Beginning WSO2 ESB

A comprehensive beginner to expert guide for learning WSO2 ESB 5.0

First Edition

Kasun Indrasiri

Apress®

Beginning WSO2 ESB

First Edition

Kasun Indrasiri

Apress®

Beginning WSO2 ESB

Kasun Indrasiri
San Jose, California, USA

ISBN-13 (pbk): 978-1-4842-2342-0 ISBN-13 (electronic): 978-1-4842-2343-7
DOI 10.1007/978-1-4842-2343-7

Library of Congress Control Number: 2016961319

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr
Lead Editor: Pramila Balan
Technical Reviewer: Isuru Udana
Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black,
Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John,
Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao,
Gwenan Spearing
Coordinating Editor: Prachi Mehta
Copy Editor: Kezia Endsley
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

Contents at a Glance

About the Author xi

About the Technical Reviewer xiii

Acknowledgments xv

■ Chapter 1: Introduction to WSO2 ESB 1

■ Chapter 2: Getting Started with WSO2 ESB 17

■ Chapter 3: Fundamentals of WSO2 ESB 29

■ Chapter 4: Processing Messages with WSO2 ESB 59

■ Chapter 5: Integrating SOAP and RESTful Web Services 105

■ Chapter 6: Enterprise Messaging with JMS, AMQP,
MQTT, and Kafka 133

■ Chapter 7: File-Based Integration 161

■ Chapter 8: Integrating Applications, Cloud Services,
and Data 179

■ Chapter 9: Security in WSO2 ESB 221

■ Chapter 10: Development and Deployment Methodology 239

■ Chapter 11: Administrating and Extending WSO2 ESB 267

Index 281

Contents

About the Author xi

About the Technical Reviewer xiii

Acknowledgments xv

■ Chapter 1: Introduction to WSO2 ESB 1

What is an ESB? 1

Core Functionalities of an ESB 4

Why WSO2 ESB? 4

Interoperability and EIP Support: Connecting Anything to Anything 5

Performance and Stability: The Fastest Open Source ESB 5

The Platform Advantage: Part of the WSO2 Middleware Platform 6

How does WSO2 ESB Work? 7

Functional Components 8

Summary 15

■ Chapter 2: Getting Started with WSO2 ESB 17

Designing a Simple Integration Scenario with WSO2 ESB 17

Building the Integration Scenario 18

Creating a HTTP Service/API in WSO2 ESB 19

Creating the Request Sent to the Backend Service 20

Sending the Request to the Backend Service 22

Transforming and Sending the Response Back to the Client 23

Try it Out 26

Summary 27

■Chapter 3: Fundamentals of WSO2 ESB 29

Message Entry Points 30

Using Proxy Services 31

Using APIs/HTTP Services 35

Using Inbound Endpoints 38

Message Processing Unit: Sequence and Mediators 43

Message Exit Points: Outbound Endpoints 47

Endpoint Types 48

Understanding Endpoint States and Endpoint Attributes 51

Scheduled Tasks 55

Summary 57

■Chapter 4: Processing Messages with WSO2 ESB 59

Pass-Through Messaging 61

Message Filtering and Switching 64

Message Filtering 67

Message Switching 67

Message Transformations 70

Using PayloadFactory Mediator 72

Using PayloadFactory and For-Each Mediator 74

Data Mapper Mediator 78

Using XSLT Mediator 78

Using the Header Mediator 81

Message Enriching 82

Message Validation 84

Service Orchestration 87

Service Chaining 87

Split and Aggregate Pattern 91

Clone and Aggregate Pattern 94

Changing the Message Protocol and Format 96

Protocol Conversions 96

Message Format Conversions 98

Using Properties in the Message Flow 100

Set/Retrieve Variables in the Message Flow 100

Use Predefined Properties to Control Message Flow 102

Summary 102

■ **Chapter 5: Integrating SOAP and RESTful Web Services 105**

Understanding SOAP and RESTful Web Services 105

Integrating SOAP Web Services 109

Exposing a SOAP Web Service Interface from WSO2 ESB 109

Integrating RESTful Web Services 118

Exposing RESTful Services/APIs with WSO2 ESB 118

Invoking RESTful Services from WSO2 ESB 126

Summary 131

■ **Chapter 6: Enterprise Messaging with JMS, AMQP, MQTT, and Kafka 133**

Integration with JMS-Based MoM 135

ESB as a JMS Consumer 135

ESB as a JMS Producer 140

Two-Way JMS 142

Using JMS Transactions 145

Store and Forward with Message Stores and Message Processors 147

Integrating with AMQP, MQTT and Kafka 151

Using AMQP with RabbitMQ 151

Integrating with MQTT 155

Integrating with Kafka 157

Summary 159

■Chapter 7: File-Based Integration.. 161
Reading Files.. 161
Reading a File from the Local File System 161
Reading Files from an FTP or FTP/s 163
Reading Files from an SFTP..165
FTP or SFTP Through a Proxy Server 165
Writing Files 165
Writing Files with VFS Transport 166
Transferring Files 168
Message Transformation with File Integration 170
File Connector 172
Protocol Transformation from File to JMS 175
Summary.. 177
■Chapter 8: Integrating Applications, Cloud Services, and Data 179
Integrating Proprietary Systems 179
SAP Integration 180
HL7 Integration..189
WebSockets Support 198
In-JVM Calls with Local Transport 209
Integrating Cloud Services 210
What is an ESB Connector?..210
Structure of an ESB Connector 211
Using an ESB Connector..211
Integrating Salesforce and SAP 216
Data Integration 218
Summary.. 220

■ **Chapter 9: Security in WSO2 ESB 221**

Transport Level Security 221

One-Way SSL (Server Authentication) 222

Two-Way SSL (Mutual/Client Authentication) 222

Using TLS/SSL with WSO2 ESB 223

Application Level Security 230

Securing REST APIs 230

Securing Proxy Services 231

Invoking Secured Service 233

Summary 237

■ **Chapter 10: Development and Deployment Methodology 239**

Development Methodology 239

Using the WSO2 ESB Development Tool 239

Data Mapper 249

Mediation Debugger 253

Deploying Artifacts Across Multiple Environments 259

Deployment Methodology 261

Summary 266

■ **Chapter 11: Administrating and Extending WSO2 ESB 267**

WSO2 ESB Analytics 267

Monitoring 274

Extending WSO2 ESB 276

Class Mediator 276

Script Mediator 277

Custom Connector 277

Other Extensions 278

Error Handling 278

Summary 279

Index 281

About the Author

Kasun Indrasiri is the Director of Integration Technologies at WSO2, a company that produces open source middleware solutions in enterprise integration, API management, security, and IoT domains. He currently provides the Architectural Leadership for the WSO2 integration platform.

Kasun has worked as a Software Architect and a Product Lead of WSO2 ESB with over seven years of experience with WSO2 ESB. He is an elected member of the Apache Software Foundation and a Project Management Committee member and a committer for the Apache Synapse open source ESB project. Kasun has provided Architectural and Technology Consulting for numerous customers in the United States, Europe, and Australia.

He researched high-performance message processing systems and contributed to an ACM publication called "Improved Server Architecture for Highly Efficient Message Mediation." Kasun holds an M.Sc. degree in Software Architecture and a B.Sc. Engineering degree in Computer Science and Engineering from the University of Moratuwa, Sri Lanka.

About the Technical Reviewer

Isuru Udana is a Technical Lead at WSO2 who mainly focuses on enterprise integration. He has more than five years of experience with the WSO2 enterprise service bus product as a Core Developer. Isuru is one of the Product Leads of the WSO2 ESB, and he provides Technical Leadership to the project. He is a committer and a PMC member for the Apache Synapse open source ESB project. Isuru graduated from the Department of Computer Science and Engineering, University of Moratuwa, Sri Lanka. As his final year project, he worked on Siddhi, a high performance complex event processing engine, which now ships with WSO2 CEP server. Isuru is an open source enthusiastic who has participated in the "Google Summer of Code" program as a student as well as a mentor in the last five years.

Acknowledgments

I would like to thank Apress for giving me the opportunity to write this book and, in particular, I must thank Pramila Balan, Acquisitions Editor and Prachi Mehta, Coordinating Editor, who have been constantly guiding and helping me from the very beginning of the writing process. Also I would like to thank Isuru Udana, the Technical Reviewer of this book. His expertise, knowledge, and feedback were quite useful to improve the technical content of this book.

I must thank Dr. Sanjiva Weerawarana who is the Founder, CEO, and Chief Architect of WSO2, for all the guidance he provided throughout all these years at WSO2. Also, I'm grateful to Prabath Siriwardana, who gave me the initial idea for writing a book, and for encouraging me with all his experiences on writing a book.

I'm grateful to my beloved wife Imesha, my parents, and my sister, who are the main driving forces behind all my success.

Last but not least, thank you to everyone who supported me in many different ways.

CHAPTER 1

Introduction to WSO2 ESB

Nowadays successful enterprises rely heavily on the underlying software applications they use. To fulfill diverse business needs, the enterprises have to pick and choose different software application and services, which are built with disparate technologies, use varying standards, and are built by different vendors. When building IT solutions for business scenarios, the enterprises have to make these disparate software applications and services work together to produce unified business functionality.

The task of plumbing different software applications, services, and systems, and forming new software solutions out of that is known as *enterprise integration*. The software application that is designed to enable that task is known as the Enterprise Service Bus (ESB). An ESB enables diverse applications, services, and systems to talk to each other, interact, and transact. It acts as the main messaging backbone in any Service Oriented Architecture (SOA); it's lightweight, built on top of open standards such as web services standards, and supports commonly used patterns in enterprise integration known as Enterprise Integration Patterns (EIP—for more information, see www.eaipatterns.com).

What is an ESB?

Let's suppose an organization is running a financial business and has web services, which expose underlying business functionalities such as providing to its customers information on stock quotes (the price of a stock as quoted on a stock exchange) for a given company. They want to expand the business by enabling mobile users to use the online store by making these business functions accessible on mobile devices.

But the mobile devices are inherently based on message formats such as JSON while the backend web service only supports the SOAP message format. The financial organization has to integrate these two systems, which are using disparate message formats, to work together to achieve its business objectives.

To solve this enterprise integration problem, someone could possibly modify either the mobile device application or the backend service to convert one message format to the message format that's understood by the other party. But this approach has several drawbacks. By modifying either the backend web service or the mobile application per the requirements of the other party, the two systems are tightly coupled to the same message format. If the backend service or the mobile application changes its message format, the company is forced to change the code of the other system. Also, if we have to

K. Indrasiri, *Beginning WSO2 ESB*, DOI 10.1007/978-1-4842-2343-7_1

further extend the business use case to include another backend service, then we need to wire all three systems with point-to-point links so that each system is connected to every other system. A change in one of these systems could break the entire business use case.

Therefore, you need a better way to integrate these systems with no modifications at the backend or the client, as well as use a configuration-based approach to integrate these systems without writing any code.

The ESB can be used as the intermediary integration layer between two or more disparate systems and services. Therefore, as illustrated in Figure 1-1, the ESB can be placed between the JSON-based mobile application and the SOAP-based web service. Without writing any code, you can configure the ESB to do the message format conversion. If the business use case needs to be extended further to include another service, you can integrate that service to the ESB, and rather than having point-to-point links, all three systems can be connected through the unified ESB integration layer.

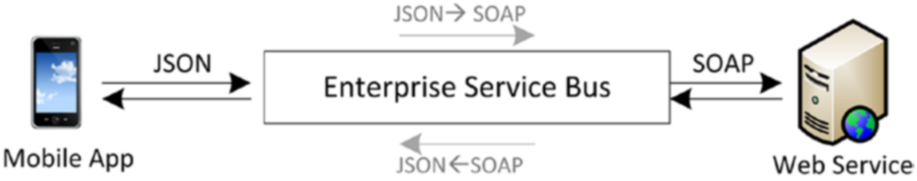

Figure 1-1. *An ESB can be configured to convert messages between the formats recognized by the mobile app (JSON) and the web service (SOAP) it wants to talk to. This is a simple example of enterprise integration.*

Now you have a clear understanding of a concrete enterprise integration use case. Let's explore the enterprise integration space further and see how the ESB is used as the integration backbone.

Modern enterprises need to integrate all the heterogeneous systems (systems using disparate protocols, message formats, and so on) to form various business solutions. The integration between on-premise systems such as web services, file repositories (FTP), proprietary systems such as Enterprise Resource Planning systems (for example, SAP), legacy systems, and data residing in databases and cloud-based solutions such as Software as a Service (SaaS), is the key responsibility of an ESB.

The absence of an integration platform leads the enterprise to require links from a given system to all other systems in the enterprise IT solution. This is known as *point-to-point integration* or *spaghetti integration*.

As depicted in Figure 1-2, the point-to-point integration approach has inherent complexity because the number of systems that participate in the integration scenario increases. If you have to modify or remove one of the systems then that affects the interaction between most of the other systems in your enterprise integration scenario. Therefore, the point-to-point integration approach is extremely difficult to maintain, troubleshoot, and scale.

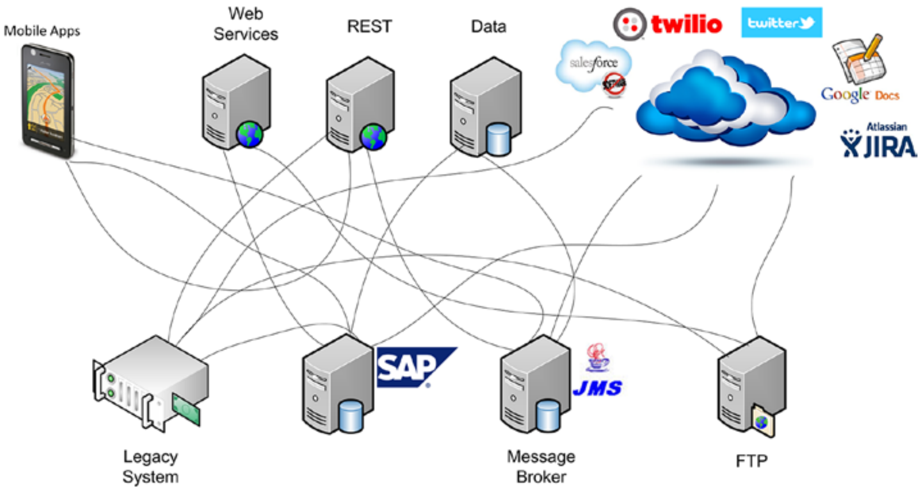

***Figure 1-2.** When an enterprise does not use ESB, each system must know how to talk directly to every other system it needs to interact with. This is known as point-to-point integration.*

As depicted in Figure 1-3, ESB can be used as the bus that all the other systems can connect to. An ESB-based approach connects disparate systems using the ESB messaging backbone, and it connects on-premise as well as cloud services. As illustrated in Figure 1-3, ESB eliminates point-to-point integration and integrates all the disparate systems using the bus architecture.

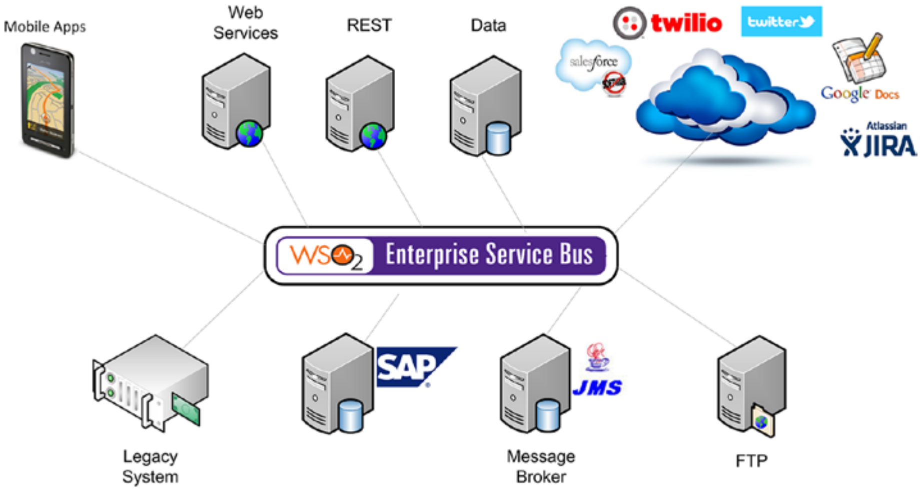

***Figure 1-3.** When an enterprise uses ESB, only the ESB needs to know how to talk to each application. The applications themselves do not need to be modified. This is far more efficient than point-to-point integration.*

Based on the previously described integration use case, you can come up with a generic description for ESB. *ESB is an architecture pattern that enables the disparate systems and services to interact through a common communication bus, using lightweight and standard messaging protocols.*

In the next section, you'll discover the core functionalities that are common to any ESB.

Core Functionalities of an ESB

In general, ESB has to offer a wide range of integration capabilities from simple message routing to integrated proprietary systems using complicated integration adaptors. These are the generic functionalities that are common to most ESB products:

- *Message mediation:* Manipulate the message content, direction, destination, and protocols with message flow configurations.
- *Service virtualization:* Wrap existing systems or services with new service interfaces.
- *Protocol conversion:* Bridge different protocols. For example, JMS to HTTP.
- *Support for Enterprise Integration Patterns (EIP):* EIP is the de facto standard for Enterprise Integration (`http://www.eaipatterns.com/`).
- *Quality of service:* Apply security, throttling, and caching.
- *Connecting to legacy and proprietary systems:* Business adapters, including SAP, FIX, and HL7.
- *Connectors to cloud services and APIs:* Salesforce, Twitter, PayPal, and many more.
- *Configuration driven:* Most functionalities are driven by configuration but not code.
- *Extensibility*: There are extension points that can be used to integrate with any custom protocol or proprietary system.

For the most part in the ESB architecture, the ESB is considered a lightweight, stateless integration bus. The architecture itself is mostly based on SOA, but that doesn't mean that you can't integrate non-SOA systems, such as proprietary systems, by using ESB.

The ESB landscape is vast, where there are numerous ESB solutions ranging from open source to proprietary integration solutions. In the following section, you'll discover the key differentiators of WSO2 ESB.

Why WSO2 ESB?

In the ESB vendor space, most of the vendors have rebranded the monolithic and heavyweight enterprise integration solutions as an ESB. But WSO2 ESB is designed and developed from the ground up as the highest performance, lowest footprint, and most interoperable integration middleware. While WSO2 ESB has to improve its graphical tooling support for designing message flows and graphical data mapping, it offers a broad range of integration capabilities and high-performance message routing support by using an enhanced and optimized message mediation engine, which was inspired by Apache Synapse. In this section, you'll discover key differentiators between WSO2 ESB and other ESB vendors.

Interoperability and EIP Support: Connecting Anything to Anything

WSO2 ESB offers a broad range of integration capabilities from simple message routing to smooth integration of complex proprietary systems. The de facto enterprise integration standards for Enterprise Integration Patterns (EIP) are fully supported in WSO2 ESB. It not only comes with 100% coverage of EIPs, but also with use cases and samples for implementing each and every EIP.

While supporting all the key ESB integration features discussed in the last section, WSO2 ESB offers various integration adapters to proprietary and legacy systems such as SAP. Also, it empowers the on-premise and cloud-based integration scenarios (hybrid integration) with numerous connectors that allow you to smoothly integrate to popular cloud services such as Salesforce, PayPal, and Twitter (see the Connector Store at `https://store.wso2.com/store/`).

WSO2 ESB offers all these integration capabilities that you can use by configuring the ESB without a single line of code, and in case of any custom requirement, such as supporting proprietary message formats, you can use the numerous extension points to plug in your custom code.

Performance and Stability: The Fastest Open Source ESB

The performance and latency of any ESB solution is a vital factor when it comes to handling large volumes of messages. Based on the regular performance comparisons done by WSO2 on the message routing performance of popular open source ESBs, WSO2 outperforms all the ESB vendors. (The latest ESB performance comparison is available at `http://wso2.com/library/articles/2014/02/esb-performance-round-7.5/`

The ESB performance comparison given in Figure 1-4 is based on the most recent performance test comparison against WSO2 ESB and other popular ESB vendors. For almost all the integration scenarios, WSO2 ESB outperforms all other ESB competitors.

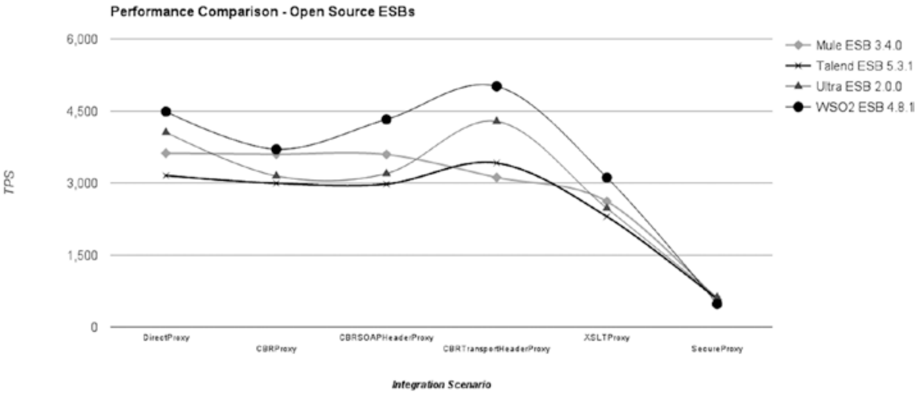

Figure 1-4. *ESB performances comparison between open source ESB vendors for commonly used message routing scenarios*

Stability is also another aspect that goes hand-in-hand with performance. Thousands of production deployments of WSO2 ESB show its stability and its maturity as an ESB solution. eBay uses WSO2 ESB for handling more than several billions of transactions per day in its live traffic (an eBay case study is available at `http://wso2.com/casestudies/ebay-uses-100-open-source-wso2-esb-to-process-more-than-1-billion-transactions-per-day/`).

The Platform Advantage: Part of the WSO2 Middleware Platform

WSO2 ESB is part of the comprehensive WSO2 middleware platform. When you're building real-world enterprise integration solutions, you require the integration capabilities offered by WSO2 ESB, as well as other middleware capabilities such as API management, identity management, data services, analytics, complex even processing, and so on, which are beyond the scope of an ESB. Few ESB vendors who aren't based on a platform concept have tried to have all these features in a monolithic ESB product, but have failed because such solutions cannot address the modern enterprise IT requirements.

A WSO2 middleware platform is built from the ground up with the holistic vision of facilitating all enterprise middleware requirements. The high-level objective of a WSO2 middleware platform is to enable a connected business.

- *All the WSO2 products are built from the ground up and on top of a common foundation:* WSO2 Carbon, a modular, reconfigurable, elastic, OSGi-based architecture, whereas most of the other middleware platforms are primarily built with acquisitions of heterogeneous middleware solutions.

- *Lean and optimized for maximum performance:* Every product in the WSO2 platform is lightweight and designed for achieving the highest performance. For instance, WSO2 ESB is the fastest open source ESB.

- *Largest middleware platform built on a single code base*: All the WSO2 products share the same code base built around a single kernel—WSO2 Carbon. Unlike middleware platforms built with the combination of heterogeneous middleware solutions, WSO2 offers frictionless cross-product integration.
- *100% free and open source under Apache License 2.0 with comprehensive commercial support:* WSO2 has no notion of commercial versus community editions. What you freely download from the `http://wso2.com` web site is the same version used for all the production deployments. The same architects and developers who contributed to the WSO2 platform drive the commercial support for the WSO2 products.
- *Cloud native:* Every WSO2 product inherently supports on-premise, cloud, or hybrid deployments.

At this point you've learned about the key differentiators of WSO2 ESB. In the next section, you'll learn the fundamental concept that's required to start integrating with WSO2 ESB.

How does WSO2 ESB Work?

In this section, you'll learn about the core functional components of WSO2 ESB and the complete end-to-end message flow. Let's design the same financial organization's integration scenario with WSO2 ESB and use that to understand the message flow of WSO2 ESB.

As illustrated in Figure 1-5, the main integration challenges that you have here are to integrate a backend service, which is a SOAP-based web service, with a mobile application that uses JSON. Therefore, the ESB primarily takes care of message format conversion (Message Translator EIP) and exposes a new JSON interface on behalf of the backend web service. The JSON request from the mobile app to the ESB is shown in Listing 1-1.

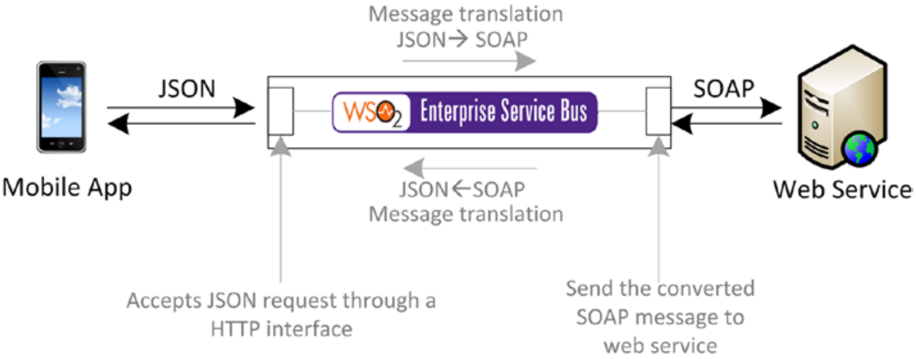
Message translation
JSON→ SOAP
JSON
SOAP
WSO2 Enterprise Service Bus
Mobile App
Web Service
JSON←SOAP
Message translation
Accepts JSON request through a HTTP interface
Send the converted SOAP message to web service

Figure 1-5. *Using WSO2 ESB to integrate a SOAP-based web service and a JSON-based mobile client*

Listing 1-1. JSON Request from Mobile App to ESB

```
{
  "getFinancialQuote": { "company": "WSO2" }
}
```

The backend web service accepts a SOAP message that's shown in Listing 1-2. Therefore, the message processing logic of the ESB needs to convert the request to the SOAP format and convert the SOAP response to back to JSON.

Listing 1-2. Request that Needs to be Sent to the Backend Web Service

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ser="http://services.samples">
  <soapenv:Header/>
  <soapenv:Body>
    <ser:getSimpleQuote>
      <ser:symbol>WSO2</ser:symbol>
    </ser:getSimpleQuote>
  </soapenv:Body>
</soapenv:Envelope>
```

The key steps related to the implementation of this integration scenario can be listed as follows:

- Build a virtual HTTP interface/service that can accept the JSON request from the mobile client and respond with the JSON response.
- In the request-handling path of the virtual HTTP interface/service, it should convert the incoming JSON message to a SOAP message and then send it to the SOAP-based web service backend.
- In the response path, the virtual HTTP interface/service needs to convert the incoming SOAP response to a JSON message and send it back to the client.

Now you'll discover how these steps can be implemented with WSO2 ESB by using its core functional components.

Functional Components

The high-level message flow of WSO2 ESB comprises three main logical components:

- *Message entry points:* Receive client requests.
- *Message processing units:* Contain the mediation logics to process client requests, request that ESB sends to the server, the server response processing logic, and the response that ESB sends back to the client.
- *Message exit points:* Integration points to backend services.

As depicted in Figure 1-6, the mobile clients send a request to the WSO2 ESB via one of its message entry points, then the ESB processes the request messages in the request message processing units, and the message is sent to the backend web service via message exit points. Once the response is received from the backend service, the message goes again through the response message processing units and finally sends the processed response to the client.

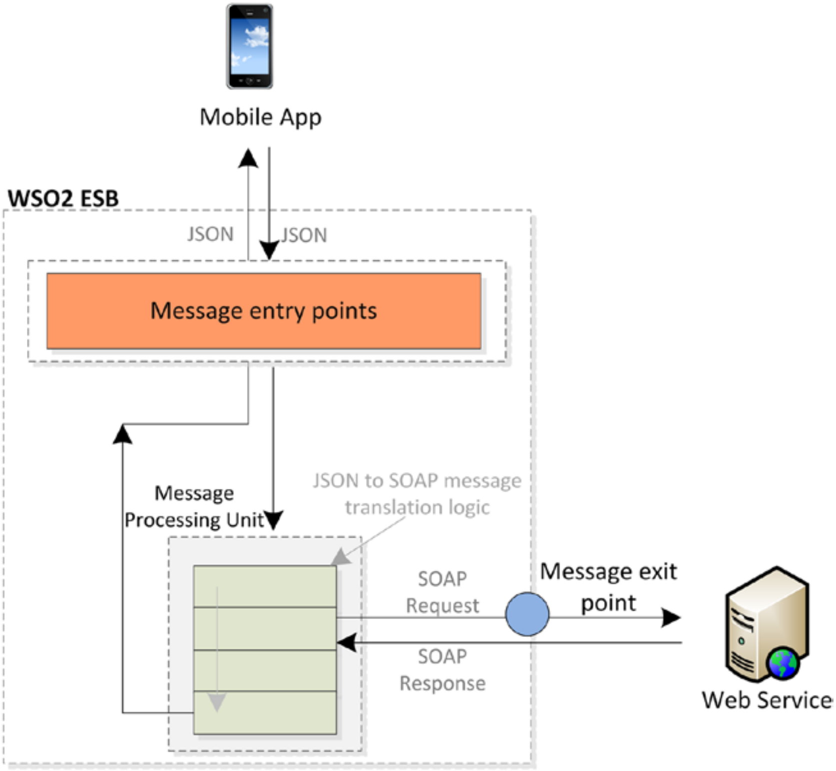

Figure 1-6. *High-level message flow of the integration between mobile app and backend web service*

The task of processing requests or responses in an ESB is known as *message mediation*. As shown in Figure 1-6, the message processing units are doing the required request and response message mediation work.

Message Entry Points: Proxy Service, APIs, and Inbound Endpoints

WSO2 ESB has three main message entry points:

- *Proxy service:* This is a web service interface exposed from the ESB.
- *REST APIs/HTTP:* An HTTP interface anchored at a specific URL context.
- *Inbound endpoints:* A message source with listening or polling capability.

The message entry points are the main components responsible for handling the message transferring from external systems to the ESB. The messages that come through any of these entry points are routed to the message-processing unit, which is responsible for the processing of the message context and message attributes. In the previous example, the messages sent from the mobile app via the HTTP protocol hit the message entry point API and route the message to the respective processing unit (sequence).

Message Processing Unit: Sequences and Mediators

The processing of the message takes place in components known as *sequences*. A given sequence can contain a sequence of components that can process a given message. These components are known as *mediators*. In our example, the logic to translate the message from JSON to SOAP and send out the message takes place in the message-processing unit.

Message Exit Points: Outbound Endpoints

The outbound endpoint (or endpoint) is the message exit point in the WSO2 ESB, which logically represents an external backend service endpoint. In our example, the service address of the backend web service is configured as an outbound endpoint and the message is routed to the outbound endpoint via a *call* or *send* mediator.

Figure 1-7 shows how the message flow is configured to implement the financial organization's integration scenario. The key points related to understanding the message flow in Figure 1-7 are as follows:

- *Expose an HTTP interface to the mobile client:* You should implement an HTTP interface that has to be exposed to the mobile client. That's where we need to configure the message entry points in the WSO2 ESB. Because we have to integrate a mobile client, we can go for the API/HTTP Service message entry point and configure that in the ESB.
- *Anchor an API/HTTP service:* An API/HTTP service is anchored at a URL context, which is designed by the ESB developer and is capable of receiving any request coming on HTTP protocol for that particular context.

- *Configure a mediation sequence:* When you create the API, you need to configure the sequence that will be used to process the request. The sequence contains a set of mediators to process the request.

- The first mediator is a payload factory mediator, which is used to convert the incoming JSON message format to the arbitrary SOAP message format and extract whatever values are needed from the original JSON request.

- You need to set a SOAP action as a header prior to sending the message out from the ESB, because it's required to send an action along with a SOAP 1.1 message. Therefore, a header mediator is used to set the SOAP action.

- Then you need a "call" mediator that can send the message out from the ESB.

- *Configure an outbound endpoint:* When you send out the message, you can configure the destination address of the message or the URL of the backend service using endpoint or outbound endpoint. In addition to specifying the address, you can add various conversion formats when configuring an endpoint, such as SOAP 1.1 and POX (Plain Old XML). Since you want to convert the message to SOAP format, you can use soap11 as the "format" attribute of the endpoint.

- *Configure the response mediation sequence:* Because you need to send a JSON response back to the client, the Property mediator is used as a flag to change the response message format to the JSON message format.

- *Set up a fault mediation sequence:* Any failure scenario can be handled using the fault sequence.

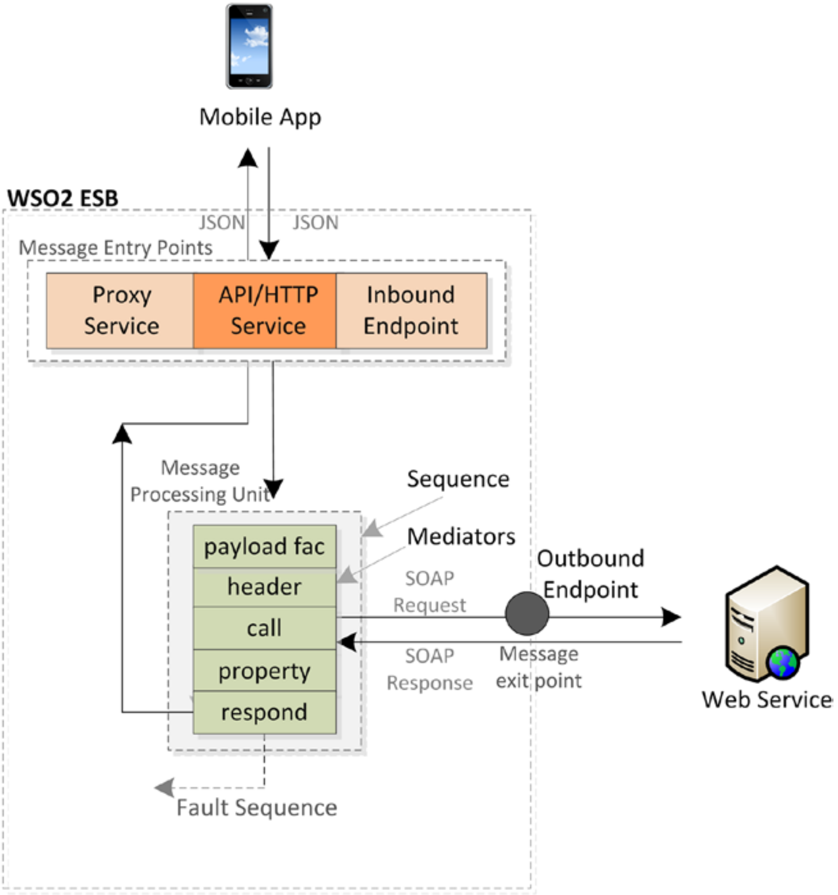

***Figure 1-7.** WSO2 ESB message flow for integrating a SOAP-based backend service with a JSON based mobile client*

This completes the end-to-end message flow of our integration scenario. Also, you have an alternative approach to implement the same scenario with a two-sequence model.

As depicted in Figure 1-8, the API/HTTP service can use its built-in, in-sequence, and out-sequence to implement the same integration scenario that was illustrated in Figure 1-8. The only difference here is that the request message always goes to the in-sequence. Hence, the request message processing takes place there. Unlike in earlier approaches, a *send* mediator is used instead of *call* mediator. The main difference is that once you use the send mediator at a given sequence to send out the request, the message flow stops at that point. When you get the response from the backend service, the response flow starts from the out-sequence. Therefore, the out-sequence is the place that you can do the response processing.

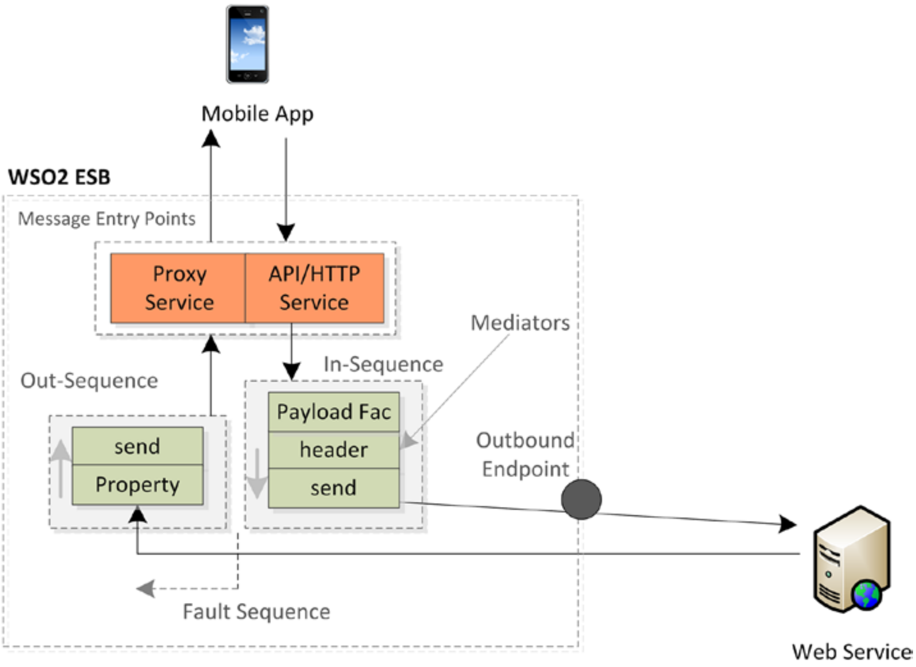

Figure 1-8. *WSO2 ESB message flow for integrating a SOAP-based backend service with a JSON based mobile client using "In" and "Out" sequences*

You can use both of the previously described approaches, but in cases when you're implementing complex service orchestration/chaining scenarios, the *call* mediator-based approach is less complicated.

WSO2 ESB Configuration Language

The entire message flow that you learned during the financial organization's use case is implemented using the XML-based configuration language of WSO2 ESB. Because the WSO2 ESB integration scenario development procedure is completely configuration driven, all the message entry points, message processing units, and message exit points are configured using an XML-based configuration language.

Graphical message-flow editors are available for WSO2 ESB. They're built on top of its configuration language, but throughout this book you'll find all the samples and use cases implemented in raw configuration language.

The following sample configuration represents the implementation of the financial organization's integration scenario that we discussed previously. Here I used an API/HTTP service as the message entry point to the WSO2 ESB and used the single sequence approach with *call* and *respond* mediators. With the following configuration, you can create a new API named `ShoppingInfo` that can be accessed through the HTTP protocol. The complete steps for implementing, deploying, and testing the integration scenarios are covered in detail in Chapter 2.

Listing 1-3. JSON Request from Mobile App to ESB

```
<api xmlns="http://ws.apache.org/ns/synapse"
name="ShoppingInfo"                           <!-- [1] -->
        context="/ShoppingInfo">
   <resource methods="POST">
      <inSequence>                              <!-- [2] -->
         <payloadFactory media-type="xml">
            <format>
               <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/
               soap/envelope/" xmlns:ser="http://services.samples">
                  <soapenv:Header></soapenv:Header>
                  <soapenv:Body>
                     <ser:getSimpleQuote>
                        <ser:symbol>$1</ser:symbol>
                     </ser:getSimpleQuote>
                  </soapenv:Body>
               </soapenv:Envelope>
            </format>
            <args>
               <arg evaluator="json" expression="$.getFinancialQuote.
               company"></arg>
            </args>
         </payloadFactory>
         <header name="Action" value="urn:getSimpleQuote"></header>
         <call>
            <endpoint>                                  <!-- [3] -->
               <address uri="http://localhost:9000/services/
               SimpleStockQuoteService" format="soap11">
</address>
            </endpoint>
         </call>
         <property name="messageType" value="application/json"
         <!-- [4] -->scope="axis2" type="STRING">
 </property>
         <respond/>            <!-- [5] -->

      </inSequence>
   </resource>
</api>
```

1. Anchoring an HTTP service/API on the context / ShoppingInfo as a message entry point.

2. Request message processing (JSON to SOAP conversion and sending messages out).

3. Configuring backend web service as the message exit point.

4. Response message processing; change the response format to JSON.

5. Sending the response back to the client.

As shown in Listing 1-3, a top-level configuration element known as `api` is the message entry point configuration, followed by the `inSequence`, which is the message processing component. The `inSequence` has the mediators, which are the message processing units. To send out the message, you have a *call* mediator with an *endpoint*. The response processing happens in the mediators that are placed after the *call* mediator (property and respond).

You've discovered about all the building blocks required to design a real-world enterprise integration scenario with WSO2 ESB. You'll learn more about how you can implement and try out the same scenario in WSO2 ESB in Chapter 2.

How to Try the Use Cases in this Book

You can try most of the sample use cases that are discussed throughout this book.

- All the source code related to these use cases can be found at `https://github.com/kasun04/maestro/`.
- The configurations for all the use cases of a given chapter are located at: `https://github.com/kasun04/maestro/tree/master/src/main/resources`.
- All the instructions to run each example are specified under the README file of each chapter. For example if you want to run samples from Chapter 2, the instructions can be found at `https://github.com/kasun04/maestro/blob/master/src/main/resources/ch_02/uc_01/README.txt`

Also, note that the latest configuration is always kept and updated at the GitHub. So, in rare cases, there can be slight mismatches between the code snippet that you find in the book and the sample configuration you'll find in the GitHub repository.

Summary

In this chapter, you learned the fundamentals that are required to build integration solutions with WSO2 ESB. Let's summarize them as follows:

- An Enterprise Service Bus (ESB) enables diverse applications, services, and systems to talk to each other through a common communication bus, using lightweight and standard messaging protocols such as SOAP and JSON. It acts as the main messaging backbone in any Service Oriented Architecture (SOA).

- WSO2 ESB was built on top of a common foundation, called WSO2 Carbon. It's a modular, reconfigurable, elastic, OSGi-based architecture on which all WSO2 products are based. It's the fastest ESB implementation currently available, 100% free, open source under the Apache License 2.0, and supports on-premise, cloud, and hybrid deployments.
- An ESB has three main logical components. Message entry points receive client requests, message processing units contain the logic to process client requests and server responses, and message exit points provide a way to send client requests onto the server and receive their response.
- To configure a message flow through WSO2 ESB between a client and a server, you should configure a message entry point for the client to talk to the ESB, configure a mediation sequence that can process the client request into a message the server understands and sends it out to the server, configure the outbound endpoint for the message to the server, configure another mediation sequence to process and send on the server's response for the client, and set up a fault sequence to deal with failures.

CHAPTER 2

Getting Started with WSO2 ESB

In this chapter, you'll get started with building integration scenarios with WSO2 ESB. The first example provides the foundation for the rest of the chapters, by ensuring your ESB server is set up correctly. We'll also start with the most basic example of system integration by implementing the message transformation use case discussed in Chapter 1.

Designing a Simple Integration Scenario with WSO2 ESB

The best way to get started with WSO2 ESB is to build a simple but real-world integration scenario. The main objective of this use case is to transform messages between two common data formats: JSON and XML.

For our use case, assume that a financial company with the domain example.com exists, that hosts SOAP-based web services to expose its business functionalities as services. The StockQuoteService financial service is one of the key business functionalities offered from example.com, which gives you the stock quote details for a given organization. But the example.com financial organization wants to enable this business functionality to its mobile users (who use a JSON as the message format) without modifying the existing backend service and the mobile client.

Suppose that the JSON request format is as follows and the response accepted by the mobile client is the one-to-one transformation of the SOAP response from a backend service to JSON.

```
{
  "getFinancialQuote": { "company": "WSO2" }
}
```

K. Indrasiri, *Beginning WSO2 ESB*, DOI 10.1007/978-1-4842-2343-7_2

The key design steps of the *integration scenario* illustrated in Figure 2-1 can be identified as follows:

- Creating an HTTP interface at the ESB layer on behalf of the existing StockQuote web service.
- Transforming the incoming JSON request to the appropriate SOAP request that needs to be sent to the `SimpleStockQuote` service of the `example.org` financial service and then invoking the service.
- Handling the SOAP response message from the `SimpleStockQuote` service and transforming it back to JSON before sending back the request.

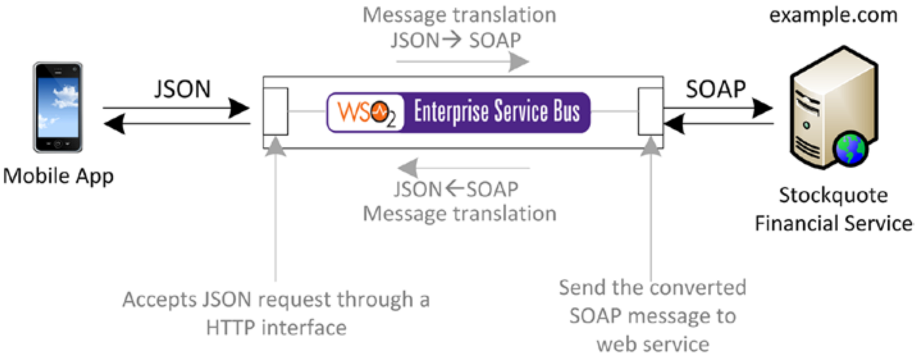

***Figure 2-1.** Integrating SOAP-based SimpleStockQuote financial service and a mobile client with WSO2 ESB*

Now let's proceed to the realization of the previous design steps using WSO2 ESB.

Building the Integration Scenario

Since now you're familiar with the sample integration scenario that we are planning to build with WSO2 ESB, in this section I'll walk you through all the steps that are required to build that integration scenario using WSO2 ESB. For our example to be minimally viable, however, we need to do preliminary work to set up the `StockQuote` service.

As illustrated in Figure 2-2, the implementation of this integration scenario requires you to configure a message entry point in WSO2 ESB, configure message-processing components, and finally configure a response-sending logic. The key design steps discussed in the previous section can be mapped into the main implementation steps that you can follow in WSO2 ESB.

1. Creating an HTTP service/API in WSO2 ESB.
2. Creating the request that needs to be sent to the backend service.