THEORY IN PRACTICE

Enterprise Service Bus

DAVID A. CHAPPELL

THEORY IN PRACTICE

Enterprise Service Bus

The Enterprise Service Bus (ESB) is a technology that is being readily adopted within IT organizations across a variety of industries. An ESB provides a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing in an event-driven Service Oriented Architecture (SOA). ESBs are being used to solve real-world integration challenges in many unique ways, as we will explore throughout this book.

Enterprise Service Bus contains practical strategies for understanding the architecture of an ESB while integrating diverse applications into enterprise-wide solutions. Throughout the book, there are integration patterns that show how an ESB can be applied toward solving today's application integration challenges. These strategies and patterns show how to integrate enterprise applications using standard components and interfaces.

This book will provide you with a conceptual and architectural overview of the ESB from a seasoned expert in the areas of standards for enterprise messaging, web services, and SOA. Dave Chappell offers his unique insights gained from working with the pioneers and innovators of the ESB. This book will serve as a guide to define the ESB technology, as viewed through the eyes of an experienced integration architect who has worked with IT professionals on real-world ESB-based integration solutions.

If you build, integrate, or architect enterprise applications, you need this book! With the knowledge you gain in these pages, you will possess the skills and expertise to become a next-generation enterprise integration architect.

DAVID A. CHAPPELL is vice president and Chief Technology Evangelist at Sonic Software (pioneers of the ESB). Dave has more than 20 years of experience in the software industry in a broad range of roles including architect, codeslinger, sales, support, and marketing. Dave has a strong background in a number of distributed computing models, and is well-known for his writings and public lectures on the subjects of the ESB, Message Oriented Middleware (MOM), enterprise integration, and evolving standards and web services. Dave led the development effort for SonicMQ, which has grown to become synonymous with enterprise messaging and the Java Message Service (JMS). He has published numerous articles on interoperability and integration technology in leading industry publications, such as *Java Developers Journal*, *JavaPro*, *Web Services Journal*, *XML Journal*, and *Network World*. He is noted for authoring popular books on JMS, web services, and ebXML.

www.oreilly.com

US $39.95 **CAN $57.95**

ISBN: 978-0-596-00675-4

O'REILLY®

Enterprise Service Bus

Enterprise Service Bus

David A. Chappell

O'REILLY®
BEIJING • CAMBRIDGE • FARNHAM • KÖLN • SEBASTOPOL • TAIPEI • TOKYO

Enterprise Service Bus
by David A. Chappell

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

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

Editor:	Mike Hendrickson
Production Editor:	Emily Quill
Cover Designer:	Edie Freedman
Interior Designer:	David Futato

Printing History:

June 2004: First Edition.

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

ISBN13: 978-0-596-00675-4
[M] [3/10]

Contents

Foreword

Integration is making a comeback—perhaps it never even left. In this book you'll be introduced to the next generation of integration, called Enterprise Service Bus (ESB). ESB is really exciting in that it introduces battle-ready integration principles in a new way using open standards, messaging, and loosely coupled Service Oriented Architecture (SOA) principles.

The costs of using proprietary integration solutions will soon become something of the past. Integration solutions will always be required, but companies can look forward with enthusiasm knowing that upcoming solutions will be based on open standards and common integration principles, especially in the area of web services and SOA. Soon, the new game in town will be integration products competing on who best supports systemic requirements (scalability, availability, performance, etc.), and not on specific product features. Not only that, the new desire to push toward SOA forces organizations to rethink their existing environment and create architectures that are based on coarse-grained, loosely coupled, shared services. However, we all know that performing the magic of "gluing" these services together is no small task. It requires new thinking in both business and technology solutions. And, in the past, because there were virtually no integration standards and few agreed-upon repeatable integration patterns, proprietary integration products were really the only option.

Now that's all about to change, and that is what this book is about. What I like about this book is that Dave shows us how ESB brings integration solutions to those of us who want to focus on integration architectures and solutions. The ESB concept, as the backplane of a highly distributed integration network, allows us to think about the architecture and the best way to design and architect our solutions using an event-driven SOA, without having to deal with specialized integration approaches and becoming middleware surgeons. It allows us to focus on how we want to architect our solutions without conforming our requirements to what a product offers.

There are two areas in this book that particularly excite me. First, as an architect of enterprise Java™ solutions I am excited about the synergy between the ESB and Java Business Integration (JBI/JSR-208). JBI combined with an ESB is a godsend to those

who have felt locked into proprietary integration products and solutions. JBI increases the proliferation of integration technology by providing a standards-based container environment in which integration processing elements run as services. These processing elements may include BPEL engines, XSLT transformers, routing engines, dispatchers, and any other integration feature engine you can think of. An ESB can provide its own JBI container environment or can integrate with one provided by another vendor. What's very cool about this is that it enables an ecosystem where ESB vendors such as Sonic can now focus on coordination, transport, and routing of a highly distributed and consistently managed SOA backplane, while at the same time providing an environment in which JBI processing engines can flourish.

Second, as a design-patterns person I am also excited by this book's use of ESB components in pattern-based approaches to integration, which are used to explain the capabilities of an ESB. Even more interesting is that Dave chose to leverage and extend the "Gregor-grams" from the *Enterprise Integration Patterns* book by Gregor Hohpe and Bobby Woolf. In the EIP book, there is a wonderful use of visuals to depict various patterns for enterprise messaging. The ESB patterns used throughout this book show the visual construction of loosely coupled, service-based integration patterns to create larger-grained solutions, or micro-architectures, which leverage the ESB architecture to uniquely solve complex integration problems in simple ways. This concept alone is reason enough to buy this book. The visual metaphor lends itself wonderfully to composing integration solutions and really helps the integration architect represent the architecture visually and form a complete ESB solution. The nice thing about any loosely coupled, messaging-based solution is that you tend to add new feature elements as requirements dictate. Using patterns to compose the ESB features allows you to not only add the integration features as needed, but also to see the visual architecture as it evolves.

I really think SOA built upon ESB is the next wave in integration. Read this book and decide for yourself. It is sure to open up new ways of thinking about solving any and all of your integration challenges.

—John Crupi
Sun Distinguished Engineer
Coauthor, *Core J2EE Patterns*
Bethesda, MD
April 2004

Preface

Welcome to *Enterprise Service Bus*. I hope you enjoy reading this book as much as I enjoyed writing it. Being part of a new technology evolution is very exciting, and now that you are beginning to explore the concept of the Enterprise Service Bus (ESB), you can become a part of it, too. The ESB provides a highly distributed, event-driven Service Oriented Architecture (SOA) that combines Message Oriented Middleware (MOM), web services, intelligent routing based on content, and XML data transformation. ESBs are being used to solve integration challenges in many unique ways, as we will explore throughout this book.

About This Book

Over the past few years, I (along with many of my coworkers) have had the invaluable opportunity of working closely with IT professionals who were trying to build or buy something like an ESB. As part of the research for this book, I met, spoke with, and interviewed many IT leaders across a variety of industries to gain an understanding of their integration challenges, and of how they are applying ESB integration concepts to uniquely solve them. The concepts in this book draw from that experience.

This book represents a new direction for O'Reilly books. It is the first of a new "Enterprise Series" of books from O'Reilly. This book is targeted to integration architects, project managers, CIOs, CTOs, and basically anyone who has a need to integrate applications and who understands the bigger picture of the integration issues within their IT organization.

At the same time, this book will also be valuable to everyday IT developers who need to understand ESB technology for their individual integration projects. As we will see in the first few chapters, the new "integration architects" are actually everyday IT developers, and the ESB concept brings the power and capability to solve their integration needs into their reach.

This book provides an architectural overview of the ESB concept, and includes discussions of many practical uses and integration patterns that explain how an ESB can be applied toward today's application integration challenges. Ideally, readers of this book should possess some knowledge of technical concepts, but this is not a hard requirement. The introductory chapters provide a conceptual overview, including the requirements, drivers, and catalysts that have fostered the emergence of this new technology category.

My intent in this book is to explain the ESB concept with just the right amount of detail that is suitable for individuals across many disciplines within an IT organization and with varying levels of technical and business acumen. This book takes a different tack from my previous O'Reilly books, and from most O'Reilly books in general, in that there are no code samples to speak of. You will, however, find a considerable number of diagrams that explain architectural concepts of the ESB.

Although this book represents a slightly new direction for O'Reilly, it also possesses a trait that O'Reilly books have become known for: the "Missing Manual" approach to a particular technology. The ESB is being rapidly adopted across a variety of industries. Many middleware, integration, and infrastructure vendors have also gotten "on the Bus" and are either shipping an ESB already, or have plans to build one. A big part of the ESB is its commitment to standards-based integration. An ESB is designed and built from the ground up with standard components and standard interfaces. However, there is no specification for the ESB itself. This book will serve as a guide to define what an ESB is, as viewed through the eyes of those of us who pioneered the ESB concept and have been working with IT professionals on real-world ESB-based integration solutions. Even other vendors who want to join in with the growing ESB market trend, and are trying to figure out exactly what an ESB is, are welcome to use this book as a guide.

Overview of the Chapters

Chapter 1, *Introduction to the Enterprise Service Bus*, gives an overview of the ESB, including the many characteristics that define it. It also explains the evidence of industry adoption and market attraction of this new concept in service-oriented integration.

Chapter 2, *The State of Integration*, provides a summary of the drivers, both business and technical, that have contributed to the need for a new approach to integration. It also examines some surprising statistics showing that, in general, the enterprise is far from connected, and it explores the shortfalls of Enterprise Application Integration (EAI) approaches to date. It explains the characteristics of an "accidental architecture" which helps you to identify your own architecture issues. Lastly, it explains how the ESB concept draws from best practices of previous integration approaches, and shows that you can refactor toward an ESB and away from an accidental architecture in incremental steps.

Chapter 3, *Necessity Is the Mother of Invention*, examines some key concepts of the ESB, including the many requirements, technology drivers, and forces in the IT climate that led to the creation of the ESB concept. This is explained in the context of the recent history of the evolution of the ESB, illustrating the point that an ESB is not merely an academic exercise; it is born out of necessity, based on real requirements and difficult integration problems that couldn't be solved by any preexisting integration technology. The discussion concludes with a high-level study of an ESB deployment, in which a manufacturer exposes inventory management and supply chain optimization functionality to its remote distributors as shared services through an ESB. This study, along with many others, will be revisited in more technical detail in later chapters.

Chapter 4, *XML: The Foundation for Business Data Integration*, explores the use of XML as a means for providing the mediation between diverse data structures and formats as data is passed between applications. It also examines how ESB-enabled data transformation and content-based routing services can support an XML data exchange architecture that insulates individual applications from changes in data structures, as integrated business processes improve over time.

Chapter 5, *Message Oriented Middleware (MOM)*, explains Message Oriented Middleware and its role in an enterprise integration strategy. A MOM is a key part of the ESB architecture, as it provides the underpinnings to the network of virtual channels that an ESB uses to route messages throughout an extended enterprise and beyond. This chapter explains how the definition of a MOM has evolved to support the highly diverse and distributed topologies and open standards that an ESB integration requires. (In other words, it's not your Mom's MOM.) The chapter goes on to explain the key differences between the low-level coding of using a MOM versus the higher-level configuration aspects of using an ESB.

Chapter 6, *Service Containers and Abstract Endpoints*, explains the details of what makes an ESB an ESB. The ESB provides an architecture that brings together all the concepts described in previous chapters into an infrastructure that can span an extended enterprise and beyond. This chapter describes the ESB *service container*—a key enabler of the highly distributed, event-driven SOA—and its concept of endpoint abstraction.

Chapter 7, *ESB Service Invocations, Routing, and SOA*, covers the service invocation model—i.e., the underlying framework that provides the SOA in an ESB. We will discuss multiple forms of process routing, including the concept of itinerary-based routing and the role it plays in enabling a highly distributed SOA across independently deployed services.

We will also discuss some fundamental ESB services such as content-based routing and XSLT transformation, and explore how service types can be defined and then reused for different purposes through elements of configuration. Finally, we will see how additional capabilities can be layered on top of an ESB through an advanced service, such as an XML persistence service and an orchestration service.

Chapter 8, *Protocols, Messaging, Custom Adapters, and Services*, examines how the ESB can extend its MOM core to create the flexibility in protocols necessary to connect to applications in an adaptable and non-intrusive way. This includes an explanation of how XML and SOAP messaging can be integral parts of an ESB strategy, yet also be flexible enough to carry other data formats such as EDI X12 messages. This chapter also shows the details of a partner integration using an ESB, and examines the use of third-party adapters and custom integration services for integrating with SAP.

Chapter 9, *Batch Transfer Latency*, explores the most common form of integration being done today: bulk data transfer and batch updating using Extract, Transform, and Load (ETL) techniques. We will discuss the business impact of the latency and reliability issues associated with this integration method, and examine the prescriptive steps for migrating toward real-time integration using an ESB in the context of a case study using pattern sketches.

Chapter 10, *Java Components in an ESB*, examines the various Java components in an ESB. The ESB is a platform-neutral concept, and could be implemented without any Java involved. However, a good ESB should take advantage of Java components due to the large adoption of Java-based technology across IT departments. A fairly extensive list of specifications that are utilized within an ESB comes out of the Java Community Process (JCP) on a regular basis; this could be the subject of a whole other book. Some Java specifications are particularly worth calling attention to because they have a special impact on the operation of an ESB. Chapter 10 will discuss the following specifications and their impact on making an ESB a more effective integration environment:

- Java Business Integration (JBI): A specification describing the way integration components, such as ESB services, can be plugged together in a vendor-neutral and portable fashion.
- J2EE Connector Architecture (JCA): A specification that defines how to use a standard set of interface contracts for creating adapters to connect into, and interact with, enterprise applications.
- Java Management eXtensions (JMX): A specification for remote management defining the means by which an application can interface with a management infrastructure and management consoles.

Chapter 11, *ESB Integration Patterns and Recurring Design Solutions*, examines some common uses of an ESB in integration scenarios. The ESB concept already has a number of common uses that solve some very common and challenging integration problems. These include the Validate, Enrich, Transform, Operate (VETO) pattern and its derivatives, a "two-step XRef" pattern, and several variations of backend integration patterns that are driven by enterprise portal requirements, such as a federated query/response pattern and forward caching using an ESB caching service.

Chapter 12, *ESB and the Evolution of Web Services*, provides an overview of the evolving web services specifications (including the WS-* family of specifications), and explains how web services and ESBs will move forward together.

The Appendix contains the list of vendors who are currently offering or are promising to offer an ESB product in the future. I will try my best to keep this up to date with subsequent printings of the book.

The Bibliography contains references to other books, specifications, and web links that were used in supporting research for this book.

Notational Conventions for ESB Integration Patterns

Throughout this book, integration patterns are often utilized to describe how concepts are put to use in an ESB. Some up-front explanations of the term "pattern" and this book's use of pattern notation are warranted.

A *pattern* is a popular concept in software construction. The concept of using a pattern to build things actually originates from other engineering disciplines that predate computers and software, and was originally made popular by Christopher Alexander, author of *A Pattern Language*.

When building a skyscraper, for example, there are certain repeatable methods to connect the steel girders. These methods can then be used in a larger architecture to form a building. Patterns for creating software architectures are similar, and have been explored in books such as *Design Patterns* by Erich Gamma, et al., and *Patterns in Enterprise Application Architecture* by Martin Fowler. More recently, the idea of messaging patterns has been popularized by Gregor Hohpe, et al., in *Enterprise Integration Patterns*, a great book that describes a series of integration patterns that are built upon low-level, message-oriented middleware.

When building a service-oriented integration network using an ESB, there are certain ways to combine individual integration components—for example, by using a data transformation service or content-based routing service to form repeatable and reusable patterns that can be built upon throughout the integration substrate. Chapter 11 conveys some examples of common integration patterns being used in ESB deployments today, and other chapters also explore ESB integration patterns in the context of the concepts being discussed. Although this book by no means covers all the patterns in use today, it should give you an idea of how an ESB can combine integration components to solve a variety of everyday integration problems. Refer to the links on this book's O'Reilly web page, *http://www.oreilly.com/catalog/esb*, for an ongoing list of ESB patterns, and for errata on the existing ones.

A pattern description must follow certain conventions so that it is easily identifiable, easily understood, and easily communicated among engineers when discussing the

larger issues that patterns can help to solve. This book uses the following form when describing patterns. (Some of this is borrowed from Alexander's, Hohpe's, and Fowler's books.)

Pattern Name
: The name of the pattern. This makes it easily identifiable and referenceable in other patterns, and helps to provide a vocabulary for engineers to use when discussing architectures that make use of patterns.

Problem Statement
: A summary of the problem that the pattern solves.

Forces
: Circumstances surrounding the problem that contribute to the need for the solution the pattern offers.

Sketch
: A drawing or diagram that illustrates the pattern in a simple, understandable form.

Solution
: A description of how the pattern solves the proposed problem.

Alternative Approaches
: Other approaches, which are often less than adequate, that could have been used to solve the problem.

Variations
: Alternate ways to apply or augment the pattern. Variations can also manifest themselves as choices that can be made in the makeup of the pattern, such as publish-and-subscribe versus point-to-point messaging models. Variations can often result in a whole new pattern with a unique name.

You won't necessarily find these as headings when reading through the patterns, but each pattern has these elements whenever possible.

Diagramming Notations

To describe the integration patterns used in this book, a simple set of glyphs were created to visually represent the different components of an ESB. These glyphs borrow from the work done by Gregor Hohpe, et al., in *Enterprise Integration Patterns*. These "Gregorgrams," as they have come to be known, are gaining rapid adoption in the industry as a means for describing messaging components and common integration patterns.

This book uses some of the glyphs from *Enterprise Integration Patterns*, and also supplies a whole new set of glyphs that represent many of the higher-level concepts and constructs used by the ESB. There is a parallel between the icons in *Enterprise Integration Patterns* and the concepts they represent, and the icons in this book and the ESB concepts that they represent. Many of the integration patterns described in *Enterprise*

Integration Patterns—such as content-based routers, load-balanced queue receivers, or routing slips—are implemented as custom messaging client code built on top of a messaging infrastructure. An ESB provides these same capabilities as built-in functionality, either as part of the bus directly or as an out-of-the-box service to be plugged into the bus. Therefore, the glyphs used in this book represent those pluggable services and are designed to be plugged together to form ESB diagrams. The diagrams can represent abstract pattern sketches or physical deployment characteristics, or can be a combination of both.

A legend showing all the glyphs that were created for use by this book is shown in the reference card which can be downloaded on the title's catalog page: http://oreilly.com/catalog/9780596006754. The glyphs in this book were designed with these goals in mind:

- To create a visual diagramming notation for explaining the core concepts of an ESB
- To allow you, the integration architect, to have a common means of designing your own integration patterns using an ESB
- To have a visual pattern language for documenting integration designs for legacy purposes

With these primary goals in mind, the icons also have these secondary uses:

- To show physical deployment characteristics of an ESB architecture
- To show message flow and business process routing for messages as they travel across an ESB
- To abstractly illustrate a pattern sketch for an integration environment, which describes the steps that a business message goes through as it travels across the various components of an automated business process

I plan to keep an updated set of these glyphs at *http://www.oreilly.com/catalog/esb*. They will be packaged as templates that can be plugged into Visio and PowerPoint so you can use them to diagram your own ESB architectures and integration patterns.

Figure P-1 shows examples of using the glyphs to show the details of individual ESB components. This figure will be explained in more detail in Chapter 6.

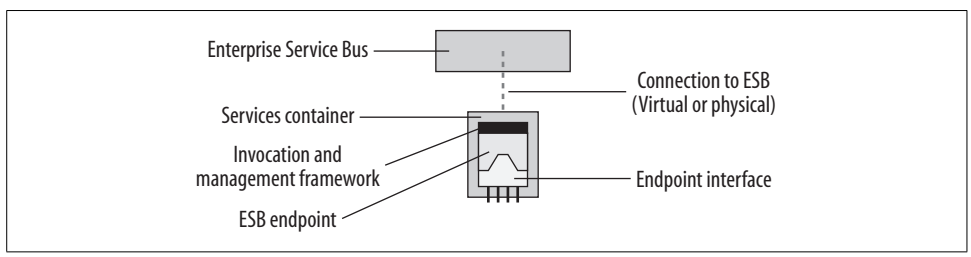

Figure P-1. Generic ESB endpoint

These glyphs can be combined to show composite components, as illustrated in Figure P-2.

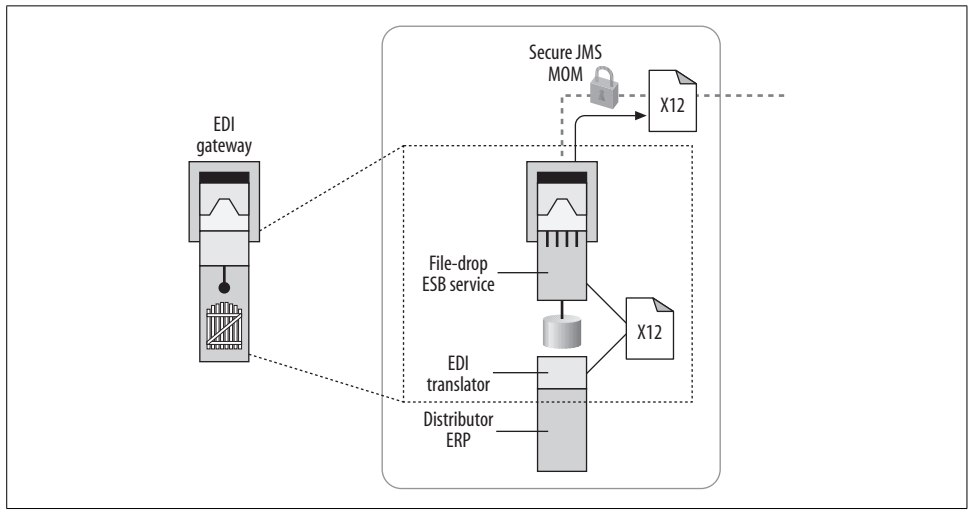

Figure P-2. Diagramming notation used to show the subcomponents of a composite ESB service

The diagramming notation can also be used to abstractly describe the components that make up an integration pattern, in the form of a pattern sketch. The particular pattern sketch convention was made popular by *Enterprise Integration Patterns*, and is akin to the way a rebus puzzle is formed. See Figure P-3.

Figure P-3. Rebus puzzle

When the rebus puzzle concept is applied to an integration pattern sketch, it can show the components that make up the pattern, and the steps that a message goes through as it follows the pattern (i.e., passes through the steps of a business process). This is illustrated in Figure P-4.

These glyphs can also be used together to show illustrations of physical deployment characteristics. For example, Figure P-5 shows different types of applications being plugged into different *ESB segments*, which may be different departments or business units, or separated by geographic location.

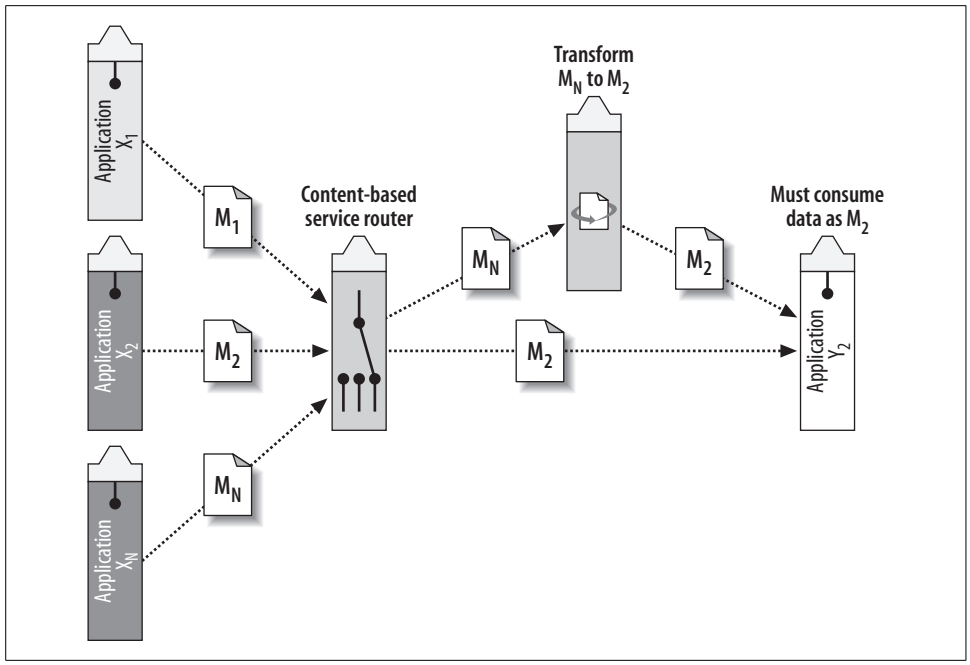

Figure P-4. Abstract pattern sketch showing the components that make up an integration pattern and the stages of processing for a message

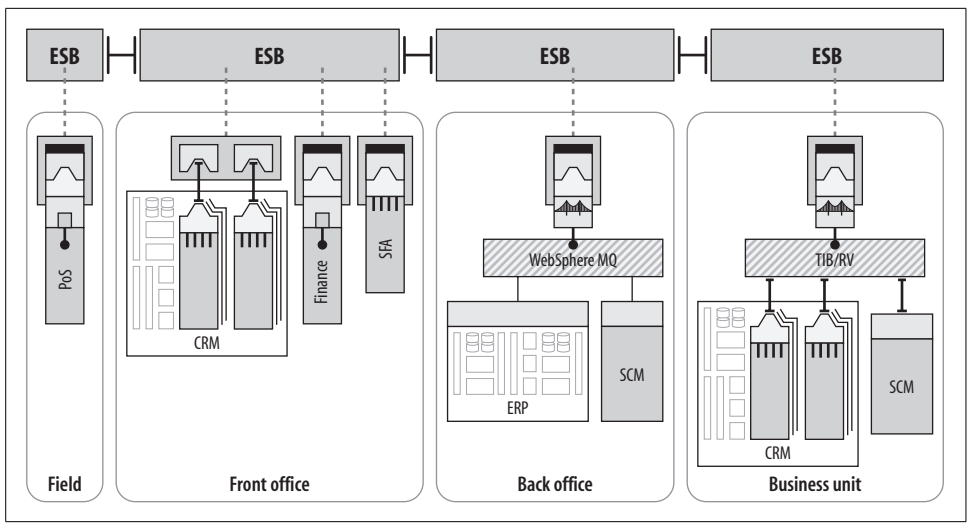

Figure P-5. An ESB diagram showing physical deployment characteristics of geographic separation, and different application endpoint types

The notation can also be used to represent flow of data across an ESB, either as messaging distribution or as multiple steps in an ESB process flow, as illustrated in Figure P-6 and Figure P-7, respectively.

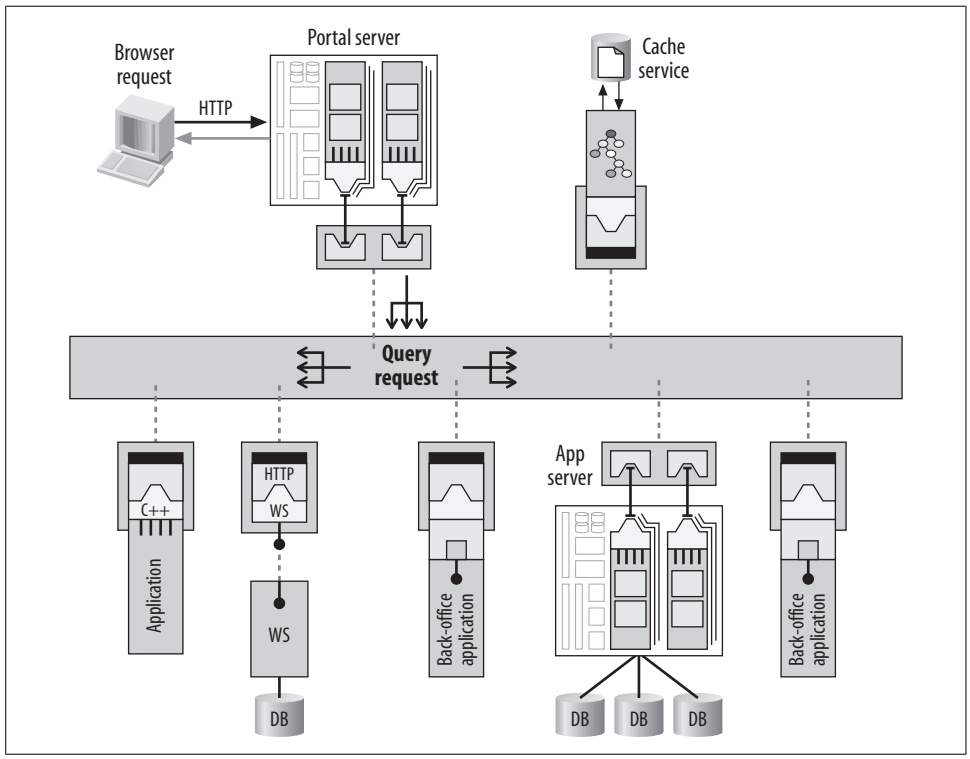

Figure P-6. Diagramming notation showing publish-and-subscribe messaging across an ESB

Finally, the visual representations of the abstract patterns, the process flow descriptions, and the physical characteristics of the bus and its components may be combined, as illustrated in Figure P-8.

Some of these diagrams may look complex at first; their meanings should become clearer when they are used to explain the concepts of the ESB throughout the book. Seeing the concepts illustrated in these diagrams will help you to understand what an ESB is, and how it is being used in real integration projects today.

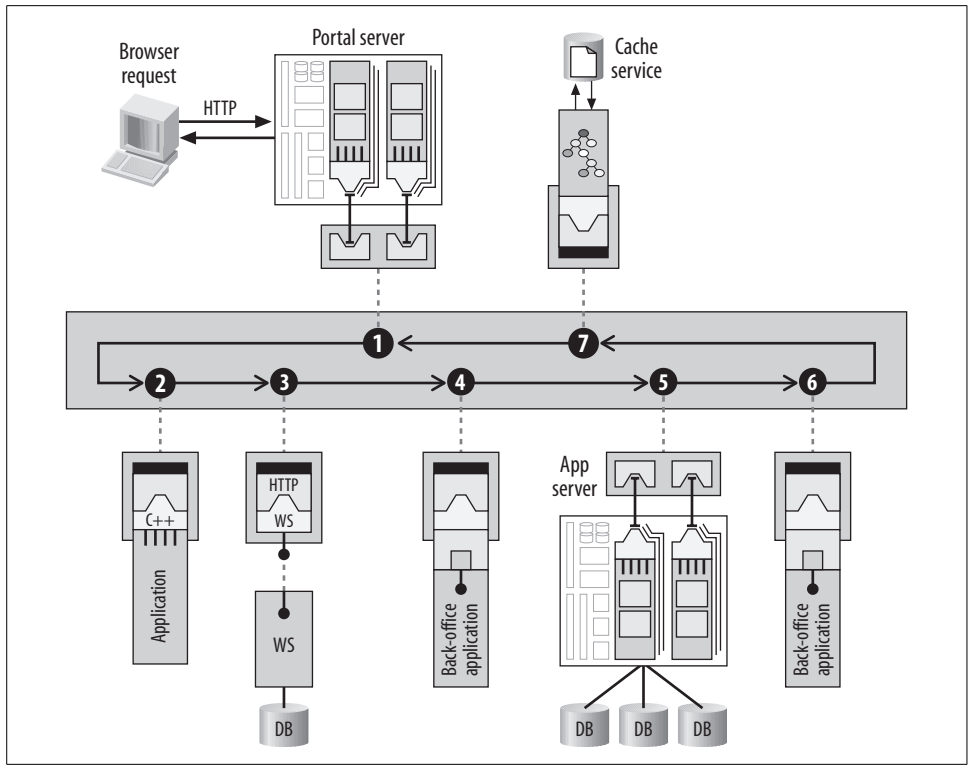

Figure P-7. Diagramming notation showing multiple steps in a process flow

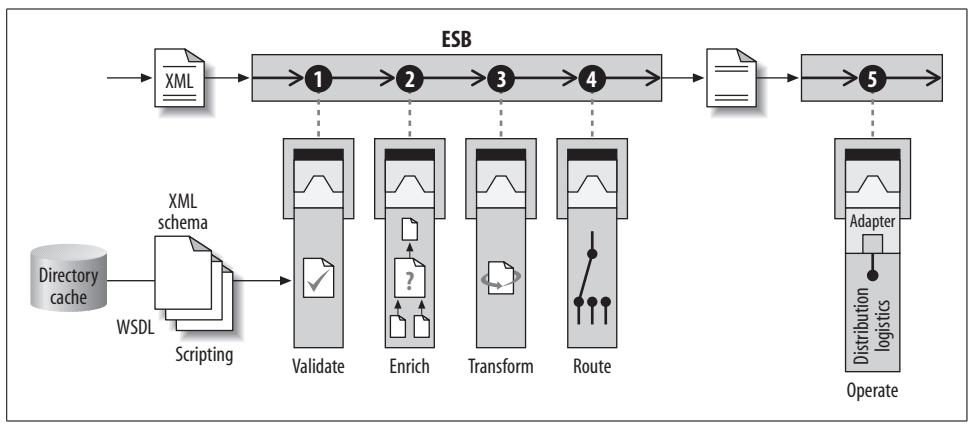

Figure P-8. Using the diagramming notation to describe the abstract pattern definition combined with the process flow definition

Conventions Used in This Book

The following typographical conventions are used in this book:

Plain text
: Indicates menu titles, menu options, menu buttons, and keyboard accelerators (such as Alt and Ctrl).

Italic
: Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities.

`Constant width`
: Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands.

`Constant width bold`
: Is used to show emphasis in code examples.

We'd Like to Hear from You

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

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

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

http://www.oreilly.com/catalog/esb

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

bookquestions@oreilly.com

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

http://www.oreilly.com

Acknowledgments

Over the course of writing this book, I interacted with countless people, all of whom helped me in some way toward getting this book written. First, I would like to thank those who provided creative input into the content of the book and provided answers to my questions along the way: Jeanne Abmayr, Dennis Attinger, Jonathan Bachman, Jon Chappell, Ray Christopher, Rob Conti, Glen Daniels, Tim Dempsey, Gary Hemdal, Mitchell Horowitz, Sonali Kanoujia, Rajiv Kotyan, Trip Kucera, Luis Maldonato, Jaime Meritt, Paul Moxon, Stephen Neal, Matt Rothera, George St. Maurice, Mike Theroux, Jamison Tomasek, and Hub Vandervoort.

Of special note I would like to recognize Jonathan Bachman, Gary Hemdal, and Hub Vandervoort for their creative input and participation in helping to formulate the visual diagramming notation that is used in this book.

Also, I would like to thank those who provided substantial feedback during the review period: Jean-Michel Coeur, Paul Connaughton, John Crupi, Bill Cullen, Tim De Borger, Chris Ferris, David Grigglestone, David Hentchel, Gregor Hohpe, Rick Kuzyk, Ted Neward, Ken Schwarz, Gordon Van Huizen, Leonard Walstad, and Bill Wood.

Of particular note are these reviewers, who provided by far the most thorough and constructive feedback: David Hentchel, Gregor Hohpe, Leonard Walstad, and Bill Wood.

Lastly, I would like to acknowledge the many members of the Sonic Software engineering team, including Bill Cullen, Jaime Meritt, and Bill Wood, for their innovation and contributions to the creation and ongoing development of the ESB as a technology concept.

1 Introduction to the Enterprise Service Bus

Across all industries, executives are demanding more value from their strategic business processes. What process is strategic to a given company may vary dramatically by industry, but a common theme is that CEOs want their IT organizations to measurably improve the flow of data and information driving key business decisions. Whether it's a financial services firm seeking a competitive advantage by guaranteeing a higher volume of faster foreign exchange trades, a retail chain looking to accelerate the flow of store data back to brand managers at corporate headquarters, or a building materials supplier striving to optimize order flow through a complex distribution chain, there are common and significant technical challenges to be overcome. Information is locked up in applications within different departments and organizations, and it is both time-consuming and costly to pry that data loose. In short, the enterprise is far from integrated.

The past several years have seen some significant technology trends, such as Service Oriented Architecture (SOA), Enterprise Application Integration (EAI), Business-to-Business (B2B), and web services. These technologies have attempted to address the challenges of improving the results and increasing the value of integrated business processes, and have garnered the widespread attention of IT leaders, vendors, and industry analysts. The Enterprise Service Bus (ESB) draws the best traits from these and other technology trends.

The ESB concept is a new approach to integration that can provide the underpinnings for a loosely coupled, highly distributed integration network that can scale beyond the limits of a hub-and-spoke EAI broker. An ESB is a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across *extended enterprises* with transactional integrity.

An *extended enterprise* represents an organization and its business partners, which are separated by both business boundaries and physical boundaries. In an extended enterprise, even the applications that are under the control of a single corporation may be separated by geographic dispersion, corporate firewalls, and interdepartmental security policies.

SOA in an Event-Driven Enterprise

In an *event-driven enterprise*, business events that affect the normal course of a business process can occur in any order and at any time. Applications that exchange data in automated business processes need to communicate with each other using an event-driven SOA to have the agility to react to changing business requirements. An SOA provides a business analyst or integration architect with a broad abstract view of applications and integration components to be dealt with as high-level services. In an ESB, applications and event-driven services are tied together in an SOA in a loosely coupled fashion, which allows them to operate independently from one another while still providing value to a broader business function.

> In the realm of SOA, events are represented in an open XML format and flow through a transparent pipeline that's open to inspection and subject to intermediation...
>
> — John Udell, *InfoWorld*

Service components in an SOA expose coarse-grained interfaces with the purpose of sharing data asynchronously between applications. Using an ESB, an integration architect pulls together applications and discrete integration components to create assemblies of services to form composite business processes, which in turn automate business functions in a real-time enterprise.

An ESB provides the implementation backbone for an SOA. That is, it provides a loosely coupled, event-driven SOA with a highly distributed universe of named routing destinations across a multiprotocol message bus. Applications (and integration components) in the ESB are abstractly decoupled from each other, and connect together through the bus as logical endpoints that are exposed as event-driven services.

With its distributed deployment infrastructure, an ESB can efficiently provide central configuration, deployment, and management of services that are distributed across the extended enterprise.

A New Approach to Pervasive Integration

The common goal of applying technologies such as SOA, EAI, B2B, and web services is to create an architecture for integration that can be pervasive across an extended enterprise and beyond. For an integration infrastructure to achieve this pervasiveness, it must have the following characteristics:

- It must adapt to suit the needs of general-purpose integration projects across a variety of integration situations, large and small. Adaptability includes providing a durable architecture that is capable of withstanding evolutions in protocols, interface technology, and even process modeling trends.
- It must link together applications that span the extended enterprise using a single unified approach and a common infrastructure.