format specification for a particular port type
- **Port / Endpoint**: A combination of a binding and a network address. This has also changed names, so you may see either one.
- **Service**: A collection of related endpoints, with their associated interfaces, operations, messages, etc

Figure 3 shows the WSDL structure as presented by Thomas Erl (see Resources section for more information).

**Figure 3. WSDL structure**
Again, remember that all the details needed to interact with a service are in its WSDL file.

**Universal Description, Discovery, and Integration (UDDI)**

UDDI defines how to find a Web service (and its WSDL file). UDDI hasn't gotten as much use as WSDL and SOAP because many times a consumer knows the location of Web services, which are often on the company intranet.

UDDI listings are kept in a UDDI registry. Each listing can include the following:

- **White pages**: Address, contact, and known identifiers
- **Yellow pages**: Industry categorizations based on standard taxonomies
- **Green pages**: Technical information about services exposed by the business

Green pages are all that's required. They give access to the WSDL information for the service.
Simple Object Access Protocol (SOAP)

SOAP is a protocol for exchanging XML-based messages across a network. Typically, HTTP is used as the transfer protocol, but other protocols such as SMTP can be used.

A SOAP message contains the following elements:

- **Envelope**: Is required and identifies the document as a SOAP message
- **Header**: Contains application-specific information
- **Body**: Is required and defines call and response information
- **Fault**: Contains information about errors that occurred

As we said before, the SOAP content may be determined by the WSDL file.

SOA

The primary goal of SOA is to create and maintain closer synergies between business and IT -- to make it easier to get functionality in a timely manner and in a cooperative fashion with the IT organization. SOA is about wiring the Web service building blocks together according to business processes so that functionality is provided from start to finish. SOA is a framework of the highest level -- corporate-wide and beyond.

You might wonder, "Why SOA"? Briefly, because it provides more flexibility and speed when building systems. The most widely cited benefit of SOA is to "remove business analysts from the shackles of the IT department." But another reason for interest in SOA was shown by Mansurov, as we see in Figure 4 (see Resources section for more information).

**Figure 4. Improvements in time to market**

Since SOA moves more focus and time to earlier phases of development, this study
showed that a 30% to 50% savings could be gained. That same study showed that 36% of defects were due to mistakes at requirements time. This bodes well for the future of SOA.

Now let's get to the real substance of this tutorial. We'll discuss categories of guidelines, starting with general SOA guidelines.

**General SOA guidelines**

**Follow standards**

You'll see "WS-*" to indicate a whole set of Web services standards. These and other SOA-related standards are an infrastructure necessity. If you vary from the standard, anyone trying to reuse your services will fail and won't come back. All standards are important, but let's go over a few of the major ones for SOA:

- **WS-I**: The protocol you use should be compliant with the Web Services Interoperability (WS-I) standard. HTTP, SOAP, and JMS are examples of a compliant messaging infrastructure.
- **WS-BPEL**: Your business processes should be defined via business process execution language for Web services (WS-BPEL). It's a long name, but this standard is necessary to document your process in a cross-product fashion.
- **WSDL**: As we said above, WSDL defines how a Web service works.
- **UDDI**: When used, UDDI registers a Web service so consumers can find it.
- **SOAP**: SOAP is a protocol for XML-based messaging over a network as defined by its WSDL file.
- **WS-***: There are many more that we will not discuss here, including WS-Coordination, WS-Notification, WS-Policy, WS-Reliability, WS-Resource, WS-Security, WS-Transaction, and others.

In order to truly have an SOA, you must follow at least these standards.
Follow SOA principles

It is worth discussing some of the key principles SOA is based on:

- **Loose coupling**: Services must remain aware of each other, but changes in location, version, and implementation should be transparent. Frameworks such as Spring are loosely coupled, basing wiring on declarative configuration files and aspect-oriented weaving, but SOA takes loose coupling to another level. In SOA, facilities such as the ESB adapt to change without involving client logic. And tools such as the WebSphere® Integration Developer make wiring easy.

- **Contractual interfaces**: WSDL is the standard way to define a Web service's interface and should be used to describe all Web services.

- **Reusable services**: There are multiple characteristics of reusability, including abstraction and coupling. A service should be defined at the smallest size possible that will still provide support for a complete business function. If you add more functionality, the service becomes less reusable since it doesn't work well with as many other services. Think in terms of defining atomic services.

- **Self-describing services**: To be reused, a component has to be able to be found. Stick to terms your business analysts can understand. Provide rich descriptions. Use WSDL files that fully describe your Web services.

- **Statelessness**: Services should not maintain state between invocations. They should be designed to be reentrant so that no problem with concurrent requesters occurs. Artus discusses issues related to why services need to be stateless (see Resources section for Artus). The bottom line? Stateful services are less reusable.

Service modeling guidelines

**Use business names for services**

The consumers of services are business analysts. As such, the names of services should use their terminology and should have meaning to them. So, for example, you should have services such as *Transfer Funds between Accounts* instead of *ProcessTransferTransaction*.
Many of the modeling techniques from object-oriented systems development can be used in this area. For example, modeling sessions with domain experts, adapted CRC cards, activity swimlanes, state transition diagrams, and other techniques are all useful on an SOA project to discover business services. In fact, Artus gives an example of how to use state transitions to discover business services (see Resources section for Artus).

Establish standard naming conventions within your organization. Check out some public UDDI registries to see how other companies are naming their services:

- BindingPoint
- XMethods

In general, the service operations should be verb phrases and the service itself should be a noun. This advice comes from years of object-oriented system development experience.

**Identify services wisely**

Not everything should be a service. When identifying services, be sure to do the following:

- Mine existing systems for candidate services.
- Focus on business value.
- Balance course- and fine-grained size.

Ali Arsanjani states that "using goal-service modeling, you use a cross-sectional approach to cut down the sheer number of candidate services that might already be identified. A more judicious approach would be to first do top-down, then goal-service modeling, and finally bottom-up legacy analysis of existing assets" (see the Resources section for Arsanjani). He defines goal-service modeling as basically the use of a business’ goals to identify services needed to satisfy those goals.

Watch for granularity and value-add. For example, if you are making a legacy system available through a set of services, it may be better to keep larger "chunks" of the functionality together if they are tightly coupled. Focus on complete functionality that works independently of other services.

Fine-grained services tend to be more reusable but at a price in orchestration effort. More services means more work for a consumer to wire a business process. On the other hand, course-grained services are at risk of being less reusable while providing a larger set of functionality (and thus less orchestration effort). So what's
an architect to do? The answer is an unsatisfying, "use your judgment on case-by-case basis." However, following are some rules of thumb:

- Subdivide even further when you see portions of a service candidate that can provide stand-alone value to consumers. Keeping these portions together makes the overall service less reusable since not all consumers will want all the functionality provided.

- Stop moving in a more granular direction when you reach a point of high coupling (sometimes called "cohesion") where further subdivision will separate highly coupled functionality. Further subdivision will not deliver a complete solution for consumers.

Finally, keep in mind that increased granularity means more messaging -- potentially across a network -- which, of course, has performance implications.

---

**Choreograph**

Despite the name, this has nothing to do with dancing. Web services are choreographed into composite applications by wiring their interfaces together to support a business process. The process has business rules as part of its definition. Orchestration is similar to choreography, but choreography spans multiple interested parties, while orchestration focuses on one participant.

---

**Follow a service-oriented methodology**

SOA is relatively new but there are already some proposed methodologies. These are often combinations and extensions of other existing methodologies. That's fine -- it's important to have a process and techniques to follow. We'll now take you through some proposed methodologies for SOA.

- **Methodology for Service Architectures**: (Jones, 2005) focuses on service discovery and provides a notation for the methodology. This methodology is focused on the earliest activities of SOA development. Figure 5 shows one of the techniques used in this methodology. This methodology can be a supplemental method, but doesn't have an entire set of steps for a whole SOA project.

---

**Figure 5. Primary tasks**
Service-Oriented Modeling Architecture (SOMA): According to Ali Arsanjani (see Resources section for link to Arsanjani), "Service-oriented modeling approach provides modeling, analysis, design techniques, and activities to define the foundations of a SOA. It helps by defining the elements in each of the SOA layers and making critical architectural decisions at each level. It does so using a combination of a top-down, business-driven manner of service identification coupled with a stream of work that conducts service identification through leveraging legacy assets and systems."

Service-oriented analysis and design (SOAD): Zimmermann makes the case for SOAD by stating that: "While the SOA approach reinforces well-established, general software architecture principles such as information hiding, modularization, and separation of concerns, it also adds additional themes such as service choreography, service repositories, and the service bus middleware pattern, which require explicit attention during modeling" (see Resources section for Zimmerman). This methodology includes portions of business process modeling (BPM) and object-oriented analysis and design (OOAD), expanding upon them to identify services and to synchronize BPM and OOAD artifacts. The methodology addresses the divide between WS-BPEL and WSDL, and addresses other topics, such as providing for

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
<th>Pre-condition</th>
<th>Post-condition</th>
<th>Invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ship order</td>
<td>Request for an order to be shipped</td>
<td>Goods on the order are available</td>
<td>Order is assembled ready for delivery, delivery is scheduled with the client</td>
<td>Price of goods</td>
</tr>
<tr>
<td>Add to stock</td>
<td>Add new items into stock</td>
<td>Item is available and is a valid and known product</td>
<td>Items are added to the stock count</td>
<td>Stock of items unrelated to the new items</td>
</tr>
<tr>
<td>Deliver</td>
<td>Request an external company, or internal logistics, to supply an order to a customer</td>
<td>Goods available for shipping</td>
<td>Goods removed from warehouse and placed on appropriate delivery route</td>
<td>Items on order</td>
</tr>
<tr>
<td>Supply</td>
<td>Receive supplies</td>
<td>Order had been placed with supplier, or supplier is managing own inventory</td>
<td>Inventory of supplied item is increased by amount supplied</td>
<td>Stock of items not related to this supply order</td>
</tr>
</tbody>
</table>
quality of service (QoS). Being a long-term OO modeler, I was comfortable with the use of OOAD techniques, including UML. SOAD sees OOAD as too tightly coupled and granular for SOA, and so uses it as a more granular layer (see Figure 6).

**Figure 6. SOAD design layers**

![SOAD design layers diagram]

SOAD focuses on higher-level concerns at a business level, such as business processes. As the development becomes more detailed, existing object-oriented techniques are used. SOAD includes SOA-specific topics such as:

- Service categorization and aggregation
- Policies and aspects
- Meet-in-the-middle processes
- Semantic brokering
- Service harvesting and knowledge brokering

- **Service-oriented modeling**: Mark Endrei starts with business decomposition, which identifies business use cases and processes (see Resources section for Endrei). When the use cases and processes move into the design phase, system use cases and subsystems to support the processes are identified. The business use cases are then used to identify services. The next step is to identify business goals and to break them down into subgoals (see Figure 7). The subgoals then map previously identified services to the goals, thereby revealing any gaps. The goal is to identify business components, so the next step is to analyze processes within the
Ali Arsanjani expounds on the methodology of SOMA (see Resources section for Arsanjani), detailing techniques for discovering services. He calls his top-down process “domain decomposition,” which results in business use cases. He calls his bottom-up process “existing system analysis,” which results in candidate functionality to turn into components. Finally, he calls his middle-out process “goal-service modeling,” which identifies potential services through a focus on business goals and performance metrics. This is well-documented SOA methodology because it includes formats for artifacts, and so may be a good place to start.

- **Agile service-oriented modeling**: Pal Krogdahl focuses on modeling the business' services as well as linking business processes to services (see Figure 8). He builds on the work of Zimmermann and Arsanjani (see Resources for Krogdahl). His value-add is to look at agile methods as applied to SOA.

Figure 7. Domain decomposition

Figure 8. SOA elements
There is no clear industry winner for SOA methodology. Until there is, choose one and adapt it to your organization. Past history says that some blending and merging of methodology proposals will end up as the de facto standard.

Service design guidelines

We listed key principles for an SOA and your designs should support these principles. We'll turn now to some design-specific advice.

Discover services based on key business initiatives

This may sound obvious, but it's worth emphasizing: One of the key goals of SOA is to have business analysts and technical staff (IT) work more cohesively. In the past, analysts were constrained by what the IT staff told them was "possible." Just as use cases have been used to drive IT efforts, business cases provide that direction in an SOA.

Some SOA methodologies use business goals to drive activities and develop artifacts. Of course, this means you understand your business’ initiatives. If you don’t, start there; if you do, use them to discover services as a team effort between business analysts and IT personnel.
Generate message details from WSDL

Interactive Development Environments (IDE) such as IBM Rational Application Developer (RAD) will generate Web service code from a WSDL file. There is no reason to type this information in by hand.

Optimize SOAP messages

SOAP content is XML. As you know, XML is descriptive but is typically larger due to the volume of tags. If you have performance problems, consider using Message Transmission Optimization Mechanism (MTOM), which builds on XML-binary Optimized Packaging (XOP). XOP provides a more efficient serialization of XML by using MIME encoding. MTOM uses XOP to encode portions of SOAP messages, while still presenting an XML Infoset to the SOAP application.

Another, more recent alternative is to use XFire. XFire is an open source, standards-compliant SOAP framework that uses the latest technologies, including a StAX parser, to boost SOA messaging performance.

Again, unless your performance is unacceptable, you should be fine with regular XML SOAP messages. However, if you are providing services to a large community of clients, you may need to boost your performance using one of these alternatives.

Provide atomic services where transactions are needed

Try to make service requests as atomic as possible. One service request should stand on its own so that if it needs to roll back, it can do so independently of other services.

For example, if you have a banking application and need to service a funds transfer, do that with a single `transferFunds(fromAccount, toAccount, amount)` service. If you only provide services for the component parts of a transfer, such as `deposit(anAmount)` and `withdraw(anAmount)`, you are asking for trouble. Not only is it more work for all your consuming applications, it is an accident waiting to happen when the withdrawal works but the deposit fails (or vice versa). The atomic services will include all the steps in a single transaction, ensuring an all-or-nothing approach to the steps.

This becomes especially interesting when you consider that the `fromAccount` and...
toAccount can be at different financial companies. So, how does a (logical) transaction work across companies? Jon Maron presents more complex (and realistic) examples of the issues in the world of distributed services and he suggests a solution (see Resources section for Maron). The solution requires more work, as you might expect, but meets the transactional requirements, as shown in Figure 9.

**Figure 9. Transaction support**

Use small, high-impact proof-of-concept services

Making services available can be done incrementally. You don't have to make everything a service at the same time. Start with an area of major benefit. For example, on my current job, a central part of our systems is a legacy system that is, well, to put it nicely, difficult to work with. So, we have attacked that issue first because it's an area ripe for creating services for groups across the corporation.
Reduce coupling between consumers and providers

During choreography, requester processes, Web services, and systems will be wired into your provider services. In and of itself, this is coupling. The difference is that ESB facilities can route to updated services, handle conversion between formats, and other necessary details. This is as loose a coupling as you can get and still reuse available services. Loose coupling is especially important when you are reusing services outside your organization.

Dave Artus lists the following set of dependencies between service consumers and providers (see Resources section for Artus):

- **Platform**: Web services provide platform independence
- **Location**: SOA ESB provides dynamic routing at runtime
- **Availability**: SOA ESB supports asynchronous invocation
- **Version**: SOA ESB mediates format changes

Be consistent and concise in your service invocation

If your services work the same way, choreographing a business process will be easier for your business analysts. In addition, if you make them all work asynchronously, you remove a dependency on service availability.

Conciseness relates to what Brad Cox calls "surface area": group parameters into a single structure so the consumer only needs to provide a map structure that makes it easy to send values to the service safely. If the service changes, the same structure can be supported, since it is still a map. This would allow you to deprecate the previous parameter set while still supporting it and not breaking the service for current consumers!

Use tools

This may be obvious, but SOA is supposed to be responsive and flexible, so you need tools that work well in an SOA effort. You could use IBM's SOA tool suite, which includes WebSphere Business Modeler, Rational Application Developer (RAD) and WebSphere Integration Developer (WID) products. These products let you develop business processes and Web services interactively and then wire them together into a deployable SOA application. You will need a registry and repository. IBM's WebSphere Service Registry and Repository (WSRR) is one possible solution.

Without tools that are geared toward SOA development, you will find it difficult to be successful. Remember, SOA is a lot more than Web services.
Service integration guidelines

Designate an interface designer

You must follow standards, just as you must strive for consistency across services. According to Thomas Erl (see Resources section for Erl), the interface designer is responsible for creating the WSDL files for Web services as well as the formatting of related SOAP messages.

Erl goes on to say that this interface designer should have component design and business analysis skills -- someone needs to be dedicated explicitly to this role.

Service management guidelines

Monitor SOA QoS

Monitoring is a challenge in an SOA because the assets are potentially scattered around the network, including outside your organization. Traditional monitoring tools are great at looking at performance QoS across a single-address space or server, but they don't help much when monitoring Web services that span servers and possibly companies. These composite applications are typical in SOAs.

Monitoring is a part of the control and feedback mechanism called SOA governance. Monitoring composite applications has unique end-to-end management issues (see Resources section for Cappelli). One example of this complex issue we discussed earlier was transactions within a SOA.

Monitoring QoS merely at the UI level won't uncover the underlying causes of any issues that arise, such as performance degradation. The IBM Tivoli Composite Application Manager for SOA supports monitoring for an SOA.

SOA Governance

The word governance is used extensively in the world of SOA. Governing, as we know, deals with exercising authority or control over something or someone. In discussing SOA, it means that service use and registration need to be managed and monitored in a standard way. This has been a weak point for delivering reusable services.

Governance includes all stakeholders in the decision-making process. All stakeholders, not just IT, are responsible for key systems.

IBM's WebSphere Service Registry and Repository (WSRR) product is a specific example of support for SOA governance. It defines a well-known location to keep and access information about Web services.
Designate a registry administrator

Services reside in a registry or repository. These facilities need to be reliable. Otherwise potential users of registry services lose confidence. Besides dealing with availability product registry, the administrator monitors usage and errors and deals with any other issues.

The registry administrator needs to be proficient in service monitoring tools, such as IBM's WebSphere Business Monitor. These tools collect key performance indicators (KPI) and help detect anomalies in the running SOA.

IBM also has a tool called the IBM WebSphere Services Registry and Repository that allows for publishing and finding services across your ESB. See resources for more information

SOA staffing guidelines

Mentor developers in the SOA process

Web services must take into account requirements from multiple use cases. Traditionally, developers work on requirements. To be more useful (in other words, reusable), build components with future and alternative requirements in mind.

Eric Pulier suggests adding a step to the process: "Brainstorm alternative use cases and discuss with potential stakeholders." The team can then adjust their follow-on efforts to take into account expanded and future requirements (see Resources section for Pulier).

Section 3. Conclusion

SOA can take your organization to the next level of integration and productivity, if used wisely. The key to your success is understanding the pitfalls along the way. Knowing SOA modeling, design, integration, and management is key to successful introduction of SOA to your business.
Resources

Learn

- Get an understanding of how an SOA would fit in your organization in *Understanding Enterprise SOA* by Eric Pulier and Hugh Taylor (Manning Publications, 2006).


- Read about developing Web services for your SOA in the IBM Redbook "Patterns: Service-Oriented Architecture and Web Services" (April 2004) by Mark Endrei, et al.

- Learn about modeling for your SOA in "Service-oriented modeling and architecture" (IBM developerWorks, November 9, 2004) by Ali Arsanjani.

- Learn about the seven levels of maturity on the path to SOA adoption in "Increase flexibility with the Service Integration Maturity Model (SIMM)" (IBM developerWorks, Sept 2005) by Ali Arsanjani and Kerrie Holley.

- Get tips on designing your Web services in "SOA realization: Service design principles" (IBM developerWorks, February 2006) by David J. N. Artus.

- Read about SOA maturity levels in David Linthicum's "What level is your SOA?" (SOA Web Services Journal, December 2, 2004).

- "A Methodology for Service Architectures" by Steve Jones and Mike Morris (OASIS, October 26, 2005) describes a SOA methodology.


- Learn the SOAD methodology in "Elements of Service-Oriented Analysis and Design" (IBM developerWorks, June 2004) by Olaf Zimmermann, et al.

• Learn how to improve SOA messaging performance and other SOA tools in "XXFire: The easy and simple way to develop Web services" (JavaWorld, May 1, 2006) by Shahid Ahmed.

• Read about the latest way to parse XML in "Does StAX Belong in Your XML Toolbox?" (developer.com, August 20, 2004) by Jeff Ryan.

• Find out what BDD is all about in "Business-driven development" (IBM developerWorks, December 2005) by Tilak Mitra.

• See how to deal with transaction issues in an SOA in "Transaction Processing in Distributed Service-Oriented Applications" (informit.com, November 24, 2004) by Jon Maron.

• Read about SOA governance issues such as monitoring and QoS in "Composite Applications and the Application Management Life Cycle" (June 2005) by Will Cappelli.

• Read a lively discussion about issues surrounding SOA in "Getting a little closer to SOA" (DotNetGuru, January 2004) by Fabrice Marguerie.

• Stay current with developerWorks technical events and webcasts.

Get products and technologies
• Download a free trial version of WebSphere Application Server Version 6.0.

• Build your next development project with IBM trial software, available for download directly from developerWorks.

• Learn how the WebSphere Service Registry and Repository helps increase the business value of your service oriented architecture.

Discuss
• Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Mark Lorenz
Mark is the founder of Hatteras Software, an OO consulting firm, and the author of multiple books on software development. He is certified in OOAD, XML, RAD, and Java. He uses XHTML, Web services and SOA, Ajax, JSF, Spring, BIRT and related Eclipse-based tools to develop Java enterprise applications. You can read Mark’s blog on technology at marklorenz.blogspot.com.