Do More with SOA Integration: Best of Packt

Chapter No. 11
"Loosely Coupling Services"
In this package, you will find:

A Biography of the authors of the book
A preview chapter from the book, Chapter NO.11 "Loosely Coupling Services"
A synopsis of the book’s content
Information on where to buy this book

About the Authors

Antony Reynolds has worked with Oracle BPEL from the days before Oracle acquired Collaxa in 2004 and with it the product now known as the Oracle BPEL Process Manager. Since then Antony has been a keen evangelist of the power of SOA and has worked with key European customers to help them realize the power of the Oracle SOA Suite within their own organizations. Prior to joining Oracle, Antony was a system architect working on the Galileo Computerized Reservation System. With more than 20 years of IT experience Antony has always focused on secure, high-volume systems on the leading edge of the technology curve and is now a leading proponent of business-focused SOA at Oracle.

Arun Poduval works as a Technical consultant at Midwave Corporation, specialized in SOA/Middleware.

For More Information:
Daniel Liebhart has over 20 years of experience in the information technology field, which has culminated in a broad technical and business know-how, which comprises the engineering, realization, and operation of complex and internationally operated IT systems for the Telecommunication, Finance, Logistic, and Chemical industries, as well as for public services. He has authored three books for Hanser Publications, is a passionate computer science engineer, has won several awards, and has worked for Trivadis, a leading independent IT service company operating in Germany, Austria, and Switzerland. He works as an assistant professor at the University of Applied Science in Zurich.

David Salter is an enterprise software architect who has been developing software professionally since 1991. His relationship with Java goes right back to the beginning, using Java 1.0 for writing desktop applications and applets for interactive web sites. David has been developing Enterprise Java Applications using both the J2EE standards and open source solutions for the last five years. David runs the Java community web site Develop In Java, a web site for all levels of Java developers.

Doug Todd is CTO of Enterra Solutions in Yardley, PA. He has more than 20 years of experience in systems architecture, applications architecture, systems integration, and applications integration with major corporations. Todd is responsible for Enterra's overall IT strategy and tactical implementation, enterprise information architecture, and technology product offerings.

Frank Jennings works in the Information Products Group of Sun Microsystems Inc. He has more than nine years of experience in Java, SOA and System Design. He is an Electronics Engineer from Madras University and has worked for several open source projects. Frank has written regular columns for leading Java journals including Java Developer's Journal and Linux Developer's Week. He holds a Post Graduate Diploma in Computer Science and an Advance Diploma in Computer Integrated Management from University of Indianapolis, and his blog can be read at http://blogs.sun.com/phantom.

Guido Schmutz is an Oracle ACE director for Fusion Middleware and SOA and works for the Swiss Oracle Platinum Partner Trivadis. He has more than 20 years of technology experience ranging from mainframes, integration, and SOA technologies in financial services, government, and vendor environments. Currently, he is focusing on SOA and application integration projects using the Oracle SOA Suite.

For More Information:
Harish Gaur has more than 13 years of experience in the enterprise software industry including more than seven years at Oracle. He is currently the Director of Product Management for Fusion Middleware at Oracle. In his current role, he works closely with strategic customers implementing SOA & BPM using Oracle Fusion Middleware. He is co-author of BPEL Cookbook (2007) and Fusion Middleware Patterns (Sept 2010). Harish holds an engineering degree in Computer Science and is an MBA from Haas School of Business, UC Berkeley.

Jason Williamson has over 17 years of experience in technology and business execution, from software development, product marketing, and management, to entrepreneurial enterprises. During his tenure with Oracle, he has been responsible for helping to develop and implement Oracle's strategy around legacy and business systems transformation. He now serves as a special advisor to Oracle's premier customers in the financial services industry. Jason spends his free time with his wife Susan and four children. He serves as a coach for youth sports and is involved in advocacy, fundraising, and speaking for Spina Bifida awareness.

Jeremy Bolie is a Senior IT Manager at QCT, managing the custom applications and Documentum development team. Jeremy has over 10 years of experience with Java and Oracle technologies, and has been involved with web services and Service-Oriented Architectures since the late 1990s.

Jerry Thomas is Chief Architect at CenterStone Software, which helps many of the world's largest organizations automate and manage their real estate, facilities, personnel, assets, leases, and workplace operations more efficiently. Thomas focuses on CenterStone's enterprise workplace management product and web services, BPEL, and system infrastructure. Prior to CenterStone, Thomas worked as a consultant and held principal development positions at Riverton, ONTOS, and Hewlett-Packard.

Kevin Geminiuc currently works as a senior software architect in Denver. Over the last 15 years, Kevin has worked as a systems architect, technical manager, developer, and hardware engineer. Kevin's technical interests include SOA, RFID, AVL, and genetic software.

Lawrence Pravin is the Product Manager, Process Integration Packs, Sierra Atlantic Inc. Process Integration Packs deliver end-to-end business process integration solutions between enterprise applications. He has over 10 years of rich experience in packaged applications, and has deep integration expertise with Oracle, PeopleSoft, Siebel, and SAP applications.

For More Information:
Marcel Krizevnik is a researcher at the University of Maribor where he is preparing a Ph.D. in computer and information science. Marcel started his career as a software developer of chemistry information systems. Now, his main research areas are service-oriented architecture and cloud computing. He is also a member of SOA Competency Center and Cloud Computing Center. In the last three years, he has been involved in several SOA technology projects.

Markus Zirn is a Senior Director of Product Management for Oracle Fusion Middleware. In this role, he heads the Strategic Customer Program, where he works with Oracle's leading and most innovative middleware customers. He has been part of the Enterprise Software industry for more than 10 years, including roles as Vice President of Product Marketing and part of the founding team of QUIQ and as a Management Consultant of Booz Allen & Hamilton's Silicon Valley High Tech Practice. Markus' passion for Service-Oriented Architecture (SOA) and BPEL stems both from practical experience designing and optimizing business processes as part of process reengineering projects and from being part of the advent of "software as a service" before web services became mainstream. He holds a Masters of Electrical Engineering from the University of Karlsruhe and is an alumnus of the Tripartite program, a joint European degree from the University of Karlsruhe, Germany, the University of Southampton, UK, and ESIEE, France.

Matjaz B. Juric holds a Ph.D. in computer and information science. He is a full-time professor at the university and head of the Cloud Computing and SOA Competence Centre. Matjaz is Java Champion and Oracle ACE Director. He has more than 15 years of work experience. He has authored/coauthored Business Process Driven SOA using BPMN and BPEL, Business Process Execution Language for Web Services (English and French editions), BPEL Cookbook: Best Practices for SOA-based integration and composite applications development (award for best SOA book in 2007 by SOA World Journal), SOA Approach to Integration, Professional J2EE EAI, Professional EJB, J2EE Design Patterns Applied, and .NET Serialization Handbook. He has published chapters in More Java Gems (Cambridge University Press) and in Technology Supporting Business Solutions (Nova Science Publishers). He has also published in journals and magazines, such as SOA World Journal, Web Services Journal, Java Developer's Journal, Java Report, Java World, eai Journal, theserverside.com, OTN, ACM journals, and presented at conferences such as OOPSLA, Java Development, XML Europe, OOW, SCI, and others. He is a reviewer, program committee member, and conference organizer. Matjaz has been involved in several large-scale projects. In cooperation with IBM Java Technology Centre, he worked on performance analysis and optimization of RMI-IIOP, integral part of the Java platform. Matjaz is also a member of the BPEL Advisory Board.
Matt Wright is a director at Rubicon Red, an independent consulting firm helping customers enable enterprise agility and operational excellence through the adoption of emerging technologies such as Service-Oriented Architecture (SOA), Business Process Management (BPM), and Cloud Computing. Matt holds a B.Sc. (Eng) in Computer Science from Imperial College, University of London.

Michael Cardella is a Staff Engineer at Qualcomm CDMA Technologies (QCT). Michael works in the custom applications development team, primarily on webservice- and business-process-related applications. Previously he served as Principal Architect for a leading web services security and management product.

Peter Welkenbach works as a consultant, senior architect, and trainer in the fields of requirement engineering, object-oriented methodologies, software engineering, and quality management. He has more than 20 years' experience of designing and implementing complex information systems for banks, automotive manufacturers, and pharmaceutical companies. Peter Welkenbach is a course developer, author of numerous publications, and speaker at JAX and international Oracle conferences.

Poornachandra Sarang, Ph.D., is CEO of ABCOM Information Systems and is currently a visiting professor for Post-Graduate Computer Science courses at the University of Mumbai. Dr. Sarang provides consulting services to worldwide clients in architecting and designing IT solutions based on Java, CORBA, and Microsoft platforms. He has authored/co-authored several books on Java, C++, J2EE, e-Commerce, and .NET.

Praveen Ramachandran works as a Technical Consultant for Midwave Corporation focusing on BPEL and other EAI technologies. Midwave is a rapidly growing firm that specializes in building highly available and highly secure information technology systems for medium to large companies and government agencies in seven midwestern states. Midwave is an Oracle Partner.

Ramesh Loganathan has 16 years of Systems engineering and R&D management experience in technology-intensive product development organizations. Ramesh has full life-cycle experience setting up and managing product development organizations and motivating high-caliber engineering teams. He has strong insight into Systems software, Middleware-technology, Database internals, Internet Architectures, and frameworks. Ramesh is now VP-Middleware Technologies at Pramati, driving the product direction and setting up a new Technology Consulting business around Middleware Systems.

Sean Carey is a Software Architect at SPS Commerce, a leader in hosted EDI. Sean has over seven years of experience in mission-critical e-commerce implementations, and 15 years of industry experience in software design.

For More Information:
Stany Blanvalet is a BPEL and J2EE consultant. Previously, working as a Java EE architect, Stany introduced and administered Belgacom's BPEL-based DSL provisioning application, a mission-critical BPEL production system. He is a contributor to the Jaisy-ORABPEL Interface project, an open-source JMX monitoring tool for Oracle BPEL Process Manager.

The Hoa Nguyen currently works for the SDC subsidiary of SpaceBel SA in Brussels as senior software engineer. His main interests are J2EE, web services, and workflow development with BPEL. Since 2001, he has been one of the lead engineers of the SSE project team at SpaceBel and is also in charge of SSE software releases and on-site SSE software installations at ESA.

Todd Biske is a Senior Enterprise Architect with Monsanto in St. Louis, Missouri. He has over 15 years of experience in Information Technology, both as a corporate practitioner and as a consultant, working with companies involved with Agriculture, Atmospheric Sciences, Financial Services, Insurance, and Travel and Leisure. He has an M.S. degree in Computer Science from the University of Illinois at Urbana-Champaign, is a member of the SOA Consortium, is a frequent conference presenter, and writes a popular blog on strategic IT topics at http://www.biske.com/blog/

When Todd isn't working or blogging, he spends the vast majority of his time enjoying life with his wife Andrea, and their three children, Elena, Spencer, and Maria.

Tom Laszewski has over 20 years of experience in databases, middleware, software development, management, and building strong technical partnerships. He is currently the Cloud Migration Director in Oracle's Server Technology organization. Most recently, Tom is spending a significant amount of his time enabling Cloud computing service providers on the Oracle software and hardware stack. This involves architecting future-proof cloud infrastructure solutions utilizing Oracle Exadata, Oracle Exalogic, Oracle Virtual Server, Sun Blade Servers, Oracle Database, Oracle Fusion Middleware, and Oracle Enterprise Linux. Tom holds a Master of Science in Computer Information Systems from Boston University.

Yves Coene currently works for SpaceBel SA in Brussels as Project Manager. He has 15 years of experience in aerospace software projects such as Ariane 5, the International Space Station, F16 MLU, and various other projects for the European Space Agency. Since 2001, he and his team have been responsible for the SSE project for ESA in Frascati, Italy.

For More Information:
Do More with SOA Integration: Best of Packt

A Packt Compendium is a book formed by drawing existing content from several related Packt titles. In other words, it is a mash-up of published Packt content – *Professional Expertise Distilled* in the true sense. Such a compendium of Packt's content allows you to learn from each of the chapters' unique styles and Packt does its best to compile the chapters without breaking the narrative flow for the reader.

Please note that the chapters in this compendium were originally written and intended as a part of various separate Packt titles, so you might find that the information included in this instance is more akin to that of a stand-alone chapter, rather than creating step-by-step, continuous flowing prose. We are sure that you will find this medley a useful and valuable resource with which you can benefit from a range of Packt books - and their authors' expertise!

*Do more with SOA Integration: Best of Packt* is a medley of eight separate titles from Packt's existing collection of excellent SOA books:

1. BPEL cookbook
2. SOA Approach to Integration
3. Service Oriented Architecture: An Integration Blueprint
4. Building SOA-Based Composite Applications Using NetBeans IDE 6
5. Oracle SOA Suite Developer's Guide
6. WS-BPEL 2.0 for SOA Composite Applications with Oracle SOA Suite 11g
7. Oracle Modernization Solutions
8. SOA Governance

Dismantling SOA Hype

Service-Oriented Architecture (SOA) has long since attracted a lot of buzz from every realm of the IT industry, and it remains a rapidly emerging technology. However, there is still some level of fear and anxiety among the IT community about SOA. Is SOA real? Do I need SOA? How is it done? You will be happy to know that SOA is a reality; it exists and you can do it as well.

For More Information:  
New technology, after being adopted by "technology enthusiasts" and "visionaries" is adopted by "pragmatists". Technology Acceptance Model is one of the most influential models to explain the process of technology adoption. According to TAM, SOA has to prove itself on two terms—perceived usefulness and perceived ease of use—for SOA to successfully cross the chasm and be adopted more widely. We think SOA has proven its effectiveness on both these fronts. Not only can SOA deliver on its promises of reusability and agility (usefulness) but it can also reduce the overall cost of ownership through the standards-based approach (ease of use). That is precisely the reason why SOA enables organizations to cut the cost of IT dramatically and use the resulting savings in building other innovations.

As organizations increase their SOA footprint, IT Managers, Architects, and developers are starting to realize that the impact of SOA on IT and business operations can be immense. After having gained confidence with web services, they want to take it to the next level. However, adopters are challenged with some basic questions: How do I SOA-enable my existing integration investment? Can I build flexible and agile business processes? How can I administer my SOA environment without spending a fortune? There have been various best practices defined around SOA. However, missing from the map is the real-world flavor. People want to learn from people. The IT community is looking for real-world examples; examples to gain an understanding of how other companies are embarking on an SOA initiative and apply industry learning to their projects.

What This Book Covers

The book begins with a refresher of SOA and the various types of integration available, and then delves deeper into integration best practices with XML, binding components and web services from Packt titles like "Oracle SOA Suite Developer's Guide" and "BPEL Cookbook". Along the way you'll also learn from a number of real world scenarios. By the end of "Do more with SOA Integration: Best of Packt" you will be equipped with knowledge from a wide variety of Packt books and will have learnt from a range of practical approaches to really get to grips with SOA integration.

Chapter 1: Basic Principles, covers the fundamental integration concepts. This chapter is intended as an introduction for specialists who have not yet dealt with the subject of integration.

For More Information:
Chapter 2: Integration Architecture, Principles, and Patterns, is an overview of the challenges in integration and why integration is one of the most difficult problems in application development. We will identify the best strategies for SOA-based integration and discuss top-down, bottom-up, and inside-out approaches. You will learn about different types of integration, such as data-level integration, application integration, business process integration, presentation integration and also, B2B integrations.

Chapter 3: Base Technologies, describes a selection of base technologies. By far the most important of these are transaction strategies and their implementation, as well as process modeling. In addition, Java EE Connector Architecture (JCA), Java Business Integration (JBI), Service Component Architecture (SCA), and Service Data Objects (SDO) are explained. Many other base technologies are used in real-life integration projects, but these go beyond the scope of this book.

Chapter 4: Best Practices for Using XML for Integration, discusses various design anomalies that may arise while designing XML schemas. Some of the broad categories covered in this chapter are design recommendations for architecting domain-specific XML schemas, tips for designing XML schemas with examples, using XSL effectively for translating Infosets from one form to another, securing XML documents with encryption and digital signature, and XML serialization and the differences between SAX, DOM, and StAX.

Chapter 5: Extending Enterprise Application Integration, This chapter focuses on very common business problem i.e. siloed applications and segregated data glued together using proprietary integration solution. How can we best leverage SOA to add value on top of existing integration infrastructure? By service-enabling existing data-integration processes, business processes could be easily automated by orchestrating underlying services. Infosys, a leading systems integrator, has helped many of its customers leverage their existing EAI investment, and explains you how to do exactly this. This chapter takes an example of broken customer data synchronization between Siebel and SAP, and outlines a strategy to automate this process by integrating with proprietary integration solutions like TIBCO and webMethods.

Chapter 6: Service-Oriented ERP Integration: Driven by the business requirements of different departments, countries, and subsidiaries, many organizations end up with multiple ERP systems. The result is data fragmentation and manual processes. This, in turn, leads to poor customer service and loss of revenue. The problem is how to address this problem without re-architecting the entire solution. Sierra Atlantic, a leading consulting firm specializing in integration technologies, encountered a similar issue with its client. In this chapter, Lawrence Pravin, Architect at Sierra Atlantic, takes an example of a broken sales order creation process. He walks you through a step-by-step approach to automate it across PeopleSoft HR and Oracle E-Business Suite using BPEL in a service-oriented approach.

For More Information:
Chapter 7: Service Engines, provides an overview of Java Business Integration (JBI) and the Enterprise Service Bus (ESB). You will learn about JBI Service Engines and how they are supported within the NetBeans IDE.

Chapter 8: Binding Components, introduces JBI Binding Components and how they provide protocol independent communication between JBI components. You will also learn about the support that the NetBeans IDE provides for Binding Components.

Chapter 9: SOA and Web Services Approach for Integration, discusses the architecture of web services and their benefits. This chapter provides an in-depth coverage of the various patterns that can be applied while creating SOA using web services. You will learn the essential differences between EAI and B2B and how to apply SOA integration techniques in this space. The chapter also discusses several guidelines for creating interoperable web services. Finally, a complete, albeit trivial, example of creating web services on the .NET and Java EE platforms is discussed.

Chapter 10: Service- and Process-Oriented Approach to Integration Using Web Services, takes a look at how ESB provides a concrete infrastructure for SOA, extending the simple services model to include a robust services bus with extensive mediation functionality.

Chapter 11: Loosely-coupling Services, describes how we can use the Oracle Service Bus to build services that are implementation agnostic. Doing so allows us to change the service location, communication protocol, or even replace a service implementation with another, with no impact on the client.

Chapter 12: Integrating BPEL with BPMN using BPM Suite, presents an interesting new feature provided by Oracle Business Process Management Suite 11g (Oracle BPM Suite 11g) - the BPMN 2.0 service engine. It allows direct execution of BPMN 2.0 processes, without the need to transform them to BPEL. The chapter discusses Oracle BPM Suite architecture and features, and demonstrates how both, BPMN and BPEL processes can be used inside a single SOA composite application.

Chapter 13: SOA Integration—Functional View, Implementation, and Architecture, focusses to place SOA in the context of Modernization. SOA Legacy Modernization will bring you legacy IT infrastructure into the world of World Wide Web, Web 2.0, and all the other latest Internet-based IT architectures. Within days, a legacy system can be accessed via a web browser. Your time-to-market using the Legacy SOA Integration approach is weeks, instead of months or years for some other modernization options.

For More Information:
Chapter 14: SOA Integration—Scenario in Detail, is an SOA Integration hands-on example using web enablement of mainframe COBOL/VSAM. We will use Java Server Pages (JSP), Java Database Connectivity (JDBC), the Oracle Legacy Adapter, Oracle Application Server, Java EE Connector API, and XA transaction processing to show a two-phase commit across an Oracle database and VSAM on the mainframe.

Appendix A: Establishing SOA Governance at Your Organization, provides a detailed overview of both the techniques explored in the Advasco story, as well as other options available to you and your organization.

For More Information:
In this chapter we will explore how service virtualization through the Service Bus of the Oracle SOA Suite can be used to deliver more loosely coupled services.

**Coupling**

Coupling is a measure of how dependent one service is upon another. The more closely one service depends on another service, the more tightly coupled they are. There have been a number of efforts to formalize metrics for coupling and they all revolve around the same basic items. These are:

- **Number of input data items**: This is basically the number of input parameters of the service.
- **Number of output data items**: It is the output data of the service.
- **Dependencies on other services**: It is the number of services called by this service.
- **Dependencies of other services on this service**: It is the number of services that invoke this service.
- **Use of shared global data**: It is the number of shared data items used by this service. This may include database tables or shared files.
- **Temporal dependencies**: Dependencies on other services being available at specific times.

For More Information:

Loosely Coupling Services

Let us examine how each of these measures may be applied to our service interface. The principles below are relevant to all services, but widely reused services have a special need for all of the items.

**Number of input data items**

A service should only accept, as input, the data items required to perform the service being requested. Additional information should not be passed into the service because this creates unnecessary dependencies on the input formats. This economy of input allows the service to focus on only the function it is intended to provide and does not require it to understand unnecessary data formats. The best way to isolate the service from changes in data formats that it does not use is not to require the service to be aware of those data formats.

For example, a credit rating service should only require sufficient information to identify the individual being rated. Additional information such as the amount of credit being requested or the type of goods or services for which a loan is required is not necessary for the credit rating service to perform its job.

Services should accept only the data required to perform their function and nothing more.

When talking about reducing the number of data items input or output from a service, we are talking about the service implementation, not a logical service interface that may be implemented using a canonical data model. The canonical data model may have additional attributes not required by a particular service, but these should not be part of the physical service interface.

**Number of output data items**

In the same way that a service should not accept inputs that are unnecessary for the function it performs, a service should not return data that is related only to its internal operation. Exposing such data as part of the response data will create dependencies on the internal implementation of the service that are not necessary.

Sometimes a service needs to maintain state between requests. **State** implies that state information must be maintained at least in the client of the service, so that it can identify the state required to the service when making further requests, but often the state information in the client is just an index into the state information held in the service. We will return to this subject later in this chapter.
Services should not return public data that relates to their own internal processing.

Dependencies on other services
Generally reuse of other services to create a new composite service is a good thing. However dependencies on other services does increase the degree of coupling because there is a risk that changes in those services may impact the composite service and hence any services with dependencies on the composite service. We can reduce the risk that this poses by limiting our use of functionality in other services to just that required by the composite.

Services should reduce the functionality required of other services to the minimum required for their own functionality.

For example, a dispatching service may decide to validate the address it receives. If this functionality is not specified as being required, because for example all addresses are validated elsewhere, then the dispatching service has an unnecessary dependency that may cause problems in the future.

Dependencies of other services on this service
Having a widely used service is great for reuse but the greater the number of services that make use of this service then the greater impact a change in this service will have on other services. Extra care must be taken with widely reused services to ensure that their interfaces are as stable as possible. This stability can be provided by following the guidelines in this section.

Widely reused services should focus their interface on just the functionality needed by clients and avoid exposing any unnecessary functions or data.

For More Information:
Use of shared global data

Shared global data in the service context is often through dependencies on a shared resource such as data in a database. Such use of shared data structures is subversive to good design because it does not appear in the service definitions and so the owners of the shared data may be unaware of the dependency. Effectively this is an extra interface into the service. If this is not documented then the service is very vulnerable to the shared data structure being changed unknowingly. Even if the shared data structure is well documented, any changes required must still be synchronized across all users of the shared data.

Avoid use of shared global data in services unless absolutely necessary. If it is absolutely necessary then the dependency needs to be clearly documented in all users of the shared data. A services data should only be manipulated through its defined interface.

Temporal dependencies

Not all service requests require an immediate response. Often a service can be requested and the response may be returned later. This is a common model in message based systems and allows for individual services to be unavailable without impacting other services. Use of queueing systems allows temporal or time decoupling of services, so that two communicating services do not have to be available at the same instant in time, the queue allowing messages to be delivered when the service is available rather than when the service is requested.

Use asynchronous message interfaces to reduce dependencies of one service on the availability of another.

For More Information:
Reducing coupling in stateful services

A stateful service maintains context for a given client between invocations. When using stateful services, we always need to return some kind of state information to the client. To avoid unnecessary coupling, this state information should always be opaque. By opaque we mean that it should have no meaning to the client other than as a reference that must be returned to the service when requesting follow on operations. We will examine how this may be accomplished later in this section.

A common use of state information in a service is to preserve the position in a search that returns more results than can reasonably be returned in a single response. Another use of state information might be to perform correlation between services that have multiple interactions such as between a bidding service and a bidding client.

Whatever the reason, the first question when confronted with the need for state in a service is to investigate ways to remove the state requirement. If there is definitely a need for state to be maintained then there are two approaches that can followed by the service:

- Externalize all state and return it to the client.
- Maintain state within the service and return a reference to the client.

In the first case it is necessary to package up the required state information and return it to the client. Because the client should be unaware of the format of this data it must be returned as an opaque type. This is best done as an <any> element in the schema for returning the response to the client. An <any> element may be used to hold any type of data from simple strings through to complex structured types.

For example if a listing service returns only twenty items at a time then it must pass back sufficient information to enable it to retrieve the next twenty items in the query.

In the XML Schema example below we have the XML data definitions to support two operations on a listing service. These are:

- searchItems
- nextItems

For More Information:
The searchItems operation will take a searchItemsRequest element for input and return a searchItemsResponse element. The searchItemsResponse has within it a searchState element. This element is a sequence that has an unlimited number of arbitrary elements. This can be used by the service to store sufficient state to allow it to deliver the next twenty items in the response. It is important to realize that this state does not have to be understood by the client of the service. The client of the service just has to copy the searchState element to the continueSearchItemsRequest element to retrieve the next set of 20 results.

The approach above has the advantage that the service may still be stateless, although it gives the appearance of being stateful. The sample schema shown in the following figure could be used to allow the service to resume the search where it left off without the need for any internal state information in the service. By storing the state information (the original request and the index of the next item to be returned) within the response, the service can retrieve the next set of items without having to maintain any state within itself. Obviously the service for purposes of efficiency could maintain some internal state, such as a database cursor, for a period of time, but this is not necessary.

For More Information:
An alternative approach to state management is to keep the state information within the service itself. This still requires some state information to be returned to the client, but only a reference to the internal state information is required. In this case there are a couple of options for dealing with this reference.

One is to take state management outside of the request/response messages and make it part of the wider service contract, either through the use of WS-Correlation or an HTTP cookie for example. This approach has the advantage that the service can generally take advantage of state management functions of the platform, such as support for Java services to use the HTTP session state.

**Use of WS-Correlation**

It is possible to use a standard correlation mechanism such as WS-Correlation. This is used within SOA Suite by BPEL to correlate process instances with requests. If this approach is used, however, it precludes the use of the externalized state approach discussed earlier. This makes it harder to swap out your service implementation with one that externalizes all its state information. In addition to requiring your service to always internalize state management, no matter how it is implemented, your clients must now support WS-Correlation.
Loosely Coupling Services

The alternative is to continue to keep the state management in the request/response messages and deal with it within the service. This keeps the client unaware of how state is managed because the interface is exactly the same for a service that maintains internal state and a service that externalizes all state. A sample schema for this is shown in the following figure. Note that unlike the previous schema there is only a service specific reference to its own internal state. The service is responsible for maintaining all the required information internally and using the externalized reference to locate this state information.

The OSB (Oracle Service Bus) in SOA Suite enables us to have services use their native state management and still expose it as service specific state management that is more abstract and hence less tightly coupled to the way state is handled.

Some web service implementations allow for stateful web services, with state managed in a variety of proprietary fashions. For example, the Oracle Containers for J2EE (OC4J) used cookies for native state management, other platforms can use different mechanisms.

We want to use native state management when we internalize session state because it is easier to manage, and the container will do the work for us using mechanisms native to the container. However this means that the client has to be aware that we are using native state management because the client must make use of these mechanisms. We want the client to be unaware of whether the service uses native state management, its own custom state lookup mechanism, or externalizes all session state into the messages flowing between the client and the service. The latter two can look the same to the client and hence make it possible to switch services with different approaches. However, the native state management explicitly requires the client to be aware of how the state is managed.

For More Information:
To avoid this coupling we can use the OSB to wrap the native state management services as shown in the following diagram. The client passes a session state element of unknown contents back to the service façade which is provided by the OSB. The OSB then removes the session state element and maps it onto the native state management used by the service, such as placing the value into a session cookie. Thus we have the benefits of using native state management without the need for coupling the client to a particular implementation of the service. For example, an OC4J service may use cookies to manage session state, by having the OSB move the cookie value to a field in the message we avoid clients of the service having to deal with the specifics of OC4J state management.

Oracle Service Bus design tools
The Oracle Service Bus can be configured either using the Oracle Workshop for WebLogic or the Oracle Service Bus Console.

Oracle workshop for WebLogic
Oracle Workshop for WebLogic provides tools for creating all the artifacts needed by the Oracle Service Bus. Based on Eclipse, it provides a rich design environment for building service routings and transformations for deployment to the service bus. In future releases it is expected that all the service bus functionality in the Workshop for WebLogic will be provided in JDeveloper. Note that there is some duplication functionality between JDeveloper and Workshop for WebLogic. In some cases, such as WSDL generation, the functionality provided in the Workshop for WebLogic is superior to that provided by JDeveloper. In other cases, such as XSLT generation, the functionality provided by JDeveloper is superior.
Loosely Coupling Services

Oracle Service Bus Console

**Oracle Service Bus Console** provides a web-based interface for creating, managing, and monitoring all service bus functions. In this chapter we will focus on using the Service Bus Console. Changes to a service bus configuration are grouped together in a session using the change center. Before any changes are made, it is necessary to **Create** a session from within the change centre. When the changes are complete, they are applied by clicking **Activate**. Clicking **Discard** will cause the changed state to be discarded.

![Change Center](image)

**Service Bus overview**

The Oracle SOA Suite includes the Oracle Enterprise Service Bus which runs on multiple vendors application servers, and the Oracle Service Bus which in the current release only runs on Oracle WebLogic Server. In the future the Oracle Service Bus may also run on other vendors' applications servers. If you plan on deploying SOA Suite onto non-WebLogic applications servers, then you should use the Oracle Enterprise Service Bus. Most users however run the SOA Suite on Oracle application servers and would be better off using the Oracle Service Bus as it is more functional than the Oracle Enterprise Service Bus and is the stated strategic service bus for Oracle SOA Suite.

**Service Bus message flow**

It is useful to examine how messages are processed by the service bus. Messages normally target an endpoint in the service bus known as a proxy service. Once received by the proxy service, the message is processed through a series of input pipeline stages. These pipeline stages may enrich the data by calling out to other web services, or they may transform the message as well as providing logging and message validation. Finally, the message reaches a routing step where it is routed to a service known as a business service. The response, if any from the service, is then sent through the output pipeline stages which may also enrich the response or transform it before returning a response to the invoker.

For More Information:

Note that there may be no pipeline stages and the router may make a choice between multiple endpoints. Finally note that the business service is a reference to the target service which may be hosted within the service bus or as a standalone service. The proxy service may be thought of as the external service interface and associated transforms required to make use of the actual business service.

**Virtualizing service endpoints**

To begin our exploration of the Oracle Service Bus, let us start by looking at how we can use it to virtualize service endpoints. By virtualizing a service endpoint we mean that we can move the location of the service without affecting any of the services' dependents.

We will use an address lookup service as our sample. To begin we create a new Service Bus Project. After logging onto the console and before we start making changes in the console we need to create a new session, as described in the previous section. We can then select the Project Explorer tab and create a new project by entering a project name and clicking the Add Project button. This creates a new Service Bus Project within the console.

For More Information:
We can now start adding items to our project by clicking on the project name either in the **Project Explorer** tab or in the list of projects under the project creation dialogue. Before adding any items to the project, it is considered good practice to organize the items into folders reflecting their type. So we will use the **Add Folder** dialogue to create folders for the various artifacts we require. We will create a **ProxyService** folder to hold the externally callable service provided by the service bus, a **BusinessService** folder to hold the backend services called by the service bus, and a WSDL folder to hold service definitions (WSDL) files.

Moving service location

To virtualize the address of our service we first need to add the service definition to the project. In this case we will be adding a web service so we start by adding a service description (WSDL file) to define the service. Before adding the WSDL file, we select the WSDL folder in our project. We then select the appropriate resource type from the Create Resource dialogue. We select the WSDL resource type from the Interface section and are presented with a dialogue enabling us to load the WSDL into the project. We then browse for the WSDL definition and then select Save to add it to the project. This registers the WSDL with the internal service bus repository.

For More Information:

Note that the dialogue requires us to provide a name for the WSDL file and optionally a description. The large blank section in the dialog is used to display the current WSDL file details and is only populated after the WSDL file has been saved.

**Endpoint address considerations**

When specifying endpoints in the service bus, it is generally not a good idea to use localhost or 127.0.0.1. Because the service bus definitions may be deployed across multiple nodes there is no guarantee that business service will be co-located with the service bus on every node the service bus is deployed upon. Therefore it is best to ensure that all endpoint addresses use actual hostnames.

---

For More Information:

Loosely Coupling Services

Now that we have a service definition loaded, we can use this to create the business service, or ultimate endpoint of the service. We do this by changing directory to the BusinessService directory and creating a resource of type Business Service. This brings up a screen allowing us to configure the business service.

![Business Service Configuration Screen]

We provide a name for the service and identify the type of service to be accessed. Note that we are not limited to services described by WSDL. In addition to already defined business and proxy services, we can base our service on XML or messaging systems. The easiest to use is the WSDL web service. Browsing for a WSDL web service brings up a dialogue listing all the WSDL documents known to the service bus. We can search this list to select the WSDL document we want to use.

![WSDL Selection Screen]

For More Information:
When we select the WSDL name that we want to use, we are taken to another dialogue that introspects the WSDL, identifies any ports or bindings and asks us which one we wish to use. Bindings are mappings of the WSDL service onto a physical transport mechanism such as SOAP over HTTP. Ports are the mapping of the binding onto a physical endpoint such as a specific server.

Note that if we choose a port we do not have to provide physical endpoint details later in the definition of the business service, although we may choose to do so. If we choose a binding, because it doesn't include a physical endpoint address, we have to provide the physical endpoint details explicitly. Once we have highlighted the port or binding we want to use we hit the Submit button.

At this point if we have chosen a binding, we can hit last to review the final configuration of the business service. If, however, we chose a port or we wish to change the physical service endpoint, or add additional physical service endpoints, then we hit the Next button to allow us to configure the physical endpoints of the service.
Loosely Coupling Services

This dialogue allows us to do several important things:

- Modify the Protocol to support a variety of transports.
- Choose a Load Balancing Algorithm. If there is more than one endpoint URI then the service bus will load balance across them according to this algorithm.
- Change, add, or remove Endpoint URIs or physical targets.
- Specify retry logic, specifically the Retry Count, the Retry Iteration Interval and whether or not to Retry Application Errors (errors generated by the service called, not the transport).

Note that the ability to change, add, and remove physical endpoint URIs as well as change the protocol used. This allows us to change the target services without impacting any clients of the service, providing us with virtualization of our service location.

Clicking Last enables us to review the configuration of this business service and if we are happy with it, then clicking Save will create or update our business service.

| Create a Business Service - Summary (Addressbook/Service/LocalAddressbookService) |
|------------------------------------------|------------------------------------------|
| **General Configuration**                | LocalAddressbookService                  |
| Service Name                             | LocalAddressbookService                  |
| Description                              | LocalAddressbookService                  |
| Service Type                             | Web Service - SOAP 1.1 (WSDL: Addressbook/Service/LocalAddressbookService) |
| Transport Configuration                  | Round Robin                               |
| Protocol                                 | http                                      |
| Load Balancing Algorithm                 | round robin                               |
| Endpoint URI                             | http://localhost:8080/Chapter4-SampleServices-context-root/AddressbookServicePort |
| Retry Count                              | 0                                         |
| Retry Iteration Interval                 | 30                                        |
| Retry Application Errors                 | Yes                                       |
| HTTP Transport Configuration             | HTTP Request Method POST                  |
| Timeout                                 | 0                                         |
| Authentication                          | None                                      |
| Follow HTTP redirects                   | ENABLED                                   |
| SOAP Binding Configuration              | Enforce WSDL Compliance No                |

For More Information:  
Now that we have created our business service we need to expose it through the service bus by adding a proxy service. We do this in a similar fashion to creating the business service by creating a resource of type proxy service in the ProxyService folder. If we are only virtualizing the location of a service then the proxy service can use the same WSDL definition as the business service. Again we can choose to use either a port or a binding from the WSDL for a service definition. In either case the endpoint URI is determined by the service bus, although we can change its location and transport. Once we have created the new proxy service we then need to link the proxy service to the business service.

For More Information:
Loosely Coupling Services

We link the proxy service to business services by selecting the newly created proxy service and clicking on the message flow icon to activate the message flow editor. Within the message flow editor we want to create a routing to the business service so we click on Add Route. This creates a new route entry in the message flow.

```
<table>
<thead>
<tr>
<th>Add Message Flow</th>
<th>AddressLookup/ProxyService/AddressLookupService</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>Cancel</td>
</tr>
<tr>
<td>Clear</td>
<td>Save All</td>
</tr>
<tr>
<td>Cancel All</td>
<td></td>
</tr>
</tbody>
</table>
```

Having created the new route node we can now edit it by clicking on it. We can choose to edit the name of the node and, more interestingly, we can choose to edit the routing details.

```
<table>
<thead>
<tr>
<th>AddressLookupService</th>
</tr>
</thead>
<tbody>
<tr>
<td>Route</td>
</tr>
<tr>
<td>Edit Route</td>
</tr>
<tr>
<td>Add Pipeline Par</td>
</tr>
<tr>
<td>Add Route</td>
</tr>
<tr>
<td>Add Conditional Branch</td>
</tr>
<tr>
<td>Add Operational Branch</td>
</tr>
<tr>
<td>Add Service Error Handler</td>
</tr>
<tr>
<td>Paste Route</td>
</tr>
</tbody>
</table>
```

Selecting Edit Route brings up the route node editor which allows us to configure the routing details. Clicking on Add an Action allows us to choose the type of Communication we want to add. Flow Control allows us to add If ... Then ... logic to our routing decision. However in most cases the Communication items will provide all the flexibility we need in our routing decisions. This gives us three types of routing to apply:

For More Information:
- **Dynamic Routing**: It allows us to route to the result of an XQuery. This is useful if the endpoint address is part of the input message.
- **Routing**: It allows us to select a single static endpoint.
- **Routing Table**: It allows us to use an XQuery to route between several endpoints. This is useful when we want to route to different services based on a particular attribute of the input message.

For simple service endpoint virtualization we only require the **Routing** option.

Selecting the **Routing** option then enables us to configure the route by first selecting a service by clicking on the `<Service>` label. This brings up a dialog from which we can select our target endpoint, usually a previously defined business service.

Having selected a target endpoint, we can then configure how we use that endpoint. In the case of simple location virtualization then the proxy service and the business service endpoint are the same and so we can just pass on the input message directly to the business service. Later on we will look at how to transform data to allow virtualization of the service interface.

---

For More Information:

Loosely Coupling Services

To provide a simple pass through function we can check the Use inbound operation for outbound checkbox. This means that the operation that is requested on the proxy service, the inbound operation, will be invoked on the business service, the outbound operation. Because there is no need to transform the data or perform additional operations we can now save our message flow and activate our changes by clicking activate in the session dialog.

Selecting a service to call

We can further virtualize our endpoint by routing different requests to different services, based upon the values of the input message. For example, we may use one address lookup service for addresses in our own country and another service for all other addresses. In this case, we would use the routing table option on the add action to provide a list of possible service destinations.

The routing table enables us to have a number of different destinations and the message will be routed based the value of an expression. When using a routing table all the services must be selected based on the same expression, the comparison operators may vary but the actual value being tested against will always be the same. If this is not the case then it may be better to use if ... then ... else routing. The routing table may be thought of as a switch statement and as with all switch statements, it is good practice to add a default case.

In the routing table we can create additional cases, each of which will have a test associated with it. Note that we can also add the default case.

For More Information:
We need to specify the expression to be used for testing against. Clicking on the `<Expression>` link takes us to the XQuery/XSLT Expression Editor. By selecting the Variable Structures tab and selecting a new structure we can find the input body of the message which enables us to select the field we wish to use as the comparison expression in our routing table.

When selecting in the tab on the left of the screen the appropriate xpath expression should appear in the Property Inspector window. We can then click on the XQuery Text area of the screen prior to clicking on the Copy Property to transfer the property xpath expression from the property inspector to the XQuery Text area. We then complete our selection of the expression by clicking the Save button.

In the example we are going to route our service based on the country of the address. In addition to the data in the body of the message, we could also route based on other information from the request, or by using a message pipeline, we could base our lookup on data external to the request.
Loosely Coupling Services

Once we have created an expression to use as the basis of comparison for routing then we select an operator and a value to use for the actual routing comparison. In the example below, if the country value from the expression matches the string `uk` (include the quotes) then the local address lookup service will be invoked. Any other value will cause the default service to be invoked, as yet undefined in the following example:

Once the routing has been defined then it can be saved as shown earlier in this chapter.

Note that we have shown a very simple routing example. The service bus is capable of doing much more sophisticated routing decisions. A common pattern is to use a pipeline to enrich the inbound data and then route based on the inbound data. For example a pricing proxy service may use the inbound pipeline to look up the status of a customer, adding that status to the data available as part of the request. The routing service could then route high value customers to one service and low value customers to another service, based on the looked up status. In this case the routing is done based on a derived value rather than on a value already available in the message.
In summary, a request can be routed to different references based on the content of the request message. This allows messages to be routed based on geography or pecuniary value for example. This routing, because it takes place in the composite, is transparent to clients of the composite and so aids us in reducing coupling in the system.

Virtualizing service interfaces

We have looked at how to virtualize a service endpoint. Now let us look at how we can further virtualize the service by abstracting its interface into a common format, known as canonical form. This will provide us further flexibility by allowing us to change the implementation of the service with one that has a different interface but performs the same function. The native format is the way the data format service actually uses, the canonical format is an idealized format that we wish to develop against.

Physical versus logical interfaces

Best practice for integration projects was to have a canonical form for all messages exchanged between systems. The canonical form was a common format for all messages. If a system wanted to send a message then it first needed to transform it to canonical form before it could be forwarded to the receiving system who would then transform it from canonical form to its own representation. This same good practice is still valid in a service-oriented world and the Service Bus is the mechanism SOA Suite provides for us to do this.

The benefits of a canonical form are that:

- Transformations are only necessary to and from canonical form, reducing the number of different transformations required to be created.
- Decouples format of data from services, allowing a service to be replaced by one providing the same function but a different format of data.

This is illustrated graphically by a system where two different clients make requests for one of four services, all providing the same function but different implementations. Without canonical form we would need a transformation of data between the client format and the server format inbound and again outbound. For four services this yields eight transformations, and for two clients this doubles to sixteen transformations.

For More Information:
Loosely Coupling Services

Using canonical format gives us two transformations for each client, inbound and outbound to canonical form; with two clients this gives us four transformations. To this we add the server transformations to and from canonical form, of which there are two per server giving us eight transformations. This gives us a total of twelve transformations that must be coded up rather than sixteen if we were using native to native transformation.

The benefits of canonical form are most clearly seen when we deploy a new client. Without canonical form we would need to develop eight transformations to allow the client to work with the four different possible service implementations. With canonical form we only need two transformations, to and from canonical form.

Let us look at how we implement canonical form in Oracle Service Bus.

Mapping service interfaces
In order to take advantage of canonical form in our service interfaces, we must have an abstract service interface that provides the functionality we need without being specific to any particular service implementation. Once we have this we can then use it as the canonical service form.
We set up the initial project in the same way as we did in the previous section on virtualizing service endpoints. The proxy should provide the canonical interface, the business service providing the native service interface. Because the proxy and business services are not the same interface we need to do some more work in the route configuration.

We need to map the canonical form of the address list interface onto the native service form of the interface. In the example, we are mapping our canonical interface to the interface provided by a web-based address solution provided by Harte-Hanks Global Address (http://www.qudox.com). To do this we create a new service bus project and add the Harte-Hanks WSDL (http://webservices.globaladdress.net/globaladdress.asmx?WSDL). We use this to define the business service. We also add the canonical interface WSDL that we have defined and create a new proxy with this interface. We then need to map the proxy service onto the Harte-Hanks service by editing the message flow associated with the proxy, as we did in the previous section.

Our mapping needs to do two things:

- Map the method name on the interface to the correct method in the business service.
- Map the parameters in the canonical request onto the parameters needed in the business service request.

For each method on the canonical interface we must map it onto a method in the physical interface. We do this by selecting the appropriate method from the business service operation drop-down box. We need to do this because the methods provided in the external service do not match the method names in our canonical service. In the example we have mapped onto the `SearchAddress` method.

For More Information:
Loosely Coupling Services

Having selected an operation we now need to transform the input data from the format provided by the canonical interface into the format required by the external service. We need to map the request and response messages if it is a two-way method, or just the request message for one-way methods. The actual mapping may be done either by XQuery or XSLT. In our example we will use the XSLT transformation.

To perform the transformation we add a messaging processing action to our message flow, in this case a replace operation. The variable body always holds the message in the service bus flow; this receives the message through the proxy interface and is also used to deliver the message to the business service interface. This behavior differs from BPEL and most programming languages where we typically have separate variables for the input and output messages. We need to transform this message from the proxy input canonical format to the business service native output format.

Be aware that there are really two flows associated with the proxy service. The request flow is used to receive the inbound message and perform any processing before invoking the target business service. The response flow takes the response from the business service and performs any necessary processing before replying to the invoker of the proxy service.

On selecting replace we can fill in the details in the replace action dialogue. The message is held in the body variable and so we can fill this (body) in as the target variable name. We then need to select which part of the body we want to replace.

For More Information:
Clicking on the XPath link brings up the XPath Expression Editor where we can enter the portion of the target variable that we wish to replace. In this case we wish to replace all the elements so we enter ./* which selects the top level element and all elements beneath it. Clicking Save causes the expression to be saved in the replace action dialogue.

Having identified the portion of the message we wish to replace, all of it, we now need to specify what we will replace it with. In this case we wish to transform the whole input message, so we click on the Expression link and select the XSLT Resources tab. Clicking Browse enables us to choose a previously registered XSLT transformation file. After selecting the file we need to identify the input to the transformation. In this case the input message is in the body variable and so we select all the elements in the body by using the expression $body/*. We then save our transformation expression.

For More Information:
Loosely Coupling Services

Having providing the source data, the target, and the transformation we can then save the repeat the whole process for the response message, in this case converting from native to canonical form.

We can use JDeveloper to build an XSLT transformation and then upload it into the service bus; a future release will add support for XQuery in JDeveloper similar to that provided in Oracle Workshop for WebLogic. XSLT is an XML language that describes how to transform one XML document into another. Fortunately most XSLT can be created using the graphical mapping tool in JDeveloper, and so SOA Suite developers don’t have to be experts in XSLT, although it is very useful to know how it works. Note that in our transform we may need to enhance the message with additional information, for example the Global Address methods all require a username and password to be provided to allow accounting of the requests to take place. This information has no place in the canonical request format but must be added in the transform. A sample transform is shown that does just this.
Note that we use XPath string functions to set the username and password fields. It would be better to set these from properties or an external file as usually we would want to use them in a number of calls to the physical service. XPath functions are available to allow access to composite properties. We actually only need to set five fields in the request, a country, postcode, username, password, and maximum number of results to return. All the other fields are not necessary for the service we are using and so are hidden from end users because they do not appear in the canonical form of the service.
Applying canonical form in the service bus

When we think about canonical form and routing, we have several different operations that may need to be performed. These are:

- Conversion to/from native business service form and canonical proxy form.
- Conversion to/from native client form to canonical proxy form.
- Routing between multiple native services, each potentially with its own message format.

The following diagram represents these different potential interactions as distinct proxy implementations in the service. To reduce coupling and make maintenance easier, each native service has a corresponding canonical proxy service, which isolates the rest of the system from the actual native formats. This is shown in the following figure in the Local-Harte-Hanks-Proxy and Local-LocalAddress-Proxy services that transform the native service to/from canonical form. This approach allows us to change the native address lookup implementations without impacting anything other than the Local-Proxy service.

The Canonical-Address-Proxy has the job of hiding the fact that the address lookup service is actually provided by a number of different service providers each with their own message formats. By providing this service we can easily add additional address providers without impacting the clients of the address lookup service.

In addition to the services shown in the figure we may have clients that are not written to use the canonical address lookup. In this case we need to provide a proxy that transforms the native input request to/from the canonical form. This allows us to be isolated from the requirements of the clients of the service. If a client requires its own interface to the address lookup service we can easily provide that through a proxy without the need to impact the rest of the system, again reducing coupling.

For More Information:
An important optimization

The above approach provides a very robust way of isolating service consumers and service requestors from the native formats and locations of their partners. However there must be a concern about the overhead of all these additional proxy services and also about the possibility of a client accessing a native service directly. To avoid these problems the service bus provides a local transport mechanism that can be specified as part of the binding of the proxy service. The local transport provides two things for us:

- It makes services only consumable by other services in the service bus; they cannot be accessed externally.
- It provides a highly optimized messaging transport between proxy services, providing in-memory speed to avoid unnecessary overhead in service hand-offs between proxy services.

These optimizations mean that it is very efficient to use canonical form, and so the service bus not only allows us great flexibility in how we decouple our services from each other but it also provides a very efficient mechanism for us to implement that decoupling. Note though that there is a cost involved in performing XSLT or XQuery transformations; this cost may be viewed as the price of loose coupling.

Summary

In this chapter, we have explored how we can use the Oracle Service Bus in the SOA Suite to reduce the degree of coupling. By reducing coupling, or the dependencies between services, our architectures become more resilient to change. In particular, we looked at how to use the Service Bus to reduce coupling by abstracting endpoint interface locations and formats. Crucial to this is the concept of canonical or common data formats that reduce the amount of data transformation that is required, particularly in bringing new services into our architecture. Finally we considered how this abstraction can go as far as hiding the fact that we are using multiple services concurrently by allowing us to make routing decisions at run time.

All these features are there to help us build service-oriented architectures that are resilient to change and can easily absorb new functionality and services.

For More Information:
Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.