

SOA Implementation Recipes, Tips, and Techniques

Java SOA Cookbook



O'REILLY®

Eben Hewitt

Java SOA Cookbook



Java SOA Cookbook offers practical solutions and advice to programmers charged with implementing a service-oriented architecture (SOA) in their organization. Instead of providing another conceptual, high-level view of SOA, this cookbook shows you how to make SOA work. It's full of Java and XML code you can insert directly into your applications, and recipes you can apply right away.

The book focuses primarily on the use of free and open source Java Web Services technologies—including Java SE 6 and Java EE 5 tools—but you'll find tips for using commercially available tools as well.

Java SOA Cookbook will help you:

- Construct XML vocabularies and data models appropriate to SOA applications
- Build real-world web services using the latest Java standards, including JAX-WS 2.1 and JAX-RS 1.0 for RESTful web services
- Integrate applications from popular service providers using SOAP, POX, and Atom
- Create service orchestrations with complete coverage of the WS-BPEL (Business Process Execution Language) 2.0 standard
- Improve the reliability of SOAP-based services with specifications such as WS-ReliableMessaging
- Deal with governance, interoperability, and quality-of-service issues

The recipes in *Java SOA Cookbook* will equip you with the knowledge you need to approach SOA as an integration challenge, not an obstacle.

www.oreilly.com

US \$49.99

CAN \$49.99

ISBN: 978-0-596-52072-4



5 4 9 9 9

“The examples are clear, well documented, and well thought out. It’s excellent work that I’m sure I’ll dig into for answers for at least a few years to come.”

—Jason Brittain,
Author of *Tomcat:
The Definitive Guide*

Eben Hewitt is a technical architect focused on designing and building SOA at a national retailer. The author of several programming books, including *Java for ColdFusion Developers* (Pearson), he is also a contributor to O'Reilly's 97 *Things Every Software Architect Should Know*.

Safari®
Books Online

Free online edition
for 45 days with
purchase of this book.
Details on last page.

Java SOA Cookbook

Eben Hewitt

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Taipei • Tokyo

Java SOA Cookbook

by Eben Hewitt

Copyright © 2009 Eben Hewitt. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safari.oreilly.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editor: Simon St.Laurent

Production Editor: Loranah Dimant

Copyeditor: Emily Quill

Proofreader: Loranah Dimant

Indexer: Lucie Haskins

Cover Designer: Karen Montgomery

Interior Designer: David Futato

Illustrator: Robert Romano

Printing History:

March 2009: First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Java SOA Cookbook*, the image of a harlequin longhorn beetle on the cover, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.



This book uses RepKover™, a durable and flexible lay-flat binding.

ISBN: 978-0-596-52072-4

[M]

1236962685

Table of Contents

Preface	xiii
---------------	------

Part I. SOA Fundamentals

1. Introduction to SOA	3
1.1 Defining a Service	4
1.2 Defining SOA	9
1.3 Identifying Service Candidates	12
1.4 Identifying Different Kinds of Services	16
1.5 Modeling Services	18
1.6 Making a Service Composable	20
1.7 Supporting Your SOA Efforts	23
1.8 Selecting a Pilot Project	30
1.9 Establishing Governance	33
2. XML Schema and the SOA Data Model	39
2.1 Designing Schema for SOA	41
2.2 Creating Canonical Data Model	48
2.3 Using Chameleon Namespace Design	52
2.4 Versioning Schemas	54
2.5 Reference Schemas	57
2.6 Common Schema Types	57
2.7 Validating an XML Document Against a Schema	61
2.8 Validating an XML Document Against Multiple Schemas	64
2.9 Restricting Schema Types with Regular Expressions	65
2.10 Using Schema Enumerations	67
2.11 Generating Java Classes from Schema	68
2.12 Generating a Schema from Java	75
2.13 Generating Java Source Files from XML Schema in Ant	80
2.14 Generating an XML Document Instance from a Schema	81
2.15 Customizing How a Java Class Is Generated from Schema	83

2.16	Validating Against a Schema During Marshaling and Unmarshaling	88
2.17	Collecting Schema Validation Events During Marshaling and Unmarshaling	91
3.	Working with XML and Java	95
3.1	Reading an XML Data Stream	96
3.2	Writing XML Data Streams	102
3.3	Filtering Data in an XML Stream	104
3.4	Selecting Values from an XML Document	107
3.5	Updating a Value in an XML Document	112
3.6	Converting a Java Object into an XML Document Instance	114
3.7	Converting an XML Document Instance into a Java Object	118
3.8	Generating a Schema from an XML Document	120
3.9	Converting XML to Java Without JAXB	122
3.10	Customizing Code Generation in JAXB	124
3.11	Finding the JAR That Contains a Given Class on Linux	125
3.12	Transparently Substituting XML Files	127

Part II. Web Services

4.	Getting Started	133
4.1	Using Publicly Available Web Services to Test Against	134
4.2	Installing Metro	140
4.3	Installing Oracle WebLogic	143
4.4	Creating and Deploying the Simplest Web Service	145
4.5	Creating and Deploying a Service to WebLogic	147
4.6	Setting Up a Maven 2 Service and Client Project	149
4.7	Understanding WSDL	155
4.8	Using References in NetBeans to Generate Web Service Clients	158
4.9	Monitoring SOAP Traffic with Metro	161
4.10	Monitoring SOAP Traffic with TCPMon	164
5.	Web Services with SAAJ	169
5.1	Creating a SOAP Element with a Qualified Name	173
5.2	Creating a Complete SOAP Message	175
5.3	Writing a SOAP Response to an Output Stream	178
5.4	Creating a Web Service Client Based on an Existing SOAP Envelope	180
5.5	Extracting Content from a SOAP Message	184
5.6	Creating a Web Service Client Using Raw XML Source and DOM	186
5.7	Adding a MIME Header	189
5.8	Adding Namespace Declarations	189
5.9	Specifying SOAPAction	190

5.10	Adding an Attribute to an Element	195
5.11	Removing a Header from a SOAP Message	197
5.12	Adding Headers to a SOAP Request	198
5.13	Accessing All SOAP Header Elements	206
5.14	Adding an Attachment to an Outbound SOAP Message	208
5.15	Accessing Inbound Attachment Data	209
5.16	Connecting to a SAAJ Endpoint Without a WSDL	210
5.17	Working with SOAP Actors	211
5.18	Asynchronous Invocation with Dispatch	213
5.19	Validating Your Payload Against a Schema on the Client	214
5.20	Providing a Web Service with SAAJ	221
5.21	Sending and Receiving SOAP Faults	226
6.	Creating Web Service Applications with JAX-WS	231
6.1	Calling a Web Service from the Command Line	234
6.2	Using JAX-WS Annotation Name Properties	238
6.3	Invoking the Simplest Web Service	241
6.4	Creating a Client Proxy	243
6.5	Consuming a Web Service from a Servlet or EJB	249
6.6	Consuming a Web Service from a JSP	253
6.7	Using a JAXB-Annotated Instance in a SOAP Message	254
6.8	Using wsimport in a Maven Project	257
6.9	Dealing with Version Errors in wsgen and wsimport	259
6.10	Adding Headers to a SOAP Request	261
6.11	Intercepting the Request to Perform Protocol-Specific Work	273
6.12	Intercepting the Request to Perform Work on Your Payload	280
6.13	Sharing Data Between Handler Invocations	281
6.14	Passing Binary Data in a Request	286
6.15	Using Binary Data in a SOAP Message	287
6.16	Enabling Binary Optimization on the Client	290
6.17	Validating a SOAP Payload Against a Schema with Metro	291
6.18	Making Asynchronous Calls with a JAX-WS Client	294
6.19	Overriding the Endpoint Address in an SEI	300
7.	Providing SOAP-Based Web Services	303
7.1	Assembling a Service for Deployment	304
7.2	Determining a Service Development Model	306
7.3	Choosing Encoding, Use, and Parameter Styles	314
7.4	Generating a WSDL and Portable Artifacts Based on a Java Service Endpoint Implementation	319
7.5	Creating a Basic Web Service	324
7.6	Specifying Namespaces	328
7.7	Creating a Web Service Operation	328

7.8	Specifying a Web Service Message Part	329
7.9	Specifying an Operation Return Value	330
7.10	Defining Zero-Argument Operations	331
7.11	Defining Operations with Void Return Type	332
7.12	Creating a Web Service That Uses Complex Types Based on Custom WSDL and a Custom Schema	334
7.13	Specifying the SOAP Binding Style, Use, and Parameter Style	349
7.14	Configuring Standard Custom Bindings	349
7.15	Excluding a Public Method from a Service	352
7.16	Creating a Service Provider with an XML View	353
7.17	Implementing Server-Side Handler Chains	361
7.18	Providing Stateful Services	364
7.19	Adding a Header with a Method Parameter	367
7.20	Accessing Incoming Header Parameters in a Service	369
7.21	Providing a Value for SOAP Action or WS-Addressing Action	369
7.22	Optimizing Transmission of Binary Content on the Server	371
7.23	Getting and Sharing Data About Users and the Request	372
7.24	Using Header References with Holder<T>	373
8.	RESTful Web Services	379
8.1	Creating a POX over HTTP Service with Servlets	390
8.2	A RESTful Service with JAX-WS	395
8.3	Creating a Client for a RESTful Service Using Sockets	396
8.4	Application: Using SSL, Atom Publishing, and the Google Finance REST API	398
8.5	Setting Up the Jersey JAX-RS Implementation	409
8.6	Creating a Jersey Project with Eclipse and Tomcat	412
8.7	Creating Hello World with Jersey	413
8.8	Creating a Single Path for Variable Resources of the Same Type	415
8.9	Restricting the Structure of Values in a Path Template	417
8.10	Accessing Query Parameters	418
8.11	Marshaling a Custom Type to XML in a Response	419
8.12	Offering Different Representations of the Same Resource	422
8.13	Creating a Resource	430
8.14	Working with Forms and URIs	435
8.15	Using SAAJ to Access a RESTful Service	443
8.16	Setting Metadata on Representations	445
8.17	Deleting a Resource	448
8.18	Redirecting to Another Service	448
8.19	Accessing HTTP Headers	451
8.20	Working with Cookies	454
8.21	Working with Exceptions and Response Status Codes	458
8.22	Working with WADL	462

Part III. Business Processes

9. Service Orchestrations with BPEL 477

- 9.1 Determining a Process Design Method 481
- 9.2 Selecting a Business Process Language 483
- 9.3 Getting Apache ODE BPEL Engine 485
- 9.4 Deploying a Process to Apache ODE 486
- 9.5 Understanding BPEL Process Basics 489
- 9.6 Using a Free Graphical Designer to Create BPEL Processes 494
- 9.7 Creating a BPEL Process That Invokes a Partner 496
- 9.8 Deploying a BPEL Process to OpenESB’s BPEL Service Engine 519
- 9.9 Testing a Deployed BPEL Process 522
- 9.10 Installing Active Endpoints BPEL Designer 525
- 9.11 Installing Active Endpoints BPEL Engine 526
- 9.12 Creating a BPEL Process in Active Endpoints Designer 528
- 9.13 Deploying a Process to Active Endpoints Server 530
- 9.14 Using Web Service Partners 531
- 9.15 Invoking a Partner Service from a BPEL Process 532
- 9.16 Manipulating Data with BPEL Variables 534
- 9.17 Using Literals 537
- 9.18 Concatenating Values 538
- 9.19 Choosing an Activity to Execute Based on Runtime Conditions 539
- 9.20 Executing Multiple Activities in a Sequence 541
- 9.21 Using Logical Divisions to Group Activities 543

10. Advanced Orchestrations with BPEL 547

- 10.1 Executing Activities in Parallel 547
- 10.2 Synchronizing Activities Executing in Parallel 548
- 10.3 Doing Nothing 550
- 10.4 Executing an Activity at a Specific Point in Time 551
- 10.5 Executing an Activity After a Specific Delay 552
- 10.6 Selective Event Processing 553
- 10.7 Handling Faults 554
- 10.8 Explicitly Throwing a Fault 557
- 10.9 Stopping a Process 559
- 10.10 Performing an XSL Transformation on BPEL Message Data 559
- 10.11 Validating Inbound Message Data 561
- 10.12 Correlation Sets 562
- 10.13 Looping 567
- 10.14 Adding Human Tasks to a Business Process 570

10.15 Invoking a RESTful Web Service from BPEL	572
--	-----

11. SOA Governance	573
11.1 Assigning Roles	574
11.2 Creating a SOA Roadmap	576
11.3 Keeping Track of Your Services	580
11.4 Determining a Data Ownership Scheme for Services	582
11.5 Handling Legacy Programs and Heterogeneity Within Your SOA	585
11.6 Documenting Services	586
11.7 Setting Up a Service Registry	588
11.8 Packaging Related Services	594
11.9 Retiring a Service	595
11.10 Browsing a UDDI Registry	595
11.11 Querying a UDDI Registry Programmatically	596
11.12 Understanding SOA ROI	598

Part IV. Interoperability and Quality of Service

12. Web Service Interoperability	607
12.1 Dealing with Arrays	608
12.2 Abstracting Addressing	610
12.3 Using Addressing in a Java Service	612
12.4 Explicitly Enabling Addressing on the Client	614
12.5 Explicitly Disabling Addressing on the Client	617
12.6 Abstracting Addressing in the Transport Layer from WSDL	618
12.7 Addressing Faults	618
12.8 Creating a .NET Web Service Client in Visual Studio	623
12.9 Creating a .NET Web Service Client in C#	624
12.10 Creating a .NET Web Service	631
12.11 Creating a Ruby Client for a Web Service	637
12.12 Creating a Ruby Client for a .NET Service	639
12.13 Conforming to the Basic Profile	641
12.14 Automating Testing for Basic Profile Conformance	644
12.15 Interoperability Best Practices	645
12.16 Using Modular WSDLs	646
13. Quality of Service	649
13.1 Understanding Reliable Messaging	651
13.2 Configuring a Java Web Service with Reliable Messaging	659
13.3 Configuring a Java Client with Reliable Messaging	660
13.4 Configuring a Java Web Service with Reliable Messaging on WebLogic	661

13.5 Using a WebLogic Reliable Messaging Error Handler	663
14. Enterprise Service Bus	665
14.1 What Is ESB?	666
14.2 ESB As a Set of Patterns	672
14.3 JBI	673
14.4 Commercial ESBs	678
14.5 Open Source ESBs	685
Index	693

Preface

Overview

I have heard it said that SOAs are like snowflakes—no two are alike. That is the case because a primary purpose of a service-oriented architecture is to offer a loosely coupled architecture for enterprise integration, and the internal landscapes differ so widely from one enterprise to the next. Additionally, SOA is about designing interfaces from a business perspective, which has historically been left up to developers.

This presents certain challenges for an author attempting to illustrate implementation choices and best practices. Many books have surfaced in the last few years that cover the general idea of SOA from an architect's or manager's perspective. These offer a conceptual picture of SOA, but not an in-the-trenches view.

Books that address SOA from an architect's perspective frequently offer little more than laundry lists of important and upcoming WS-* specifications. These books, while successful in sorting out the abstract ways in which an SOA can be built, do not tell a programmer/architect what to actually type to make things work. That is, many SOA books might tell you *what* you're supposed to do, but not *how* to do it.

They might, for example, indicate that you should make a composite service. It all sounds very convincing and important. But then you get back to your desk, fire up your IDE, and realize that you don't know what to type. Some books may go further, offering a syntax overview of XML-based languages such as BPEL, but then exit stage left before telling you how to really use it. It's hard to fault these books because the very nature of SOA means that you can build it with such a wide variety of tools.

My goal with this book is to show you how to really use some of the basic building blocks of SOA: web services, orchestrations, policies, and more. It is intended to fill in the gaps for developers in the real world. But to do this, I had to get concrete. And to keep the book to a manageable length, I had to focus the spotlight on what matters most, and leave some things out.

The focus of this book is on the following:

SOAP-based web services

A SOAP web service in .NET or Java EE 5 is a component whose annotations generate an XML description of the services it offers (called a WSDL). This description is not specific to the platform your component is written in, so a client written in another language can invoke your service.

This makes SOAP-based services an important part of SOA, and much of this book is devoted to using XML and SOAP web services and the Java APIs that support them.

It can be complex to use SOAP initially (as opposed to something like POX over HTTP), which gives SOAP its detractors. Vendors implement the SOAP standards for a variety of decorating features (including reliability, security, location transparency, and so forth); there's a lot of value in getting these features in a standard, interoperable way.

And while you may recall a number of web services books published some years ago, things have changed considerably. Creating SOAP web services in Java SE 6 and EE 5 is an entirely different animal. This book covers the very latest material, and doesn't stop at creating web services. It shows you how to bring them together in a real-world way.

RESTful web services

REST (REpresentational State Transfer) is a way of building on the architecture of the World Wide Web that is opposed to SOAP, at least in the popular mind. We examine this argument in the REST chapter, and I then offer fairly complete coverage of how to create RESTful services in a variety of ways, including using the new JAX-RS specification, JSR 311 (the Java API for RESTful web services). I also cover how to use popular APIs that have become de facto standards for REST, such as the Atom Publishing Protocol.

Java EE 5

While I show examples of consuming web services from languages other than Java, the overwhelming bias is toward implementing SOA with Java. The reason for that is simple: my background and area of expertise are in Java.

I do not address how to write web services with anything earlier than Java EE 5 and Java SE 6, as there are a number of books that cover how to do that. There have been considerable changes in web services in the latest version of Java. Annotations, new APIs, and burgeoning implementations of various WS-* specifications mean real changes for web services in just the last year or so. This book sticks with the latest stuff.

SOA

This cookbook is unusual among O'Reilly cookbooks in one respect: the solutions are not always code examples because SOA problems are not always code problems. The predominant focus of the book is how to implement Java EE 5 web

services and work with related technologies such as BPEL orchestrations and WS-* specifications. The chapters covering these topics offer concrete, real-world code examples that indicate what to type to make something work. In that respect, this book is like other O'Reilly cookbooks. But, where possible, the book also offers solutions to the “people problems” of SOA, such as organization and ROI. These chapters do not involve code solutions because they are not code problems. I have tried to restrict these topics to those that have a clear solution, recommendation, or best practice in order to honor the general cookbook format. I have not always succeeded, and many such topics are hotly debated. Beyond that, I must leave such items to the many competent books in the category of general SOA. If you want a really good book on such SOA matters, I recommend *SOA in Practice* by Nicolai M. Josuttis (O'Reilly) (<http://www.oreilly.com/catalog/9780596529550>).

Glassfish and WebLogic

While there are a number of excellent application servers that provide web service functionality, Glassfish is the first project to support Java SE 6 and fully implement the new Java web services standards in an interoperable way. There is no leftover disposition toward RPC, as remains in some vendors. While many examples have been tested and are shown in other containers, such as Oracle's WebLogic 10gR3 or Axis 2, Glassfish v2 is the default for these examples.

I really tried to focus on portability, and you should be able to make these examples work in your engine without much trouble.

BPEL orchestrations

The WS-BPEL standard is an important development in recent years as companies look for ways to organize, streamline, and represent their business processes in ways that are available to developers, systems analysts, and architects. BPEL allows you to create composite services by sewing multiple web services together in a flow, presenting a single view of the orchestration to the world. Because you can combine multiple web services in a very loosely coupled manner, orchestrations also help promote reuse of your services.

BPEL is supported by a few open source implementations, such as Apache ODE, as well as a variety of commercial products; each flavor is examined here.

Enterprise Service Bus (ESB)

The ESB is not strictly necessary for SOA, but it frequently is an important part of mature SOAs. An ESB acts as a mediator, router, and layer of indirection between service integration points. Early EAI (Enterprise Application Integration) efforts soon found themselves laden with many point-to-point integrations that required creating a specific interface for each connected node's route to any other node. The ESB reduces such complexity. Instead of connecting every service to every other service, you can just connect them to the ESB. In this way, the ESB serves the same function as any computer bus, and is responsible for mediation and routing messages to appropriate services.

Unlike most other topics in the book, there is no standard for ESB; as a consequence, vendors implement them very differently. The terminology is not always the same (what one vendor calls a pipeline, another might refer to as a channel, though the concepts are similar). The features in some cases overlap with other elements of an SOA suite of tools. For example, some ESBs do light repository work, or maintain an internal rules engine.

In 2005, Sun began promoting the Java Business Integration (JBI) specification, and ultimately created a reference implementation in OpenESB. Many vendors, however, view this as a Java-centric, bloated spec. Moreover, a Sun specification doesn't take advantage of whatever vendor tools might already exist. As a result, JBI doesn't rule the SOA marketplace in any way.

This book therefore offers an overview of the ESB landscape to help you understand the features offered by a variety of popular vendors, to outline the role that ESBs play within SOA, and to help you make an informed decision about what to look for in a vendor.

Intended Audience

This book is intended for experienced Java developers and architects. The world of web services is enormous, complex, and rapidly changing. I assume that you have a background writing, deploying, and maintaining enterprise applications in Java. You need to be familiar with Java SE 5 or 6; servlets and JSP; Enterprise Edition containers such as Glassfish, WebLogic, or Tomcat; and Internet protocols such as HTTP. You should be comfortable using Enterprise JavaBeans, and be familiar with all of the standard enterprise stuff, including JDBC, JNDI, EARs and WARs, and the basic principles and concerns of modern enterprise development. If you have used web services with earlier versions of Java, this book will be handy in learning the updated APIs. They have radically changed, and this cookbook should help you get underway quickly.

Daily Java enterprise developers will get the most out of this book. But SOA is a special kind of architecture and an organizational strategy, not a development style or methodology, so some of the recipes address architectural matters such as governance and patterns, with the expectation that it is useful for developers to understand the larger context in which their work will operate. On the other side of the coin, architects will find that the APIs and capabilities may augment their work. Finally, there are more hands-on architects today than ever before, those who are responsible for both designing and writing applications that must advance the architectural agenda of the overall enterprise. This book attempts to address these audiences. I only touch on business matters, such as SOA ROI. While such topics may seem out of place in a development book, I believe it is important for you as a developer to take a comprehensive and global view of SOA to keep your activities in line with what SOA requires not just in code, but in perspective.

It is assumed that if you are doing web services development that you have a basic understanding of XML, XML schema, and the Java APIs that deal with them. In some cases, you might have been working with a technology for years, but haven't used the latest APIs. So the chapter on XML addresses how to use the StAX APIs for working with XML streams, but it doesn't cover SAX and DOM. It also includes a variety of tools that can aid in SOA development. DTD is left out entirely in preference of Schema.

To maintain a solutions-based focus, I left out a lot of basic web services overview material and assumed that you are an SOA implementer-to-be and therefore have at least heard of SOAP and WSDL. Introductory sections within different chapters offer a good overview of each topic and situate them within the ongoing debates regarding SOA implementations. Then we'll quickly get down to business.

I have tried to balance the book so that you have everything you need, and nothing you don't. It's impossible to please everyone, of course, but I hope the balance works for you.

Because SOA outlines a way for getting different platforms to interoperate, there are recipes dealing with languages other than Java, such as Ruby, Python, and .NET. It is not required that you have a background in these languages, as the recipes are fairly simple and limited, shown only to express how you can get different platforms to communicate.

What This Book Covers

While Java is known for its portability, and many web services artifacts are portable, vendors frequently offer considerable extras in an attempt to make things more convenient for you (or lock you in, depending on your level of cynicism). Given the fact that we're covering such a wide array of material already, it is simply not possible to cover the minor changes in how to do everything on specific platforms. Given products such as WebLogic, Tomcat, WebSphere, Glassfish, ServiceMix, CXF, Mule, Active Endpoints, OpenESB, and dozens of other vendor offerings that address different aspects of SOA, there is problem of multiplication in a book such as this one. So I've chosen a middle-of-the-road stance for the most part, and kept mostly focused on what tools will be available to the most people. Here's the lineup:

Part I, SOA Fundamentals

Chapter 1, *Introduction to SOA*

This chapter defines relevant terms and introduces architectural topics that will guide the growth of your SOA. The items covered in this chapter are not exhaustive, nor are they intended to be followed in a specific order; it is an orientation to SOA concepts.

Chapter 2, *XML Schema and the SOA Data Model*

Because SOA and web services rely so heavily on XML, this chapter covers how to use XML, XPath, and Schema in ways that specifically support your service-oriented effort. Some useful tools are highlighted, as well as how to use JAXB to convert between Java and XML. We'll cover key SOA topics such as canonical data models and schema-based validation.

Chapter 3, *Working with XML and Java*

Whereas Chapter 2 focuses largely on XML schema, design, and validation topics, this chapter extends the discussion of Java and XML into document instance processing. It addresses the new StAX API, JAXB, and XML catalogs.

Part II, Web Services

Chapter 4, *Getting Started*

As Java programmers, we're used to a variety of convenience methods and classes; this chapter is a bit of a "convenience chapter." That is, you'll just set up shop with some different containers so that you are ready to execute clients and services in a few different popular environments. You'll do a "Hello World" service, and I'll introduce WSDL, discuss service clients, and show you tools for monitoring and debugging your services during development. Then we can move on to more specific API work in the coming chapters.

Chapter 5, *Web Services with SAAJ*

The SOAP with Attachments API for Java (SAAJ) is introduced in this chapter, which deals largely with how to build SOAP messages programmatically. SAAJ is a low-level API that works directly with the XML in your services and clients.

Chapter 6, *Creating Web Service Applications with JAX-WS*

The Java API for XML Web Services (JAX-WS) builds on the older JAX-RPC API to make creating and invoking web services much easier than it is with SAAJ. This chapter covers how to do many of the same kinds of things you do with SAAJ, but in a slimmer, more abstract manner. But beware: if you're used to the JAX-RPC API, things have changed considerably.

Chapter 7, *Providing SOAP-Based Web Services*

Now that we've seen how to consume SOAP-based web services, this chapter covers how to provide them with a variety of advanced features, including binary content, headers, and more.

Chapter 8, *RESTful Web Services*

REpresentational State Transfer, popularly known as REST, does not use SOAP. As a consequence, you don't use the SAAJ or JAX-WS APIs to work with a REST-based SOA. This chapter covers REST, discusses why it's popular, and explores how to create meaningful web services that take advantage of the native protocols of the Web, without some of the overhead that SOAP-based services can create.

Part III, Business Processes

Chapter 9, *Service Orchestrations with BPEL*

This chapter introduces using the WS-BPEL (Business Process Execution Language) 2.0 standard to create business processes or workflows. BPEL allows you to orchestrate invocations of multiple web services and perform a variety of powerful activities on the inputs and outputs, including XSL transformations and other common elements of structured programming.

Chapter 10, *Advanced Orchestrations with BPEL*

BPEL is a large and complicated beast, and this chapter dives into some of the more advanced aspects of using orchestrations, including dealing with faults, parallel activities, delays, and correlations.

Chapter 11, *SOA Governance*

Your SOA needs some measurement, some standards, and some tools in place to watch over it. Without these, you could spend considerable time and effort inadvertently creating a monstrous mash of rogue and duplicate services and unknown clients. Add to the mix the potential for poor documentation, low visibility, and inconsistency, and you've got a very expensive art project on your hands. SOA governance, an extension of IT governance, can provide some structure, visibility, consistency, and even policy enforcement to your SOA.

Part IV, Interoperability and Quality of Service

Chapter 12, *Web Service Interoperability*

Integration has been a thorn in the side of architects and developers for decades, and web services valiantly aim to overcome many of the issues surrounding it. And while very simple web services may readily interoperate out of the box, in the real world things can be trickier. This chapter introduces standard ways to enhance the ability of your web services to interoperate with services and clients on other platforms. The WS-Addressing specification is examined, along with implementing clients in a variety of languages, such as Ruby and .NET. You'll also find tips and tricks for working around gray areas in the specifications.

Chapter 13, *Quality of Service*

This chapter examines concrete, standard ways that SOAP-based services can improve their reliability, with specifications such as WS-ReliableMessaging.

Chapter 14, *Enterprise Service Bus*

Here we step away from the recipe format to provide an overview of the enterprise service bus (ESB), which frequently serves as the backbone of an agile SOA. Because ESBs are typically implemented as a collection of patterns and there is no standard specification for what an ESB is, I offer an overview of some of the leading ESBs. Despite the fact that they generally serve as message mediators and provide routing, security, and rules, they all work very differently. There is no single, clear market leader in this area, but we'll take a look at some of the most popular, including the

Oracle Service Bus, TIBCO's ActiveMatrix ESB, OpenESB, and Apache ServiceMix.

Unfortunately, there wasn't time to cover more topics relevant to SOA, such as security with WS-Security. Although related topics such as authentication are addressed in a few recipes, web services security is not given a full treatment here.

How to Read This Book

This is a cookbook, which means that (for the most part) topics are presented in a consistent problem/solution format, each succinctly stated. As necessary, solutions are elaborated upon within a discussion section. You could read the book cover to cover, though you probably won't use it that way. Generally, the chapters build on the knowledge gained in the previous chapters, but it is also structured as a useful reference that you can turn to during the different stages of building out your SOA, or use as a jumping-off point for your work.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Java SOA Cookbook*, by Eben Hewitt. Copyright 2009 Eben Hewitt, 978-0-596-52072-4."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

The examples are intended to be as self-contained as much as is reasonable. I find it frustrating when I read a computer book that contains small snippets of toy code with lots of //... throughout where stuff you need to know happens, or admonishments to refer to the downloadable code elsewhere that leaves out any pertinent code from the discussion.

There is arguably nothing that takes as much raw code as web services. Even a simple WSDL can span multiple pages. Add to that any schemas it references and you've got 10 or 20 pages of raw code on your hands before you even start talking, most of which

doesn't apply to the topic at hand. That is hard to read in a book. I have tried to balance this appropriately so that you have what you need when you need it.

It's also tough when every example makes such liberal use of the author's environment scripts and utilities that a newcomer can't distinguish the API ostensibly being illustrated from the author's own convenience classes. That makes it difficult to follow and understand. Such examples might make life easier for the author, but not for the reader. So I try not to do that.

Another way of approaching code examples is the single book-length "case study" that all examples refer back to. Some readers like these, but I find that the story of the case study begins inevitably to take over after a certain point, and it contorts the examples trying to fit every item into an artificial environment. Some systems just won't need to pass basic authentication data in headers; why strain your example or add reams of useless storyline to your fictional case study? This book takes a simpler approach.

For examples to be effective, I think that they must generally be complete, self-contained, and as easy as possible to replicate in the reader's own environment. Finally, they must be free of "toy code" and reflect real-world usage. These goals are admittedly hard to achieve when covering a topic with so many moving parts. Building web services is all about layers of indirection, which means even a simple example can involve a lot of different files and setup. As you can see, I've thought a lot about how to achieve all of these goals for the solutions outlined here, and I hope I've found a good working balance.

You can find the code examples at <http://examples.oreilly.com/9780596520724>.

Hang in There!

Finally, don't be discouraged. This stuff is hard. Really hard. The existing specs are sprawling, and loads of new specs are being cranked out faster than the industry can keep up. Vendors are behind, and developers are further behind them. It's an incredibly complex undertaking. There is also a lot of hype out there around SOA, which doesn't make it any easier to sort out. In such a setting, it can be tough to figure out what's real, what's fried air, and what's real but still two years away.

Happily, recent advances in the Java APIs have made doing web services much easier than it has been in the past. But as the world gets used to what's out there and gains an appetite for more sophisticated software, web services alone won't be enough. They have to orbit in a solar system with BPEL orchestrations, brokered ESBs, Business Activity Monitoring tools, SCA, and all manners of new layers. But SOA is real, and this stuff works. While the ideas on which SOA is based have their roots in technologies that are decades old, SOA truly represents a shift in how enterprises can do business.

Thank you very much for picking up this book. I hope it gives you some of the answers you need to make your stuff work. It is impossible to cover everything that an SOA practitioner needs in a single book, but I've done my best to hit the big targets and give you a solid foundation on which to build. I really hope you enjoy it.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width

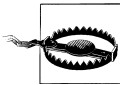
Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width bold

Used for emphasis in code listings.



This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Safari® Books Online



When you see a Safari® Books Online icon on the cover of your favorite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at <http://mysafaribooksonline.com>.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

<http://www.oreilly.com/catalog/9780596520724>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

<http://www.oreilly.com>

Dedication

This book is dedicated to all of my friends and colleagues at DTC, especially the Architecture and Programming groups. Barney Marispini, Bob LaChapelle, Bob Lemm, Brian Lee, Brian Mericle, Chris Servis, Deryl Heitman, John O'Brien, Kevin Williams, Mike Moore, Phillip Rower, Scott Ramsey, Steve Miller, Tom Schaeffer—this is for you. Thank you for being such a terrific team to work with, and for always pushing each other to be the best. It is a luxury to work with such a fantastic, dedicated set of people.

Acknowledgments

Special thanks go to Steve Miller, Deryl Heitman, John O'Brien, and Rich Kuipers for being so unbelievably supportive of this project. This book would not exist without your unmitigated generosity, understanding, patience, and raw vision. Thank you for believing so strongly in this project. I'm so grateful to work for the best company in the world.

Thank you to my tech reviewers, Jason Brittain and Nicolai Josuttis. You caught me in occasional flights of fancy and helped me to clarify important aspects of the topics.

Big thanks go to Barney Marispini, who painstakingly read complete early drafts, offered numerous helpful corrections and input, and really improved the quality of the book.

Thank you to Simon St.Laurent, with whom it has been a consummate pleasure to work, for his editorial guidance on this project. A huge thanks is due to my production editor, Loranah Dimant, my indexer, Lucie Haskins, and to all of the staff at O'Reilly, for their professionalism, dedication, and attention to detail. It's been a real pleasure to work with you all.

Thank you to Alison Brown, for being perfect. This, as with so many things in my happy life, could not have happened without you.

SOA Governance

11.0 Introduction

While this book is predominantly technical, it is also essential to address some non-technical items in order to highlight SOA as a unifying concept. Without considering the non-technical work within the framework of SOA, we are doomed to wind up over-engineering a bunch of software components that could have been written less expensively, more efficiently, and in a more easily maintainable manner. To focus solely on building web services is to miss the point of SOA entirely.

We have been creating web services in the industry for several years, and while the term “SOA” may already be 10 years old, many organizations are still only at the evaluation stage. Gartner research shows that only a small percentage of organizations that self-identify as those doing SOA are actually in a mature stage, meaning that they utilize a wide catalog of reused services in a repository; activity monitoring and automated alerts; brokered ESBs executing under governance domains; rules-based services; solid change management; federated partnering and security; advanced process automation that includes dynamic discovery, binding, and composition; and a complete governance board. Governance can act as perhaps the primary caretaker that helps nurture a fledgling SOA into a mature SOA.

Some of the discussions in this chapter, such as those on return on investment, may seem out of place in a technical cookbook. But SOA brings to the forefront the alignment of software with business strategy.

SOA attempts to undermine the traditional dichotomy between business and IT, an exaggerated view of which casts admins and engineers as feared and loathsome Quasimodo-like creatures, madly toiling nerdy beasts who work on their exotic art projects for vending-machine treats. They secretly ridicule their executive and marketing overlords for having all the flash, polish, and witless bravado of a hopped-up surfer dude with nothing on his mind but the next chance to hang ten on some sick Aussie waves. SOA recognizes that this dichotomy is part of the problem in modern software development, and seeks to define strategic ways of overcoming it. The architectural, design, and management principles that SOA makes available help support the

alignment of business and IT, recognizing that a little more harmony is key to realizing the greatest benefit.

Many of the topics in this chapter will address practical matters that developers and architects must face, such as service versioning and retirement processes. But to really make this an SOA book, and not a “Let’s Tape a Bunch of Web Services Together and Call It a SOA” book, we need to examine some of the more business-oriented topics as well. We won’t get into too much detail, but at this stage a dose of awareness is probably all we need.

11.1 Assigning Roles

Problem

There are a variety of roles specific to SOA development, maintenance, and governance, and these do not necessarily map to your existing human resources. You need to map current developers, analysts, executives, and infrastructure team members to support your SOA work, and you’re not sure who you’ll need.

Solution

As appropriate given the skills, backgrounds, and flexibility of the people you have or can hire, consider filling the following roles:

- Enterprise architect
- Solutions architect
- Technical architect
- Service developer
- Service custodian

These are discussed in detail in the next section.

Discussion

You may already have people in place for some of these roles, such as enterprise architect. In that case, making a slight adjustment to keep the special needs of SOA on their radar might be all the updating for that role that’s required. But let’s take a moment to discuss each of these roles in turn, to see what value they might add to your organization:

Enterprise architect

The custodian and ultimate decision maker of the overall architecture, enterprise architects take a 50,000-foot view of general activities. This is a business role, and its primary function is interacting with other business executives. It is his responsibility to ensure that the broad SOA efforts are in step with business goals, and

that the SOA is helping increase business agility. He will translate business goals into broad SOA objectives, set the overall direction of the SOA, and ensure that the work progresses within the established architecture for the organization.

There is often only one enterprise architect, and he is usually most visible at the incipient stages of a project.



Large corporations may have many enterprise architects, following Lines of Business. This is the case, for example, at a company like Dell.

Solutions architect

Solutions architects generally report to the enterprise architect or CTO, and may have more or less of a hands-on role, depending on the size of the organization. Solutions architects works with software architects, network specialists, and developers to come up with a technical plan for implementing business goals. To do so, they typically also work with Lines of Business managers or domain owners.

Solutions architects are really translators between business objectives and designs that can fulfill those objectives, and they consult with both the management and technical sides of the house to architect a solution. They may assist in vendor selection and focus on business integration, making sure that their solutions are aligned with all impacted business domains.

Depending on the size of the organization, there may be one or more solutions architects.

Technical architect

Technical architects fall right in the middle of the SOA, as they work closely with the solutions architect, business analysts, service infrastructure administrators, and service developers to identify viable technical specifications that conform to the needs of all of these different areas. This is a technical role, deepening the contours suggested by the solutions architect's work product. They must be sharply aware of SOA technology options, as well as fundamentals like network technology, infrastructure support technology, and so on.

Technical architects should have deep knowledge of the products in use. Just as the solutions architect is the translator between the business and the architecture, technical architects are the translators between everyone at the technology level in order to determine an appropriate architecture that's ready to be implemented by the service developers. They can also serve in a consulting capacity, when project managers and others in standard IT roles need assistance determining budgets, timelines, etc. Technical architects typically come with a specialty, such as service implementation, network infrastructure, or business processes.

Achieving the real promise of SOA ultimately means analyzing current and future business processes in order to streamline them and discover service candidates.

Architects who focus on business processes will use business process modeling tools to create standard representations of business processes. These models typically can be exported to a form such as BPEL. The output of this work can be used by service developers.

There are generally a number of technical architects.

Service developer

Service developers work with technical architects to implement services using an IDE. They work with the network administrators to ensure the safe arrival of the services into production. They may also support and maintain services after they are developed.

Service custodian

Service custodians' primary tools are the registry and repository. They ensure that services are available and documentation is correct. They may serve with architects on a change control or governance board, and they keep on top of versioning and availability.

11.2 Creating a SOA Roadmap

Problem

You want to build out your SOA in an organized, thoughtful manner, and be ready to address the wide variety of concerns that come into play as you embark on your SOA journey.

Solution

Create an SOA roadmap.

Discussion

SOA is a kind of architecture, or, rather, an approach to systems integration that represents a long-term strategy to realize an architecture. It takes a long time to build: as the SOA cliché goes, you can't boil the ocean. Between figuring out services, security, registries, and repositories; performing vendor selection; getting an ESB running; and advancing to technologies such as business process management, rules, and service virtualization, there is a lot to do in one lifetime. To get a picture of everything you need to do and the order in which you need to do it that makes the most sense for your business, you need a roadmap.

A roadmap in general terms defines checkpoints along the path of a journey. So the first thing you need to know is what you want out of SOA. Where are you going? Why are you employing SOA in your organization? What benefits, in concrete terms, do you hope to realize? What is your target state? What is the dream architecture that you hope

to have in place? Frequently cited reasons for implementing SOA in an organization include the following:

Business agility

IT needs to be in step with the business, so that it can respond more quickly to changing markets and business demands. Services can help you realize this by promoting reuse, a clean and manageable infrastructure, and interoperability with a wide variety of systems.

Legacy modernization

Companies that have been around for a long time may have lots of code that is 20 or 25 years old. In order to scale or take advantage of developer skills and new productivity and automation tools, you may feel the need to modernize certain applications. Ripping out and replacing lots of working technology, especially in business-critical systems, is often a very costly and time-consuming proposition. This is aggravated by the fact that businesses don't often see the value (and indeed, sometimes there may not be enough value) in simply rewriting working systems just for the sake of using the language of the moment. But wrapping legacy systems with services is a good first step that allows you to gradually create platform-independent facades for your key systems. In this way, you can eventually reach a point where it is less expensive and time-consuming to achieve the kind of flexibility you want within your enterprise.

Process improvement and automation

Organizations may want to improve their business processes using business management strategies such as Six Sigma or Lean. Six Sigma, for example, is a management strategy created by Motorola in 1986 that defines five basic steps: define, measure, analyze, improve, and control. Process management allows organizations to run at optimal efficiency. While these ideas have been around for more than two decades now, we have only recently gotten to the point where SOA can assist in management improvement efforts. The SOA tools give us a high degree of flexibility, transparency, tracking, and analytics.

Business process management (BPM) is an approach that enables organizations to map out their business processes, including the interactions of systems and humans, to determine how their processes actually work and to allow selective process automation with services. This is often done with business process modeling (also BPM) tools. BPM in conjunction with BAM (Business Activity Monitoring) is the chief means of realizing process improvements within a SOA.

Business Activity Monitoring Tools

Business activity monitoring will be a crucial aspect of SOA going forward. BAM allows companies to define key performance indicators (KPIs) around various aspects of their services, at the business and operational levels. A BAM-enabled SOA allows you to siphon off data as it flows across your service invocations (in near-time or real-time) in order to help make business decisions. For example, imagine that your company offers a New Hire service that allows people to apply for jobs, and then takes the job applications through the process of review by the hiring manager and HR, interviewing, hiring, and on-boarding. You define metrics around various data points in the process, and allow the tools to generate executive dashboards, typically represented as portlets.

Presenting and understanding the data culled from BAM tools is a terrific way to make process efficiency improvements. For the New Hire process, such metrics might include a ratio of interviews to actual hires, or the average time it takes to provision an employee workstation once they are hired, or the number of times a given hiring manager took too long to review an application, causing that point in the process to be escalated up the organizational chain.

To get this kind of functionality, you'll need to get BAM tools and a Business Process Management Suite from a vendor such as Oracle, TIBCO, IBM, or SoftwareAG, sometimes in conjunction with rules engines and analytics engines such as Cognos.

This is considered an advanced stage for a SOA, and some organizations that are heavily invested in such strategies and supporting tools have even termed their architectures POAs, or process-oriented architectures.

The basic idea here is that there are a variety of reasons that organizations might want to adopt SOA. And SOA doesn't happen overnight. So understanding why you want to invest in this long-term architectural strategy in general terms can help you refine your general goals into smaller, more workable, and more concrete goals.

Building on the principles of agile development methodologies such as Scrum, we can create a roadmap that allows us to do a little work and deliver value to the business incrementally. We generally don't use a waterfall method to develop our software, and likewise we shouldn't create a roadmap that front-loads lots of research and development with no proposed pay-off for a long time to come. Such a "big bang" approach is very dangerous in SOA.



As support for the idea that SOA represents a real change not just in technology but in organizational approach, it is useful to recall Conway's Law: "organizations which design systems...are constrained to produce designs which are copies of the communication structures of these organizations." You can't do SOA without examining how your project teams and organizational attitudes need to change.

There are a few general steps that every SOA roadmap is likely to define. These steps can (and should) be performed in an iterative fashion, as you reach each new milestone:

1. Starting a SOA, especially if you have not done much web services development before, can be very difficult. The market is overwhelmed with hype and noise. Figuring out what you need for your particular enterprise can be tricky, especially when you don't know what you don't know about SOA. You therefore need to start with a research and planning stage that will probably involve a lot of reading. Attend industry conferences and talks. Read whitepapers. Engage vendors in conversations with a clear statement that you're in a discovery stage and don't plan on buying anything just yet. If you have relationships with analysts such as Gartner or Burton Group or Forrester, engage them. Figure out first what you don't know. Remember that SOA is not just about getting an implementation of the WS-* specifications. The business must be involved. And while you can start top-down, and indeed you will need support up the organizational chain, your SOA will ultimately be realized bottom-up.
2. Now that you know what you don't know, use this knowledge to create a set of flexible checkpoints. To borrow from Six Sigma, this is a kind of measuring stage. Determine what needs to be worked on. Establish a set of tools that you will need for the current iteration. Establish organizational boundaries within IT. SOA represents a significant change in how a business operates. It's not the case that, in the manner of Internet startup accounting departments circa 1997, the SOA team should decide that all of the old rules don't apply to them. But an enterprise that is used to operating with traditional project teams may find the cross-cutting concerns of SOA very difficult to work with. Expect some resistance from teams who may feel threatened by SOA. Expect resistance from a potentially large set of tools that you may have in place to manage projects. SOA requires some organizational change, and you need to be prepared to address this. While this step may gradually become easier with each iteration of the architecture, each iteration will bring new concerns, and it will likely remain an important a challenge.
3. Do the work. Get the tools, write the code, automate the processes, build out infrastructure, update build processes, flesh out the governance scheme, do whatever it is that needs to be done to fulfill your concrete objectives. This is the execution step. Recall that SOA is not merely a technological endeavor, so part of the execution is also to engage the business to ensure that you have sponsorship and support. They must understand to some degree what SOA is, why you're doing it, the organizational changes that are typically necessary, and the new kind of input and involvement you'll require from them. At each iteration, re-engage the business to ensure that you're still on the same page, and highlight the ways in which future business initiatives will be supported by new SOA increments.
4. The final step is to optimize and adapt. Some of the choices that you made earlier might need to be revisited. Determine how you can improve efficiency, interoperability, processes, or other items within your SOA.

Create a set of roadmaps, each with a different scope. One might be a five-year plan for SOA that contains general goals. Another could zoom in on a more confined scope, perhaps a year or a quarter, and define tasks very concretely. It should indicate how you plan to address each aspect of SOA:

- Governance
- Business agility and organizational shift
- Tools
- WS-* specs or other related supporting items to learn and use
- RESTful or WOA concepts
- Monitoring, KPIs, BAM
- Rules engines/rules as a service
- Security enforcement points/security as a service
- Policies
- Registry
- Repository
- Interoperability
- Service identification and creation
- Infrastructure such as hardware and networking software

You don't need to address each of these items fully in every iteration. For example, you might choose to make your first services for internal use only, behind the firewall, therefore taking certain security measures out of scope so that you can focus initially on the core service creation and management.

11.3 Keeping Track of Your Services

Problem

You have developed several web services and want to keep track of them. They need to be visible to potential consumers, and your architecture team needs a central place to define their SLAs (service-level agreements) and policies and keep documentation and other metadata. Generally, you need to manage the entire service life cycle.

Solution

Use an enterprise repository. The repository will support your governance efforts. A good repository is a network-enabled data store that will serve as the location of record for all of the design-time and runtime artifacts your operating SOA requires. This includes XML schemas, WSDLs, service policies, and so forth.

There are many tools for this sort of thing, each with a different user interface and slightly different feature set. Software AG and Fujitsu have jointly developed a popular repository product called CentraSite, and offer a Community Edition license with limited functionality for free. The client is Eclipse-based, which is likely familiar to you, so this might be a good way to get started. Download CentraSite Community Edition 3.1.7 from <http://www.centrasite.org>.

Vendors such as IBM, Oracle, TIBCO, and others have robust enterprise repository tools that integrate with Eclipse-based development tools. Repositories are much more than regular databases, and are frequently combined with UDDI registries to allow dynamic discovery of services at runtime. They also can support migration of services between environments, and at design time can offer governance support. Such support often includes things like the ability to limit who can add a service to the repository (to prevent the Bunny Services and Rogue Services anti-patterns), and who can use which services.

Discussion

The enterprise repository is a cornerstone of your governance program. It is one of the chief tools employed by your architecture team or Center of Excellence. Repositories can also be very expensive. And while you can definitely get started without one, doing so will require careful maintenance of your meta-model. Still, using a model cannot prevent developers from putting their own services into production, and after a time you could end up with six slightly different services that you will need to maintain, version, decommission, and migrate clients for.

Ideally, your repository product should be standards-based and offer both a visual client browser and an API that you can use to invoke it programmatically. Repositories are typically integrated with the developer IDE so that during application development, programmers can simply browse the list of available services just as they would see classes in a selected package.

Many of the better enterprise repositories also make available a view for business users, or at least business analysts who work at the process level. As your SOA matures, and your service catalog is built out, you may find that you want to connect your services using executable business processes. Such business processes will be sewn together using services available in your repository. So a business analyst using something like BPMN within the IDE or studio tool will be presented a different view from that of the developer based on LDAP credentials, and see documentation and high-level details of each service. The analyst can then drag and drop items from the repository catalog to build the executable process.

Once the process is complete, the analyst can typically make a change to it and immediately see the impact across schemas and other artifacts participating in the composite service. Frequently, the impact analysis is presented in a graphical interface that shows

the interconnected web of resources. This can be very handy in helping to ensure that your business runs smoothly.

11.4 Determining a Data Ownership Scheme for Services

Problem

SOA presents a particular difficulty for determining ownership of data. A service must be able to not only manage but to truly own the data that it represents, or chaos will ensue. But an enterprise will frequently define services that must share common data such as customers, products, and so forth. How can this be managed?

Solution

As you define your architecture, differentiate between services that *own* objects, and services that *reference* objects owned by other services. Depending on the context, you may need to use caching techniques or subscription patterns to handle updates to the data.

Every service must be self-sufficient and must be defined as being totally in control of its own data. It is clearly preferable to never use data across business domains, as doing so can destroy the loose coupling SOA attempts to achieve. In the real world, however, our business domains sometimes do not match our data definitions as purely as we might like, especially when we're dealing with large companies with lots of opportunities for legacy modernization. You must make sure that a service only *references* data from other domains so that only one single domain is ever in control of a given business object.

Discussion

In your SOA, you have a `CustomerDataService`. This service represents the data for a fundamental aspect of your enterprise, and it can participate in many business processes or composite services. It will be used in a variety of contexts, including your Invoice service. Similarly, an Employee data management service could be used within an HR process and a Payroll process. Because service reuse is a chief aim of SOA and the reason it can realize significant ROI (return on investment), it would appear that you are on the right track.

But the way you handle the referencing of data across these services presents a problem, one that can become quite serious if not addressed clearly and decisively. The problem lies in data ownership. Services must own the data that they manage. It is perfectly sensible that the Customer service should be the ultimate decision maker and system-of-record or “gold copy” for all customer data. But the Invoice service must refer to customer data as well. What if a customer record needs to be updated, such as in the case of an address or phone number change? If the Invoice service maintains a copy of

the customer data, it must duplicate the update efforts of the Customer service, effectively undermining the point of your SOA.

So you cannot have the customer data stored in 2 or 3 or 17 places, as the Customer service would lose its credibility as the owner of that data. It cannot manage what it does not have a view into. Moreover, inventing a clever way to have the Customer service reach into each of the local data stores of the referencing services to manage the customer data they have “borrowed” is brittle, unmaintainable, and clearly does not support the agility concerns that SOA aims to address.

The naive solution would appear to be this: simply have the Invoice service go out to the Customer service every time it needs any information regarding customers. The Invoice service stores only an identifier, and not a full copy, reducing the record management problem. In this case, you don’t have two competing services that both own the customer record; instead, you have the Invoice service only referencing the data that the Customer service owns. This is a common solution in a number of environments, but it does not cleanly and wholly address the problem at hand within a SOA. For example, what happens to the Invoice pointer if a customer is deleted? This creates instability and invalidates the Invoice service without anyone being aware of it.

Moreover, this introduces an altogether different kind of problem: accumulated network latency. Having a variety of composite services reference such core services as Customer Information can create undue or unanticipated load. This can have a ripple effect that slows down services across domains.

So the crux of the problem is this: the Invoice service (which does not own but only references the data managed by the Customer service) must be able to use customer data flexibly and must have up-to-date information, but not be allowed to sink beneath the weight of the considerable latency involved in constantly invoking the Customer service every time it needs a small slice of information.

Happily, there are some reasonable ways of dealing with this issue.

Timed caches and distributed caches

An obvious but perfectly reasonable solution to this problem is to have the referencing service obtain a copy of the data from the owned service and treat the records as it would any other cached data. So the data is not a first-class citizen within the referencing application.

One way to handle this is to use a distributed caching mechanism. In Java, you can try EHCache, which is available from <http://ehcache.sourceforge.net> under an Apache license. The current version as of this writing is 1.5.0.

As a distributed cache, EHCache maintains stores on disk and in memory and can replicate data to listeners, as well as invalidate stale data. It has a number of extensions that allow it to support advanced performance improvement facilities such as ZIP compression and caching of JSPs and servlets. EHCache allows you to

programmatically create a cache manager object that is responsible for conducting the CRUD business of the cache and for coordinating work among its constituent caches.

In its upcoming version, EHCACHE supports both SOAP and RESTful web services, as well as SOAP security specifications and the WS-I Basic Profile. Under the hood, the EHCACHE server edition uses Glassfish.

When you push updates to the cache, interested services (such as Invoice) can reference the distributed cache. There are a couple of trade-offs though. First, you will need to have a lot of storage capacity, and you could waste processing cycles and consume network resources pushing updates that may never be read.

You might also use a timer (such as the EJB `TimedObject` interface) to perform a cache update in the middle of the night. The update could be pushed from the service of record (in this example, the Customer service) out to a distributed cache that referring services use. Or, if each referring service maintains its own cache, it can pull the data into its local database or cache structure. The advantage to this approach is that it's simple. The disadvantages are many, and include the inability to get updates outside the pre-set schedule. It might work for some brick-and-mortar retailers, but it is not effective for global organizations operating 24/7.

It's important to keep in mind that the cached copy is nothing more than that, and that the real service of record must always remain the true manager of its data. Perhaps more importantly, you must not undermine the authority of the service of record by going in any back doors. This issue is a reality in modern organizations. A developer who knows the location of the "real" data (and has been accessing it that way for years) may be reluctant to go through the front door of your web service interface and incur the attendant overhead. This is dangerous, and you must take steps to lock down your data store or record set to ensure that your service of record is the only custodian of that data.



As a general architectural principle, you can also consider data partitioning within your caches to improve speed. Partition data across date ranges, alphabetic keys, by function, group, or whatever makes sense for your dataset.

Messaging

Some of the problems introduced by caches and their maintenance can be addressed with messaging. Actually, used together, this is a one-two punch combination that can really improve the robustness, dependability, and performance of your service-based systems.

Synchronous communications (such as the request/response-based system of the Web) are frequently required. They are simple from a user standpoint and from an architectural standpoint. But they can also tie up resources under heavy load, and forego a degree of reliability.

Standards-based messaging systems, such as JMS, can scale very, very well. Because they are asynchronous as opposed to synchronous, they do not send a request and wait for an immediate response. Email is a form of asynchronous communication, whereas a telephone call is synchronous. Because the two sides of the line are not tied up concurrently, in asynchronous communication the system is free to perform the work with the highest priority and provide a timely response. Generally, of course, this happens very fast.

Your service can post to a messaging system, or be directly invoked over JMS, and if a reply is required, there are a few different ways clients can receive their reply. You can employ the JMS Reply-To mechanism, and set up a second queue as the destination for the initial queue to deliver its reply messages.

Clients can then poll the queue to check for messages. You can also use the message selector feature of JMS to narrow down the list of potential messages a client will receive.

11.5 Handling Legacy Programs and Heterogeneity Within Your SOA

Problem

Your evolving SOA defines services in different ways: you have some services that use WSDL and SOAP over HTTP, you have some standard JMS messaging services that use mediators, you have a couple of services defined using RESTful services implemented in PHP and need to unify them all.

Solution

In fact, you don't need to unify them all. This is not a problem, and may be a sign that you're doing some things right. Of course, it would be convenient if there were only one way to do everything in the world, but there isn't. So just make sure that you know exactly what services you actually have (a registry/repository product helps with this) and know the consumers of your services. As your SOA grows, make sure that you have products in place to identify "rogue" services.

Discussion

There is not one single way to create a SOA, and an enterprise large enough to bother with creating an SOA will have services defined in different languages. Programs that have been running for 15 or 20 years on a mainframe should not be ripped out and replaced just because they are not written in the language du jour. They have stood the test of time, and more importantly, they do what they are supposed to in a reliable way. There may be considerable knowledge within your enterprise about how those

programs work and what they do (even if most of that is stored in an old-timer's head). In fact, this situation is exactly what SOA is intended to address.

There are many options (which are generally vendor-specific) for wrapping legacy programs to expose them over a network so that they are accessible within a SOA. If a legacy program or some generated “modernization” wrapper code does not present an interface that you want to carry over into the future, consider using the Facade pattern to present to the world the interface that you want it to see, at the level of granularity that you like. Under the hood, use adapter code to connect with the legacy program.

With respect to having a variety of platforms implementing your services, such as SOAP/WSDL, REST, or straight Java components, that is OK. You want to have your services visible to the enterprise, and using a variety of formats may make representing your service catalog in a uniform matter slightly more difficult. But that is certainly not a show-stopper, and most enterprise repository products will handle this just fine. Do not attempt to wrap everything in the single services style you've selected. Allow services to be defined in the format that best suits them, assuming that considerable work has already been put into creating, testing, and documenting these existing systems.

11.6 Documenting Services

Problem

You want to document your web services appropriately so that developers can find them, and understand what they provide and how they provide it.

Solution

Use your enterprise repository. If you don't have one, there is no single way to do this, but there are some questions that you should make sure to ask yourself as you document your services. We'll examine these now.

Discussion

Given the wide variety of implementations and platforms for services and governance within any given organization, there is not one single way to do this. You might be running REST services with PHP clients, with OpenLDAP as a service repository. Or you could have services implemented with JAX-WS using a vendor platform for your registry/repository such as Software AG's ContraSite, or IBM WebSphere Service Registry and Repository, or you may be using Apache Scout for UDDI. The capabilities provided by these different products and platforms obviates any discussion of a single way to document services. But there are some things to keep in mind as general rules, which you can then adapt for your organization.

Here is a list of things you should include in your documentation:

- The name of the service.
- The general type of service: a business process, a data service representing a business entity, a functional service, and so on.
- A succinct statement (one sentence) indicating the purpose of the service.
- A descriptive overview that acts as an executive summary. This makes browsing services easier for project managers and analysts.
- A specification that is a business analyst could understand. It should be detailed enough for the analyst to be able to decide if a service is appropriate for use within a given solution.
- A technical specification. This is written for an audience of developers who need to know how to use your service. Keep this discussion client-facing; they should be able to figure out what they need to do to consume this with only this section. It should indicate operation signatures and actions (if present).
- Any special items for consumers, such as whether using the service requires an account, how to authenticate, what headers might be required, if attachments are expected, and so forth.
- The major and minor versions of the service. If this service is deprecated in favor of a newer version, indicate that along with an end-of-life date (assuming that you know all of the consumers or are ready to do that). If there are other versions in use, you might link to them as well.
- Optionally, an example of how to consume the service in different languages, if that's appropriate in your organization.
- Any related services that may also be of interest, or that might be what the user is really looking for.
- Service-level agreements, including QoS response times, responsible parties, etc.
- Any standards or conventions that have been used. Indicate if you are using WS-Addressing, MTOM, etc. These might include in-house conventions if that's your audience.

Here are some special considerations to take into account if the service is being developed for internal consumption, within your organization:

- Indicate which compositions may use this service. Often, tools will do this for you, showing graphically what other items in the repository refer to this service and even its attendant documents such as schemas.
- Include data resulting from tests that you've executed against the service, so that architects and administrators can make informed decisions regarding usage levels.
- Describe network-related data for the service, such as throughput, response time, availability, scheduled maintenance windows, etc.
- Optionally, include dates the service entered production and dates for each version.

Sophisticated commercial repository tools will handle all of this for you automatically. This should give you a reasonable start as you consider how to document your services. If you do not have a vendor tool available for this purpose, consider adding this information as a part of your build process. If you've used Maven to create your project, this is very easy to add to the site template and send to an internally available web server. Alternatively, you could consider using a doclet, Docbook, or other markup to generate this structure for you. Docbook is a good idea if you want your documentation in a variety of formats, such as HTML, PDF, or Office documents. If you have little in the way of automated technical documentation tools, you might use an XML grammar and XSLT to generate a site.

11.7 Setting Up a Service Registry

Problem

You want a centralized location in which to store your web services so that they can be dynamically discovered.

Solution

Try using an implementation of the Java UDDI (Universal Description, Discovery, and Integration) specification, such as the free and open source Apache jUDDI project.

Discussion

A UDDI registry provides a standard way of storing information about web services and the organizations that provide them. Registries allow you to store, query, and update the information surrounding your SOAP-based web services. They support models for two basic items: information about organizations, including government bodies, corporations, and business units; and these organizations' requirements for accessing their services.

UDDI

jUDDI (pronounced "Judy") is an Apache project implementing UDDI v2 (many commercial products now support UDDI v3). The most recent release of Apache jUDDI is from December 2007, and it supports JDK 1.5 and Servlet 2.3. The registry is deployable as a WAR, and acts as a frontend to a relational database.

Originally the idea with UDDI was this: businesses would create services and add them to a registry, and then make the registry available to the public. The conglomeration of registries in the world would act as a sort of Yellow Pages, in which software agents could be set up to query service registries, discover services that matched their needs, and establish a business relationship on the fly. Runtimes would use these agents to select and invoke services at runtime, using a Java API called JAX-R.

Things never quite happened like this. To begin with, businesses don't (and probably shouldn't) trust software to establish business relationships on the fly for them. Businesses establish relationships on the golf course, and with careful consideration. Developers still had to integrate the services on which their software depended using the grizzly JAX-R API.

As a consequence, UDDI has waned significantly in popularity in recent years. That having been said, there is some potential value in setting up a private, internal registry for your company that you do not intend for public consumption. As your organization grows its SOA, you need to have a single location that catalogs your services. The process of service enablement in your organization is likely to graph like a hockey stick. That is, it can take a while for people to understand SOA, see the value, get the infrastructure set up, and deliver some services. But once it takes seed in an organization, SOA can blossom fairly quickly. You need a way to manage service proliferation so that developers do not create redundant services.



If you are looking at repository products, many of them will fold a registry into the repository, so you may not require separate installations. You might look into products that combine a registry with an ebXML repository, as ebXML offers a more robust set of features.

Getting jUDDI

The current releases of jUDDI are available for download from <http://ws.apache.org/juddi/releases.html>. As of this writing, the current release is 2.0rc5, available as a WAR for installation in a container such as Tomcat, and as a standalone JAR. jUDDI is a frontend to a relational database, accessed with JDBC. So aside from the jUDDI bundle, you need to connect to one of the supported databases as well.

Deploying jUDDI on Tomcat

Let's look at how to set up jUDDI on Tomcat. These instructions work with jUDDI WAR 2.0rc5 and Tomcat 6.0.16. Once you have downloaded jUDDI and have Tomcat running, here are the steps to set it up:

1. Rename the file "juddi", as this will make configuring the datasource and other items easier. There are a number of properties that expect your installation to be at `<host>/juddi`. Of course, you can change all of those if you prefer. They can be found under the root of the WAR in `juddi.properties`.
2. Copy `juddi.war` into the `tomcat-home/webapps` directory. It will deploy automatically.
3. Visit the jUDDI homepage in your new installation at `http://localhost:8080/juddi-web-2.0rc5`.

4. Click the “validate” link. You will see the “happy jUDDI” page indicating whether jUDDI was able to find all of the components necessary to run.

If your database is not set up or is set up incorrectly, you may see an error such as this one on the “happy jUDDI” page:

```
DB connection was not acquired.  
(Cannot create JDBC driver of class '' for connect URL 'null')  
- SELECT COUNT(*) FROM PUBLISHER failed (null)
```

In this case, you’ll have to set up the jUDDI database and a Tomcat datasource for it.

Setting up the jUDDI JDBC connection

For this example, you’ll use MySQL as the jUDDI database. To complete the installation, you’ll have to get the MySQL driver installed, get the datasource set up, and create the jUDDI database.

Setting up Log4J. First, some minor housekeeping. To log jUDDI correctly, create a file called *log4j.properties* under the *WEB-INF/classes* folder of your jUDDI WAR. You may have to create the classes directory:

```
/opt/tomcat-6.0.16/webapps/juddi/WEB-INF/classes/log4j.properties
```

Edit the file so it has the following contents (according to the location of your Tomcat installation):

```
log4j.appender.LOGFILE.File=/opt/tomcat-6.0.16/logs/juddi.log
```

The next step is to set up the database.

Setting up MySQL database. There are a number of steps involved in setting up the database. Navigate to your expanded jUDDI WAR and find *WEB-INF/juddi.properties*. Uncomment all of these lines:

```
juddi.jdbcDriver=com.mysql.jdbc.Driver  
juddi.jdbcUrl=jdbc:mysql://localhost/juddiDB  
juddi.jdbcUsername=juddi  
juddi.jdbcPassword=juddi
```

This will allow you to connect using JDBC directly.

Now you need to create the database that will store the registry entries. It will be called juddiDB, with a username of “juddi” and a password of “juddi”.

jUDDI comes with a bunch of scripts specific to different database platforms that will create the database it needs for you. Supported platforms include:

- db2
- derby
- hsqldb
- mysql
- oracle

- postgresql
- informix
- daffodildb

However, you should be able to use jUDDi with any ANSI-compliant server. The only trick here is that the scripts are not in the WAR you downloaded. You have to download *juddi-2.0rc5.jar* from the [ws.apache.org/juddi releases](http://ws.apache.org/juddi/releases) page to get the necessary scripts. Navigate to the directory to which you downloaded the JAR and extract the scripts:

```
$ jar -xvf juddi-2.0rc5.jar juddi-sql
```

This creates a directory called *juddi-sql* that contains the scripts you need to execute to create the database. Navigate into the *mysql* directory and open the README file, which contains instructions on how to proceed.

Now you'll set up the database on the command line with the MySQL client. First, make sure that MySQL is started. I'll use this command:

```
$ /etc/rc.d/init.d/mysqld restart
```

Then make sure that you have a user named “juddi” with a password of “juddi”.

Now copy the *create_database.sql* script in the *mysql* directory to a file named *create-juddi-db.sql*. This is to ensure that you retain a pristine copy in case you make a mistake when modifying the file. Add an insert statement to the bottom that will create a publisher—someone who can add services to the registry:

```
INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,
    EMAIL_ADDRESS,IS_ENABLED,IS_ADMIN)
VALUES ('eben','Eben Hewitt','eben@example.com',
    'true','true');
```

Now open the file using a text editor. You'll see a number of `${prefix}` variables. Remove those using a search-and-replace with an empty string. Then add the following code to the top of the file to create the database and allow *juddi* to access it:

```
DROP DATABASE IF EXISTS juddiDB;
CREATE DATABASE juddiDB;

GRANT ALL ON juddiDB.* TO juddi@%' IDENTIFIED BY 'juddi';
GRANT ALL ON juddiDB.* TO juddi@'localhost' IDENTIFIED BY 'juddi';

USE juddiDB;
```

Here you are going to use MySQL in batch mode to process this collection of SQL statements all at once. Using the < sign indicates this to the server:

```
$ mysql -h localhost
-u root -p < /home/ehewitt/mysql/create-juddi-db.sql
```

This will create the database, add the tables, and put a publisher in it. You can now run a simple query to make sure your publisher exists:

```

$ mysql -u juddi -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 50
Server version: 5.0.22

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> use juddiDB; select * from PUBLISHER;

```

If you see the data you inserted into the database before, you're in business.

Setting up a Tomcat datasource. Now that the database is set up, let's create a Tomcat datasource for it so that the web application can use it.

Get the MySQL JDBC driver (Connector/J) from <http://dev.mysql.com/downloads/connector>. I am using *mysql-connector-java-5.0.6-bin.jar*. Install the *.jar* file containing the JDBC driver in Tomcat's `<tomcat-home>/lib` folder.



There is already a `<resource-ref>` element set up in the *web.xml* file for jUDDI, so you don't need to add that.

You'll now create a datasource called `jdbc/juddiDB`. Navigate to the `<tomcat-home>/juddi/META-INF` directory in the exploded WAR. Create a file called *context.xml*. This will hold your datasource information for connecting with Tomcat. If you're using a different container, just set up the datasource as you normally would using the same properties here. In the file, type the following:

```

<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/juddiDB" auth="Container"
    type="javax.sql.DataSource"
    username="juddi" password="juddi"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/juddiDB?autoReconnect=true"
    maxActive="100" maxIdle="30" maxWait="10000"
    />
</Context>

```

If you want to use a different database, just install its driver and put the appropriate URL in the `url` field.

Note that you will have to restart Tomcat to pick up the changes in the context. To do so, navigate to your Tomcat *bin* directory and issue these commands:

```

$ ./shutdown.sh
$ ./startup.sh
$ tail -n1000 -f ../logs/catalina.out

```

The `tail` command allows you to see the end of the logs file, and the `-f(force)` switch keeps it scrolling as entries are added. (Of course, that's optional.) Now that the server

is running again, go to the home page at <http://localhost:8080/juddi/> and click the “Validate” link to make sure that everything went well.

If the Happy jUDDI page has no red in it, as shown in Figure 11-1, you’re all set to start publishing to your repository using the JAX-R API.

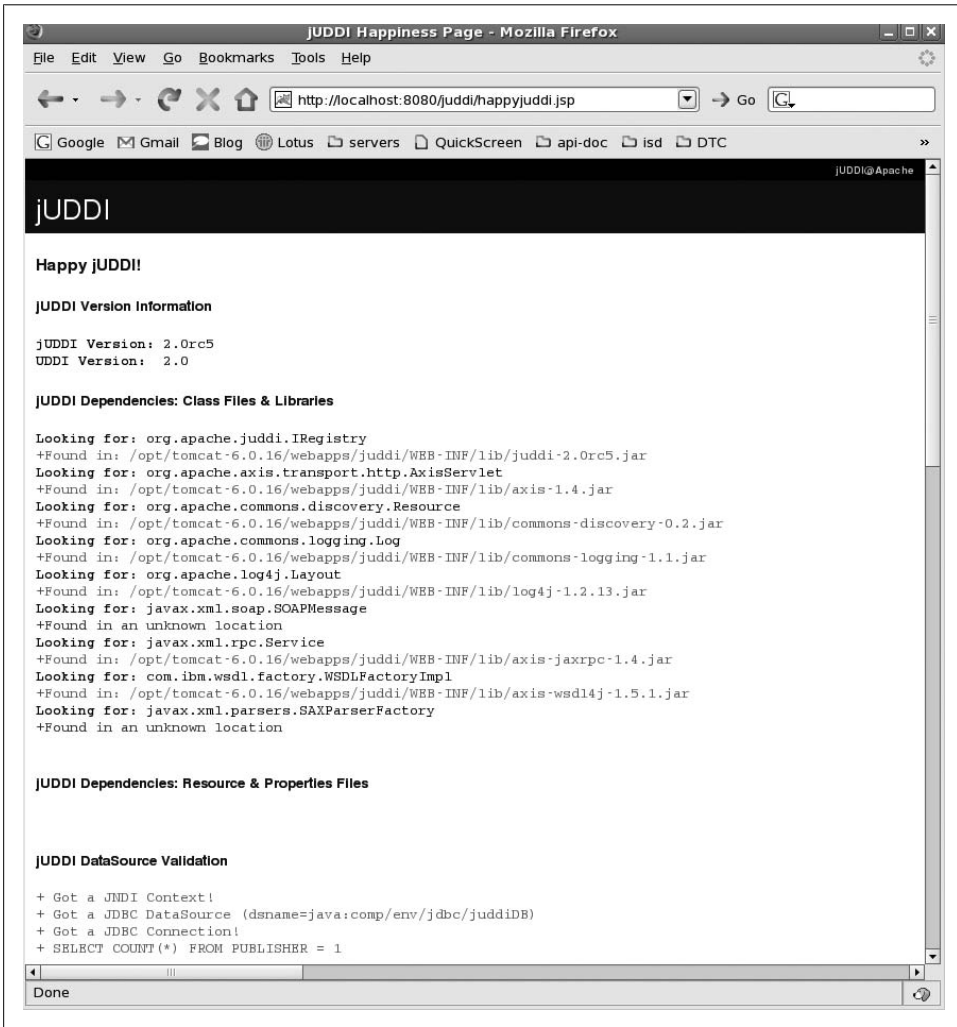


Figure 11-1. Happy jUDDI page showing complete installation

See Also

Recipe 11.11.

11.8 Packaging Related Services

Problem

You want to create a set of web services that are related conceptually and package them appropriately.

Solution

Require a separate source project and deployable artifact for each individual service implementation. Never allow more than one service implementation per WAR or EAR, even though it may initially seem more convenient.

Discussion

We're used to creating web applications that require us to put dozens of servlets in a single WAR. We're used to creating EAR and EJB-JAR archives that allow us to put multiple EJBs in a single deployable artifact. Do not allow this habit to permeate into your source projects when creating services. If you know that for a given solution you need to create a set of services, resist the urge to put more than one servlet or EJB that implements a service endpoint into a single deployable artifact.

For example, you might be charged with creating some data services for internal use in your organization. One will retrieve customer information, and another will retrieve store information. These services may appear related for a variety of reasons: because they are not intended for external use, because you're going to encapsulate them behind the same domain's ESB, because they are read-only, because they wrap legacy code on the same iSeries mid-range box, or because they both must use the same third-party adapter to communicate with a backend. However, to package these two services together is a category mistake. If two services are so designed that it appears incontrovertible that they be deployed together, they are poorly designed. A service should have a single purpose (high cohesion). If it is composable within a larger service, that's terrific. But that's yet another deployable artifact because it, too, is just a service.

The obvious drawback to lumping multiple services together is that you cannot truly version them separately, and you cannot fix bugs in one without recompiling and re-deploying the lot of them. Because, like regular servlets, they have separate URLs (and separate WSDLs), it may seem that you can package them together to save development time. This may seem especially important when you have to include some third-party API, such as a JAR that allows your related services. Do not allow the idea of saving a few development steps to seduce you into defeating the whole point of SOA.

Always allow only one service implementation per deployment. If your service is implemented as a servlet, put that single servlet in your WAR, along with any supporting classes. If you have another web service that seems related, create a separate project, and put the second implementation in it. If it, too, requires some of the same supporting

classes as your first service did, then you have a shared dependency that must be treated as a first-class citizen. JAR those classes up, put them in a versioned repository somewhere (such as Archiva if you're using Maven), and declare the dependency in both service implementation projects. The extra time you take here is absolutely negligible in comparison with the flexibility you gain down the road. There's no point in doing SOA to create business agility, just to tie everything together on a fundamental and practical level. This will strangle your future efforts.

11.9 Retiring a Service

Problem

You need to retire a service from use.

Solution

As with so many items in the world of SOA, some of these activities will be dependent on your environment. But here are the basic steps for retiring a service:

1. In the repository, discover any compositions or partners that are using this service. Update them to the new version, or, if you are removing the functionality entirely, make sure that you compensate within the partner services so that they don't break once this service is gone.
2. The service is still deployed, and rogue consumers in your organization can discover the WSDL and start invoking it. So undeploy the service entirely from its container so that it can't possibly do any more work.
3. If you are using a Business Activity Monitoring solution, remove the service entry in that tool to minimize skewed numbers and unnecessary alerts. You do this after undeploying the service so that you can catch any continued activity and take necessary precautions.
4. Remove the service from the registry/repository. This takes down the "advertisement" for the service so that no new consumers begin using it.

11.10 Browsing a UDDI Registry

Problem

You want a quick and easy way to view the contents of a UDDI registry without having to write loads of JAX-R code.

Solution

Try UDDI Browser, available from <http://uddibrowser.org>.

11.11 Querying a UDDI Registry Programmatically

Problem

You want a way to query your UDDI-compliant registry in Java code.

Solution

Add the `<glassfish-home>/lib/javaee.jar` to your classpath, and use the JAX-R API to publish and query your registry.

Discussion

If you have set up a UDDI registry, you may want a way to query it from within Java code. The API for doing this is called JAX-R, or Java API for XML Registries. You can check out the JavaDoc for JAX-R online at <http://java.sun.com/j2ee/1.4/docs/api/>.



The UDDI spec is enormous and very complicated, and would require a whole book to cover it. Here you're just going to get a feel for using the registry, and I'll point to some useful tools for you to explore on your own. You can read the spec at http://www.uddi.org/pubs/uddi_v3.htm.

Overview

There are two basic interactions with a JAX-R registry: publish and inquiry. Publish adds information about organizations and services, while inquiry runs query statements against it that allow you to find information.

There are a few steps to get set up. First, put the `javaee.jar` on your classpath. That gives you the JAX-R API. You also need an implementation of things like the Connection Factory.

Testing the client

Example 11-1 shows a Java client that accesses your jUDDI registry and queries it for a little information.

Example 11-1. Querying a jUDDI registry with JAX-R

```
package com.soacookbook.scout;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Properties;
import javax.xml.registry.*;
import javax.xml.registry.infomodel.*;
```

```

public class Query {

    private static String INQUERY_URI =
        "http://localhost:8080/juddi/inquiry";
    private static String PUBLISH_URI =
        "http://localhost:8080/juddi/publish";

    private static String OREILLY = "O'Reilly Media";

    public static void main(String[] args) throws Exception {
        getVersion();

        queryOrgs();
    }

    public static void queryOrgs() throws Exception {
        Connection conn = createConnection();
        RegistryService registry = conn.getRegistryService();
        BusinessQueryManager query =
            registry.getBusinessQueryManager();

        List<String> orgs = new ArrayList<String>();
        orgs.add("O'Reilly Media");

        BulkResponse response =
            query.findOrganizations(null,
                orgs, null, null, null, null);

        @SuppressWarnings("unchecked")
        Collection<Organization> data = response.getCollection();

        System.out.print("Response Size: " + data.size());
        for (Organization org : data) {
            System.out.print("Org: " + org.getName());
        }
    }

    public static void getVersion() throws Exception {
        Connection conn = createConnection();
        RegistryService registry = conn.getRegistryService();
        CapabilityProfile profile = registry.getCapabilityProfile();
        String version = profile.getVersion();

        System.out.print("Version: " + version);
    }

    public static Connection createConnection() {
        Connection connection = null;
        try {
            Properties props = new Properties();
            props.setProperty("javax.xml.registry.queryManagerURL",
                INQUERY_URI);
            props.setProperty("javax.xml.registry.lifeCycleManagerURL",
                PUBLISH_URI);
        }
    }
}

```

```

        System.setProperty("javax.xml.registry.ConnectionFactoryClass",
            "com.sun.xml.registry.uddi.ConnectionFactoryImpl");

        ConnectionFactory factory = ConnectionFactory.newInstance();
        factory.setProperties(props);
        connection = factory.createConnection();

    } catch (JAXRException ex) {
        ex.printStackTrace();
    }

    return connection;
}
}

```

You get the properties for connecting to the UDDI registry just as you might for connecting to a JMS or EJB service, and once your connection is created, you use the `BusinessQueryManager` class to execute JAX-R queries. You execute a `findOrganizations` query, which returns a `BulkResponse`, and then loop over it to print the results.

See Also

Recipe 11.7.

You can also try the free and open source Apache Scout project, available at <http://ws.apache.org/scout>. There are precious few examples of using Scout, and continued activity on the project is unclear. But if you're willing to slog through the JavaDoc for the API, it's available. Scout is packaged with some popular open source containers, including Apache Geronimo and JBoss 4.0.2 and better.

To use Scout, you'll also need XMLBeans. The latest release as of this writing is 2.4, which you can download from <http://xmlbeans.apache.org>.

11.12 Understanding SOA ROI

Problem

Determining ROI for a traditional software project is different from determining ROI within a SOA. You need to understand these differences.

Solution

The guidelines below may help you to find operational money from the general principles of SOA. Or, you may determine that there is *not* significant ROI in SOA, and that it is rather a cost of doing business.

Discussion

As SOAs wind their way into more and more enterprises, the question of ROI becomes unavoidable. Many industry analysts and stack vendors (who, surprise! have a bunch of SOA software they're happy to equip you with) imply that the ROI in SOA is significant, but also suggest that the value is deferred until the benefits of reuse can be assessed. This recipe explores the reasons for this.

At the end of this recipe, we'll examine the contrarian view that suggests that you can't determine the ROI associated directly with SOA because the question itself is inappropriate.

ROI in traditional software projects

The return on investment for a traditional software project is simpler to calculate than that for SOA projects. Traditional projects tend to focus on the following factors:

- Structure or design that will reduce maintenance and overhead costs. These factors include developer time as well as environmental factors such as how much electricity or cooling is required.
- Features that will win new customers and streamline business processes.
- Features that will reduce time to market.
- Cost reduction. This could occur through licensing arrangements, reduced hardware requirements, consolidation of equipment and function, etc.
- Compliance with regulations such as Payment Card Industry or Sarbanes-Oxley. Indemnification is a form of relief that can be monetized.
- Enabling features that do not directly generate cash themselves but aid in creating the business intelligence can drive more profitable decisions.

The traditional pillars of calculating ROI include the following:

Cost-benefit analysis

Determining the real value of a proposed solution by examining the monetary benefits against the costs of deriving those benefits. This requires measurement of a variety of factors throughout the business, not just the development process.

Net-present value

A method of capital budgeting that focuses on figuring the time-value of money against present-day terms. This is used to determine the benefit of long-term projects.

Opportunity cost

This represents the amount of money it costs you to obtain the new product as compared to the money you would have made, if you had instead invested in a competing product. A common example of opportunity cost is that just leaving your money in the bank allows it to collect interest. But if you choose to invest that

money in order to make more than you can in interest, that interest is still money lost if you withdraw it to invest.

This all gets very complex rather quickly, and goes far beyond the scope of this book. The important thing to realize is that it's difficult enough to determine an ROI for a standard software project, despite the fact the constraints and structure of doing so are fairly well understood at this point. In general, managers focus on a fairly limited data set, regarding the proposed project in terms of costs and future expected benefits derived from features, consolidation, time to market, and other factors just shown.

Frequently, cost-benefit analyses are calculated with careful, perhaps even fanciful precision. I was part of a software project where a small portion of the ROI (a couple hundred thousand dollars) was attributed to its new user interface, in which users would have to make only three clicks instead of five to perform a certain use case. This portion of the ROI was determined thusly:

- T = Total amount of time in seconds it currently takes to perform the use case with the existing “five-click” user interface, averaged over a small sampling with a stopwatch.
- S = Time in seconds it would take to complete the use case with the “three-click” user interface, or roughly $3/5 T$.
- U = Projected total number of users performing the use case.
- W = Average hourly wages of said users, divided by 360 to determine wages per second.
- N = Number of times each user was projected to perform that use case in the next five years.

That gave something like the following equation: $R = (T * .6) * W * N * U$.

Everything gets rather more slippery in an SOA environment. Let's examine why now.

ROI in SOA

While SOA as a concept has been around for a number of years, it remains relatively new as a widespread practice. Determining a long-term ROI therefore has little precedent. But remembering what the goals of SOA are in the first place can help you establish some clear guidelines for ROI.

Primary business drivers for SOA are agility and flexibility. But these concepts are too general to be monetized and must be unpacked in operational terms. Because the advantages inherent in the idea of software reuse are well understood through their roots in traditional software projects and components, reuse makes a good starting point.

There are different kinds of reuse that SOA affords, however. In “application modernization,” SOA architects and developers find meaningful ways to service-enable legacy code. There are millions upon millions of lines of COBOL code in operation in the business world. These applications may be viewed as dinosaurs, and developers and

business users alike might love to replace them with slick new applications. But these legacy applications are frequently the foundation of the enterprise, and represent decades of stability, testing, and proven ability in the field. Rather than ripping and replacing, SOA seeks to leverage these significant investments by wrapping them with modern web services that can increase the agility and responsiveness of the enterprise. At the same time, there is a diminishing return on any system as it becomes more costly to contrive ways to manipulate and adapt these older applications natively into use within a modern application. SOA recognizes the benefit of tried-and-true applications in the enterprise, and attempts to walk this line by extending established value once compositions enter the picture. Which is where the next aspect of reuse enters the picture: after a service catalog is established.

Reusing existing services from a variety of clients through orchestrations and automated business processes makes a compelling cost benefit. Once your enterprise is service-enabled, the ability to quickly deploy new products on new platforms that build on the existing infrastructure can be quite valuable. The “network effect” (wherein the value grows exponentially as the number of nodes on the network increases) really blossoms at this point.

But it also seems clear that SOA as an integrative technology is going to be more expensive up front. Developers need education, the organization needs infrastructure to support it, and it can take some time to get used to and really learn to leverage the depth and breadth of the tools. Creating a service is far more complex than writing a series of classes or a component. You must put more careful consideration into how the service will be used, and by whom. The runtime environment for a service can be more volatile, particularly as you grow the number of service compositions.

A primary benefit of SOA is agility through reuse, and therefore you see the greatest return the more times a service gets reused, often over a period of many years. There is a little bit of art and a lot of educated guessing in the work of calculating an ROI. We can't lay out an entire ROI plan here, but there are some general points to look for:

Improved quality

Because services are vetted through the governance process, and because of their more manageable level of granularity, services can be developed in a straightforward manner, monitored closely, and improved without changes to clients. Often services show greater quality than their application predecessors, which translates into real money.

Increased return on legacy applications

By service-enabling or modernizing legacy applications, you extend their functional life, which increases the value gained by the original legacy applications.

Reduced integration costs

Integrating with purchased systems such as Peoplesoft, SAP, or Oracle Financials can be expensive, and you typically need to purchase adapters for working directly with their proprietary interfaces. Products that expose a service interface can be

much less expensive to integrate with. Providing a service interface to business partners on unknown systems can make on-ramping easier, less labor-intensive and error-prone, and more cost-effective.

Reduced traditional application development costs

While this book focuses on developing services, one key driver of SOA is that it is easier to put together traditional applications by leveraging services. Within a standard development project, introducing an existing service (perhaps created by a third party) to handle specific functional aspects can reduce costs as well. Moreover, your developers can be more effectively used if they are orchestrating composite services from existing services rather than writing custom code for everything. If you have made an investment in SOA tools offered by the suite vendors (such as Software AG, IBM, TIBCO, etc.), you may find a drastic reduction in the amount of time it takes to create and deploy service-based applications. Their tools can require little or no coding to wrap legacy services and build sophisticated workflows. The time saved can make a good argument for real ROI in SOA.

In order to realize these benefits, you must make a sustained investment in SOA. The return happens over time—specifically, the amount of time it takes for your organization to have some significant number of services and a significant number of consumers. You must track, so that you can later show, the cost avoidance benefits due to your SOA efforts.

ROI implications of building out services within traditional projects

As you are building out your SOA, it is unlikely that you will be given a blank check and a well-appointed private room and told to write services until it's done. Most of your services will be developed alongside or as part of traditional application development projects. The way that these projects are budgeted, and the way that their ROI is determined, will not take into account the potential ROI locked inside the services that you will develop. That ROI will only be realized across a long time frame, as it obtains stature within a vibrant SOA.

So that presents a challenge. Specifically, you need to create a separate business case and a separate ROI prospect for the services themselves, taking their role within SOA into account. That is, the business case for creating the application that will use a service that you're about to develop will not be the same as the business case for the service itself.

Project managers and business managers will often want to conflate the service and the initial application that consumes it. This is tempting because it's easy to do, it's what we're used to, and we spent a long time learning our existing IT tools for project management. However, that doesn't mean it's the right thing to do.

You must protect the long-term life of the SOA by showing the benefits with which it rewards the business for its patience. You won't be able to do that if the ROI is baked together with applications. Baking them together is a kind of category mistake. The use

of the application project will generally be well known, clearly understood, and fairly succinct. An application that allows business users to change the prices of their products is probably not going to be used in any other way.

Just as you govern and monitor the life cycle of the services themselves, be prepared to govern their ROI life cycle as well. This will require a separate set of criteria than in traditional project-based software applications. For example, consider an item such as Time Value. Services can allow you to reduce batch processing cycles. Typical data warehouses perform ETL (extract, transform, and load) operations that can take significant time to process. This results in “data latency,” postponing the ability to make business decisions. Rather than saving data in staging areas and moving it around the enterprise, SOA may allow you to get to your data more quickly through data-based services.

There’s no ROI in SOA

The contrarian view suggests that there may *not* be significant ROI that can be readily and clearly associated with SOA. This view says:

- SOA is an architecture. It is a particular strategy for building software. It is not a business venture (unless you live in the Bay Area). It gives you certain benefits, and the reason that it has become popular in recent years is that it is an architecture that includes the business within its scope. It can generate real business value, but it’s not clear how an architectural approach translates readily and directly into a return on investment.
- SOA is a way to make the software you really care about. Let me make the point this way. If it’s fair to ask what the ROI on SOA is, then it is presupposed that SOA *is* something: that is, that it’s a product in its own right that the business is interested in. But I suspect that this would be greeted with considerable skepticism. The great Irish playwright Samuel Beckett once wrote, “Art isn’t *about* something, it *is* something.” With SOA, it’s the opposite: SOA *is about* something, it *isn’t* anything. SOA helps you make the products that your business needs. SOA is not what you want; it helps you get what you want. It’s just another cost of doing business. In the same way, businesses may not want to spend \$25 million on a building for their corporate headquarters. But you can’t have very productive executive meetings in a tent in the backyard.
- SOA is a software strategy that has certain aims, including better support of business initiatives. Within the context of a pilot project, in which you build a service and perhaps a project around it in order to show value quickly and support your fledgling SOA, it may be easy to see how avoiding services altogether and writing the application using a traditional, baked-in approach would be far less expensive. Detractors from the idea of SOA ROI suggest that there are other reasons for creating a SOA: that as an approach to software, the better able SOA is to support your business initiatives, the more return you can realize in the context of your

business initiatives. However, they would contend that there is little way to precisely map real dollars to SOA.

- SOA has certain benefits, not every conceivable benefit, including ready-bake ROI. Businesses do things all the time that don't show tangible returns. For example, many companies send 2, 3, 6, or 10 employees to the JavaOne conference each year, or have large, lavish corporate parties. Such items, frequently contextualized as “team building” and related matters, are difficult to map directly to a return. But these companies believe that such events and others like them help create a desirable workplace. While desirability cannot be quantified, it seems reasonable to believe that such benefits can allow companies to hire and retain better employees and to have a more productive workforce. Intangible benefits, such as “increased customer satisfaction” and “improved customer service” are no less real simply because they are difficult or impossible to quantify.

So it may be that SOA is a good idea for some businesses in some contexts, and just because you can't map an ROI precisely doesn't mean you shouldn't do it. It may also be the case that there is ROI in SOA, but that attempts to show it will only subject you to the fallacy of False Precision. One example of False Precision is to suggest that the normal human body temperature is 98.6 degrees. This is simply a traditionally used value, and there is no actual “normal” temperature that can be identified to a precision of a tenth of a degree. Instead, there is a basic range of perhaps 97–99 degrees in which different people normally operate.

The point of this recipe is to get you thinking about all of these factors in your organization. You may have to do some work to “sell” SOA within your organization, and only you can decide on the best way to determine the ROI for SOA. But consider if you're asking the right question, or even a fair question.

Finally, just because SOA is an architecture that includes the business within its scope in a variety of ways (including governance, BPM, and so forth), consider carefully if you are asking SOA to be responsible for more than it can reasonably be. Consider the well-publicized fringe benefits Google offers, including onsite dry cleaning and a gourmet chef. What's the ROI for gourmet food? Could you specify, with anything other than wild speculation, a return on investment in Kobe beef instead of ground chuck?

I have tried to give reasonable voice to both sides of the argument here. But sometimes, if solving a certain problem is really, really hard, it's because we're trying to solve the wrong problem.

One simple and perfectly reasonable way of looking at SOA ROI is to present SOA as a solution, and compare it with the cost of not getting a solution.