TIBCO®
The Power of Now®

Realizing the Two-Second Advantage

TIBCO® Architecture Fundamentals

Paul C. Brown

Praise for *TIBCO® Architecture Fundamentals*

"*TIBCO® Architecture Fundamentals* is a must-read for anybody involved with the architecture and design of distributed systems, with system integration issues, or with service-based application design. In particular, solution architects responsible for TIBCO-based systems architectures should consider reading this book and its planned follow-on titles.

"The product portfolio of TIBCO today is simply too broad for anybody to have an ongoing detailed understanding of what is in there and what elements of the portfolio are best suited in a given business scenario. Paul Brown provides the required oversight in this book, helping both experienced solution architects and newcomers in the field find their way through the myriad technology options TIBCO offers today."

—*Bert Hooyman, Chief Architect, Europe for MphasiS (an HP Company)*

"In his previous books, Dr. Brown developed the 'total architecture concept' in a generic setting. In this one, he presents a concrete application of it to the TIBCO product line. It will be a valuable resource to anyone developing solutions with those tools."

—*Glenn Smith, Principal Consultant, Appian*

"This material is spot on for what is needed in enterprises today, to give a level set to all the architecture teams and project teams they interact with, to outline what is expected, and the roles that each play. In addition, it is a timely overview of the latest TIBCO product suites, and I am anxious to see the follow-ups to this (BusinessEvents- and BPM-focused materials).

"This book provides a detailed look at what happens in the creation of an integration architecture for a business problem. Paul's attempt to capture in words the years of project experience will be a benefit for groups getting familiar with establishing an enterprise architecture standard, as well as a refresher for those performing this function today.

"I would like for all the folks on my team to read this to ensure we are all on the same page with the deliverables that are expected from architecture teams involved in global projects, and the role that the TIBCO tools play in implementing these solutions."

—*Joseph G. Meyer, Director of Architecture Services and R&D, Citi*

"Brown's approach to presenting the highly complex architectural issues is by far the best I have encountered. While each of the individual areas has been detailed in other texts, this is the only publication I have read that lays out each aspect of the architectural issues and describes them in an easy-to-read, comfortable style."

—*James G. Keegan Jr., President, Intrepico, Inc.*

"I recommend the book wholeheartedly. The combination of breadth and depth is not usually found in technical books."

—*Lloyd Fischer, Senior Software Architect, WellCare Health Plans*

This page intentionally left blank

TIBCO® Architecture Fundamentals

This page intentionally left blank

TIBCO® Architecture Fundamentals

Paul C. Brown

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Brown, Paul C.
TIBCO architecture fundamentals / Paul C. Brown.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-321-77261-9 (pbk. : alk. paper) 1. Service-oriented architecture (Computer science) 2. Business—Data processing. I. Title.
TK5105.5828.B76 2011
00.5—dc22
2011006244

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-77261-9
ISBN-10: 0-321-77261-X
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing, May 2011

For Jessica and Philip,
my most prized creations.

This page intentionally left blank

Contents

Preface xvii

Acknowledgments xxiii

About the Author xxv

PART I: Concepts 1

Chapter 1: The IT World Is Evolving 3

- From Systems to Processes 3
- Architecture and Architects 7
- Summary 8

Chapter 2: The Scope of Total Architecture 9

Chapter 3: Aspects of Architecture 13

- Process Models 13
- Architecture Patterns 17
- Process-Pattern Mapping 18
- Why Should You Care about Architecture? 19
- An ATM Architecture Example 20
  - ATM Architecture Pattern 20
  - ATM Withdraw Cash Process Model 21
  - ATM Withdraw Cash Process-Pattern Mapping 24
- ATM Architecture Example with Services 25
- Summary 26

Chapter 4: Reference Architecture 29

- Reference Process Model 30
- Reference Architecture Pattern 31

Reference Process-Pattern Mapping 32
Applications of Reference Architectures 32
Summary 33

Chapter 5: Architects and Their Roles 35

Business Processes and Organizational Silos 35
Development Processes 36
The Architecture Step 38
The Project Charter 40
Quantifying Business Expectations 40
Establishing Cost and Schedule Expectations 41
Quantifying Business Process Risks 41
The Integration Test Step 42
Architecture Improves Project Schedules 42
The Roles of Project and Enterprise Architects 44
Project Architect Responsibilities 45
Defining the End-to-End Business Process and Systems Dialog 45
Identifying and Applying Reference Architectures 45
Identifying and Applying Existing Services 46
Identifying New Service Opportunities 46
Enterprise Architect Responsibilities 47
Defining the Target Architecture for the Enterprise 47
Defining a Practical Evolution Strategy 47
Defining Reference Architecture(s) Consistent with the Target Architecture 48
Guiding Project Teams in Evolving toward the Enterprise Architecture 48
Directly Participating in Projects Requiring Complex Designs 49
Train and Mentor Project Architects 49
The Importance of Vision 50
Summary 51

Chapter 6: SCA Concepts and Notation **53**

An Example Service Design 54
Components and Composites 55
Services 56
References 57
Component Type 58
Implementation Type 59
Complex Composites 59
Summary 60

PART II: TIBCO Product Architecture **61**

Chapter 7: The TIBCO Product Suite **63**

Chapter 8: TIBCO Enterprise Message Service™ **67**

Enterprise Message Service™ Product Structure 67
Message Delivery Transports 69
Conventional Message Delivery 69
High-Fanout Message Delivery 69
Multicast Message Delivery 70
Enterprise Message Service Feature Highlights 72

Chapter 9: TIBCO ActiveMatrix® **73**

The TIBCO ActiveMatrix Product Suite 73
Basic TIBCO ActiveMatrix Architecture Patterns 74
Implementation Types 75
Binding Types 77
ActiveMatrix Node 78
TIBCO ActiveMatrix Service Bus 78
TIBCO ActiveMatrix Service Grid 81
ActiveMatrix Environments and Administration 82
Perspectives on Run-Time Environments 82
Logical Environments 83
Physical Environments 83

Administration Organization 84
ActiveMatrix File System Folder Structures 86
ActiveMatrix Solution Life Cycle 88
Deploying SCA Designs on ActiveMatrix Nodes 91
Service and Component Deployment 91
Service, Component, and Reference Deployment 92
Complex Composite 94
TIBCO ActiveMatrix BPM 96
BPM Functional Organization 96
BPM Solution Deployment 98
Summary 98

Chapter 10: TIBCO BusinessEvents™ 101

Complex Event Processing 101
Information Extraction, Caching, and Persistence 103
State Machine Modeling 103
Event Channels 104
Rules and Decisions 105
Queries 105
Visualization 105
BusinessEvents Solution Roles 106
Basic Solution Role of a Complex Event Processor 106
Director Role 106
TIBCO BusinessEvents Product Suite 107
TIBCO BusinessEvents™ Views 108
TIBCO BusinessEvents™ Data Modeling 108
TIBCO BusinessEvents™ Decision Manager 109
TIBCO BusinessEvents™ Event Stream Processing 109
BusinessEvents Solution Deployment 110
BusinessEvents Solution Life Cycle 112
Summary 114

PART III: Design Patterns with TIBCO ActiveMatrix® 117

Chapter 11: Basic Interaction Patterns 119

Basic Interaction Pattern Overview 120
Example Case Study: A Newspaper 121
In-Only Example and Implementation Options 122
In-Out Example and Implementation Options 123
Synchronous Variation 124
Asynchronous Variations 125
Out-Only Example and Implementation Options 127
Out-In Example and Implementation Options 130
Summary 131

Chapter 12: Event-Driven Interaction Patterns 133

The Pub-Sub Architecture Pattern 134
Queue Semantics 135
Topic Semantics 137
Bridge Semantics 137
Other Sources of Events 139
Summary 139

Chapter 13: ActiveMatrix Policy Framework 141

Aspect-Oriented Design 141
The ActiveMatrix Policy Approach 143
Policies and Policy Sets 144
Policy 144
Policy Sets 144
Policy Set Templates 146
Policy Applicability 148
Policy Enforcement Points 148
Associating Policy Sets with Design Elements 148
Policies That Access External Systems 150
An Example: Implementing a Policy Accessing LDAP 153
Policy Intents 157
Summary 158

Chapter 14: Mediation Patterns **161**

Straight-Wire Mapping 162

Mediation Flow Design 163

Use Case: Access Control 164

Use Case: Transport Mapping 164

Content Transformation 165

Data Augmentation 166

Routing 168

Mediation Capabilities and Limitations 170

Summary 171

Chapter 15: System Access Patterns **173**

Approaches to Accessing External Systems 173

Application Programming Interface (API) Interaction 174

Database Interaction 174

File-Based Interaction 175

Protocol-Based Interaction 175

The Event Recognition Challenge 175

Combining API and Database Interactions 177

Direct Interaction via ActiveMatrix-Supported Protocols 177

Indirect Interaction via ActiveMatrix Adapters 179

Direct Interaction via Non-ActiveMatrix-Supported Protocols 181

General Considerations 182

Database Interactions 182

File Interactions 183

Summary 183

Chapter 16: Two-Party Coordination Patterns **185**

Fire-and-Forget Coordination 186

Request-Reply Coordination 187

Delegation 188

Delegation with Confirmation 189
Distributed Transactions 190
Two-Phase Commit 191
Messaging and Transactions 193
Distributed Transaction Limitations 193
Third-Party Process Coordinator 194
Compensating Transactions 195
Approximating a Two-Phase Commit with Compensating Transactions 195
Compensating Transaction Strengths and Limitations 195
Summary 197

Chapter 17: Multi-Party Coordination Patterns 199

Multi-Party Fire-and-Forget 200
Multi-Party Request-Reply 200
Multi-Party Delegation with Confirmation 201
Data Validation 202
Types of Validation 202
Where to Validate Impacts Coordination Pattern Selection 203
Multi-Party Breakdown Detection 205
Adding Feedback to Improve Breakdown Detection 205
Third-Party Process Monitoring 206
Evaluating an Architecture for Breakdown Detection 207
Summary 207

PART IV: Building Solutions 209

Chapter 18: Services 211

Traditional Approach 211
Service-Oriented Architecture (SOA) Approach 212
Standardized Data Semantics: Common Data Models 213
Standardized Operation Semantics 213

Benefits of Services 213
Most SOA Benefits Require Service Interface Stability 214
Where Do Services Make Sense? 214
Service Granularity 216
Summary 217

Chapter 19: Solutions 219

Solution Architecture 219
Membership Validation Service 220
Membership Validation Service Requirements 220
Membership Validation Solution Architecture 221
Refinement 224
Process Model Refinement 224
Architecture Pattern Refinement 224
Mapping Refinement 226
Reference Architecture as the Entire Solution 228
Process Model Refinement 228
Architecture Pattern Refinement 230
Mapping Refinement 230
Reference Architecture as a Solution Fragment 231
Architecture Pattern Refinement 232
Mapping Refinement 232
Summary 235

Chapter 20: Beyond Fundamentals 237

Recap 237
Looking Ahead 238

Index 239

Preface

About This Book

The subject matter for this book lies at the intersection of three very broad topics: architecture, solutions, and TIBCO products (Figure P-1). Each of these topics, individually, has been the subject of many volumes. The purpose of this book is to begin to tie these three topics together in a very pragmatic way, providing a foundation for architecting solutions with TIBCO products.

This book is not intended to provide a comprehensive introduction into any one of the three broader topic areas. Nevertheless, some coverage of these topics is a necessary prerequisite to discussing the specifics of architecting solutions with TIBCO products. Part I provides an introduction to some of the essential concepts of architecture. Part II provides a cursory overview of the TIBCO product stack and explores the architecture of some of the most broadly used products,

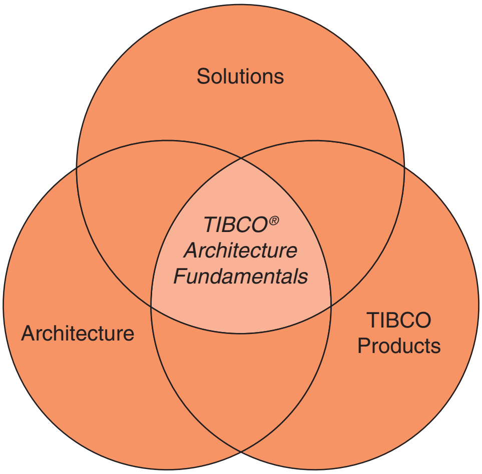

Figure P-1: *Subject Matter for* TIBCO® Architecture Fundamentals emphasizing information not readily found in the individual product manuals. Part III takes a bottom-up approach to exploring the most basic and commonly found design patterns used in architecting solutions with TIBCO products. Part IV begins the discussion of services and solutions, emphasizing the application of the design patterns discussed earlier.

Solutions built with TIBCO products tend to be distributed solutions involving multiple systems, multiple data stores, and multiple business processes along with the people participating in those business processes. Thus, the discussion in this book covers the structure and organization of both the participants and the work being performed, with particular emphasis on the mapping of the work onto the participants.

TIBCO® Architecture Fundamentals lays the groundwork for architecting these systems. Part I provides simple working definitions for architecture and reference architecture. It discusses the roles to be played by project and enterprise architects, and the measurable reduction in project duration (up to 25%) that can be achieved by paying appropriate attention to architecture. Part II discusses the organization of the major TIBCO products and describes how solutions progress from design into production. Part III uses design patterns to explore dozens of design choices defining how people and systems can interact and coordinate their work. Part IV examines solution architecture, exploring the notion of services and discussing how reference architectures can be applied when building solutions.

TIBCO Architecture Book Series

As the first book in a series, *TIBCO® Architecture Fundamentals* only begins the discussion of architecting solutions with TIBCO products (Figure P-2). It lays the foundation for architecting TIBCO-based solutions and serves as a common foundation for the series. Each of the more advanced books explores a different style of solution, all based on TIBCO technology. Each explores the additional TIBCO products relevant to that style of solution. Each defines larger and more specialized architecture patterns relevant to the style, all built on top of the foundational set of design patterns presented in this book.

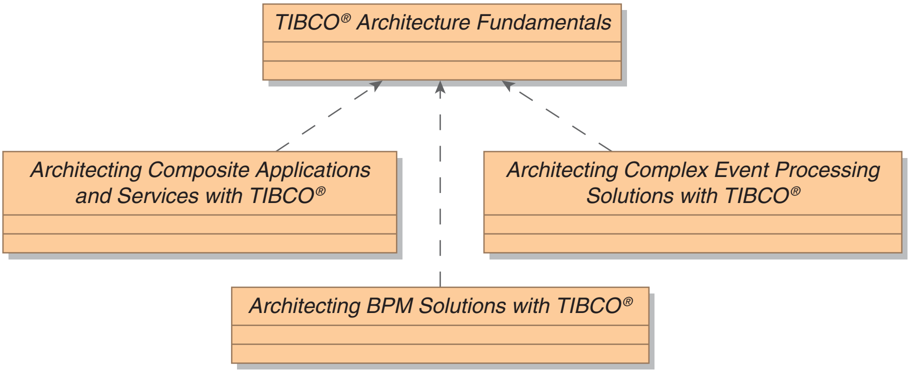
TIBCO® Architecture Fundamentals
Architecting Composite Applications and Services with TIBCO®
Architecting Complex Event Processing Solutions with TIBCO®
Architecting BPM Solutions with TIBCO®

Figure P-2: *Initial TIBCO Architecture Book Series*

Intended Audience

TIBCO® Architecture Fundamentals is written for architects and lead engineers designing solutions in which TIBCO products play a significant role. Enterprise architects will also gain some insight as to how they can employ reference architectures to document design patterns. Such reference architectures give voice to their design intent and serve to efficiently give direction to project teams.

To derive maximum benefit from this book, it is useful for the reader to already have some familiarity with the TIBCO product set. The provided overview of the major TIBCO products and their organization is supplementary and is intended to augment the information contained in the product manuals.

Throughout this book the majority of the diagrams employ UML notations, particularly Class, Activity, and Composite Structure diagrams, with occasional use of other UML notations. For the most part, the meaning of these diagrams should be intuitively obvious, and thus a formal understanding of the UML notation is not a requirement for reading this book. On the other hand, the UML notations have a formality and precision that, when properly understood, allow the reader to extract even more information from the diagrams. The *Unified Modeling Language Reference Manual, Second Edition*,[1] is an excellent reference in this regard.

[1] James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual, Second Edition*, Boston: Addison-Wesley (2004).

Detailed Learning Objectives

After reading this book, you should be able to:

- Explain the design perspective required for modern IT projects and the concepts of total architecture, architecture, and reference architecture
- Predict the positive impact that architecture can have upon project duration, and explain the roles of project and enterprise architects in achieving this benefit
- Explain the basic SCA concepts and read an SCA diagram
- Describe the core TIBCO products: TIBCO Enterprise Message Service™, TIBCO ActiveMatrix® Service Bus, TIBCO ActiveMatrix® Service Grid, TIBCO ActiveMatrix® BPM, and TIBCO BusinessEvents™
- Select appropriate design patterns for basic system interactions and identify and select the appropriate TIBCO products to be used
- Outline the capabilities of policies in TIBCO ActiveMatrix Service Bus
- Select appropriate design patterns for mediation, external system interaction, and coordination of activities
- Explain the concept of a service and the criteria for deciding when an investment in a service is warranted
- Explain how a solution architecture should be characterized and how reference architectures can be applied to the building of solutions

Organization of the Book

The book is structured into four parts, as shown in Figure P-3. Part I covers foundational concepts: architecture, reference architecture, solution architecture, the role of architects, and Service-Component Architecture (SCA). The discussions in this portion of the book are relatively abstract (high level) and technology independent.

Part II covers the architecture of the most commonly used TIBCO products: TIBCO Enterprise Message Service (EMS), the TIBCO ActiveMatrix product suite, and TIBCO BusinessEvents. The discussions in

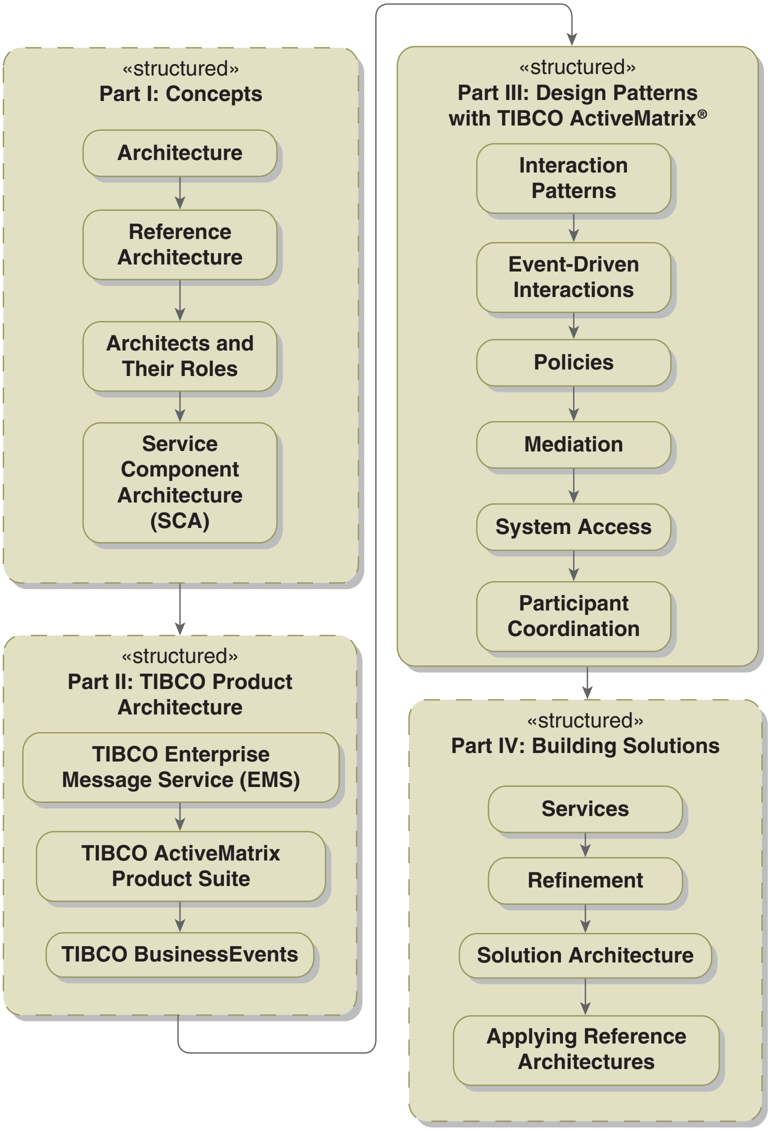
«structured»
Part I: Concepts
Architecture
Reference Architecture
Architects and Their Roles
Service Component Architecture (SCA)
«structured»
Part II: TIBCO Product Architecture
TIBCO Enterprise Message Service (EMS)
TIBCO ActiveMatrix Product Suite
TIBCO BusinessEvents
«structured»
Part III: Design Patterns with TIBCO ActiveMatrix®
Interaction Patterns
Event-Driven Interactions
Policies
Mediation
System Access
Participant Coordination
«structured»
Part IV: Building Solutions
Services
Refinement
Solution Architecture
Applying Reference Architectures

Figure P-3: *Book Structure*

this section are technology specific and detailed, getting into the product structure and architecture. Although the discussions are specific to the current version of the products (TIBCO Enterprise Message Service 6.x, TIBCO ActiveMatrix Service Bus and Service Grid 3.x, TIBCO

ActiveMatrix BusinessWorks 5.9, TIBCO ActiveMatrix BPM 1.x, and TIBCO BusinessEvents 4.x), most of the discussions will remain valid as these products evolve. Most product changes will result in augmentations rather than alterations.

Part III examines foundational design patterns: interactions between pairs of components, event-driven interactions, policies, mediation, external system access, and the coordination of activities. The discussions in this section are a mixture of technology-neutral design patterns and product-specific implementation choices for these patterns. Some discussions, particularly those surrounding policies, get quite detailed.

Part IV looks at building solutions, examining the concept of services, building solutions through the process of refinement, and applying reference architectures (design patterns). The discussions in this section are, once again, abstract (high level) and technology independent.

The book is intended to be read linearly, but there is some flexibility in this. Parts I and II can be read independently, but the discussions in Part III require an understanding of both prior parts. Part IV can be read after Part I, but the reader will find its discussion more compelling if Parts II and III have been read first.

Acknowledgments

Presenting material that touches on as many topics as this book does is, to say the least, challenging. This book series, and in fact the entire approach to presenting the material, would never have occurred without the persistent combination of challenge and encouragement provided by Michael Fallon, Madan Mashalkar, and Alan Brown over the past decade. Through them I have learned a great deal about both the challenges and techniques of knowledge transfer.

The design patterns presented in this book are a synthesis of the collective experience of the TIBCO global architects with whom I have worked over the years: Dave Ashton, Pong-Ning Ching, Richard Flather, Ben Gundry, Nochum Klein, Dave Leigh, Marco Malva, and Janet Strong. It is through their collaboration and the efforts of the other field architects that these patterns have been explored, refined, and tested.

I have received much support from TIBCO Software Inc. in the production of this book. For this I would like to thank Wen Miao, Paul Asmar, Jan Plutzer, and Murray Rode.

Many people reviewed the draft manuscript and provided valuable feedback. Comments from Bert Hooyman, Ignacio Silva-Lepe, and Lee Kleir led to significant improvements in the structure and content of the book. Feedback from Jose Carlos Estefania Aulet, Michael Blaha, Massimiliano Bonaveri, Antonio Bruno, Lloyd Fischer, Alex Garrison, Yuri Gogolitsyn, Jose Maria Lopez Higuera, Brian Hinsley, Alexandre Jeong, James Keegan, Joseph Meyer, Alexander Orsini, Mohan Sidda, Mark Shelton, and Moritz Weinrich helped to further refine the content. I thank you all for your support.

Finally, I would like to thank my wife, Maria, for supporting me in the writing of yet another book. Without her support, nothing is possible.

This page intentionally left blank

About the Author

Dr. Paul C. Brown is a principal software architect at TIBCO Software Inc., author of *Succeeding with SOA: Realizing Business Value Through Total Architecture* (Addison-Wesley, 2007) and *Implementing SOA: Total Architecture In Practice* (Addison-Wesley, 2008), and a coauthor of the SOA Manifesto (soa-manifesto.org). His model-based tool architectures are the foundation of a diverse family of applications that design distributed control systems, process control interfaces, internal combustion engines, and NASA satellite missions. Dr. Brown's extensive design work on enterprise-scale information systems led him to develop the total architecture concept: Business processes and information systems are so intertwined that they must be architected together. Dr. Brown received his Ph.D. in computer science from Rensselaer Polytechnic Institute and his BSEE from Union College. He is a member of IEEE and ACM.

This page intentionally left blank

Chapter 11

Basic Interaction Patterns

This chapter examines the simplest possible interactions between two parties. The architecture pattern for these discussions (Figure 11-1) is, as you would expect, trivial. It consists of the two parties, here referred to as the service consumer and service provider. Despite the fact that we are referring to services, the patterns being discussed can be generalized to represent any interactions between two parties.

The examination of interactions will consider four of the most common ActiveMatrix protocol and transport options: SOAP over HTTP, SOAP over JMS, SOAP over ActiveMatrix Virtualization, and XML over JMS.

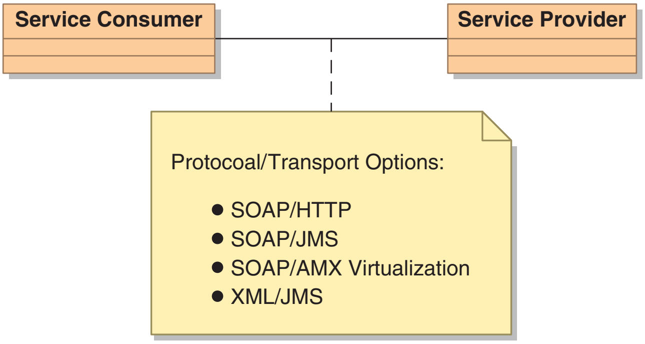

Figure 11-1: *Architecture Pattern for Two-Party Interactions*

Basic Interaction Pattern Overview

There are four basic message exchange patterns between the two parties: In-Only, In-Out, Out-Only, and Out-In. The In-Only pattern is shown in Figure 11-2. In it, the service consumer sends a single message to the service provider and expects no response. The intent is generally that the arrival of the input will trigger the service provider to do something useful. Common examples of this pattern include e-mails and text messages.

The In-Out pattern (Figure 11-3), also referred to as the request-reply pattern, is a simple extension of the In-Only pattern that adds a response (the output) from the service provider. Here the intent is a bit more explicit: The service consumer provides the input and expects the arrival of the input to trigger the service provider to do something and then send a response. This is the pattern you encounter when you execute a search online: You submit the search terms (the input) and expect a list of "hits" as a response (the output).

The Out-Only pattern (Figure 11-4) is very similar to the In-Only pattern, the distinction being that the single message is an output going from the service provider to the service consumer. Common examples of this pattern include announcements of various sorts. It is common in this pattern for there to be many service consumers for a given input (this will be discussed further in Chapter 12). When the service provider is a system of record for some information, this pattern is suitable for announcing changes to this information.

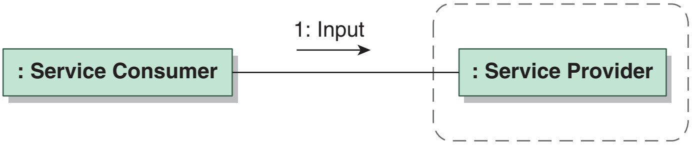

Figure 11-2: *In-Only Pattern*

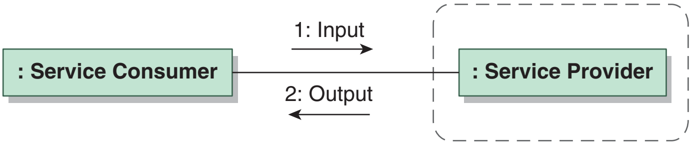

Figure 11-3: *In-Out Pattern*

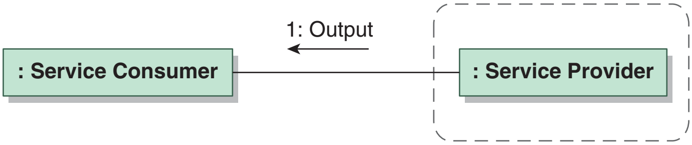

Figure 11-4: *Out-Only Pattern*

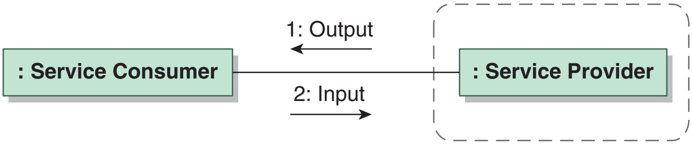

Figure 11-5: *Out-In Pattern*

The Out-In pattern (Figure 11-5) extends the Out-Only pattern to include a response back to the service provider. A common example of this is an automobile recall notice: The manufacturer sends you a notification that there is a defect in your automobile that requires correction. The manufacturer expects a response from you to schedule an appointment and get the defect corrected. Another example is an offer that requires a response.

Example Case Study: A Newspaper

To illustrate these four interaction patterns and their implementation options we will use a simple example based on a newspaper business (Figure 11-6). In this example there are three participants: the newspaper itself, a party acting as a news source, and a customer of the newspaper.

We will examine four use cases (processes) involving these participants:

- The news source delivering a news tip to the newspaper (In-Only)
- The customer subscribing to the newspaper (In-Out)
- The newspaper sending the news electronically to the customer (Out-Only)
- The newspaper sending an offer to the customer that requires a response (Out-In)

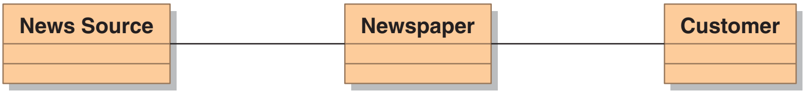

Figure 11-6: *Newspaper Example Architecture Pattern*

In-Only Example and Implementation Options

The In-Only example from the newspaper is the news source sending a news tip to the newspaper (Figure 11-7). Here the news source invokes a `receiveTip()` operation provided by the newspaper's service interface.

If you were to implement both the news source and the newspaper as ActiveMatrix components and indicate the news source's reference of the newspaper's service, the result would be a design similar to that shown in Figure 11-8.

For this design you have four transportation options in ActiveMatrix:

- SOAP over HTTP
- SOAP over JMS

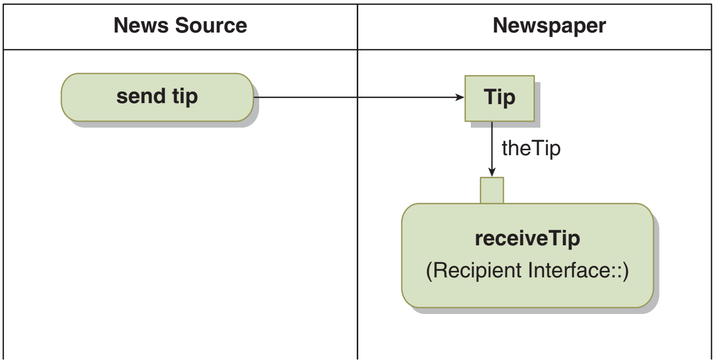

Figure 11-7: *Send Tip Process*

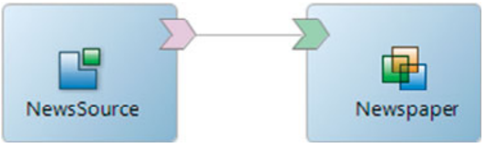

Figure 11-8: *ActiveMatrix Design for Send Tip Process*

- SOAP over ActiveMatrix Virtualization
- XML over JMS

The first of these options uses HTTP as a transport. The implication is that both parties need to be active simultaneously in order for an interaction to occur. The SOAP over JMS and XML over JMS options, because they use a JMS server as a communications intermediary, make it possible for the news source to send the tip when the newspaper is not actively receiving communications. The JMS server will forward the message when the newspaper becomes active.

Despite the fact that ActiveMatrix Virtualization also uses JMS as its underlying communications mechanism, it will not be able to forward a message if the newspaper is not active at the time it is sent. For an explanation, see the "ActiveMatrix Virtualization Transport Limitations" sidebar.

There are five implementation types that would be appropriate for the News Source: TIBCO ActiveMatrix BusinessWorks, Java, C++, Spring, and WebApp. There are four that would be appropriate for the Newspaper: BusinessWorks, Java, C++, and Spring. Note that WebApp would not be appropriate since its input is just the raw HTTP protocol.

ActiveMatrix Virtualization Transport Limitations

When the ActiveMatrix Virtualization transport is used, ActiveMatrix determines the routing between the service consumer and service provider. If the two parties are on different nodes (or if directed by policy), this communication will occur via the JMS server being automatically administered by ActiveMatrix.

When both parties are active, the communications will occur as expected. However, when one or both parties are stopped or undeployed, or the node is stopped, the JMS destination being used for communications between them will be destroyed and any pending messages will be lost.

In-Out Example and Implementation Options

There are two variations on the In-Out pattern: synchronous and asynchronous. In the synchronous pattern, the service consumer (the Subscriber in the example) waits for the response from the service provider (the Newspaper). In the asynchronous variation, the service consumer does not have to wait for the response. Since there are significant differences between these variations in both behavior and implementation options, they will be discussed separately.

Synchronous Variation

The subscribe In-Out process, implemented as a synchronous interaction, is shown in Figure 11-9. The subscriber is invoking the `subscribe()` operation on the newspaper, sending a `SubscribeRequest` and expecting a `SubscribeResponse` in return. In the synchronous variation, the subscriber is actively waiting for the response.

If both subscriber and newspaper were to be implemented as ActiveMatrix components, the result would be a design similar to Figure 11-10.

For this design you have four transportation options in ActiveMatrix:

- SOAP over HTTP
- SOAP over JMS
- SOAP over ActiveMatrix Virtualization
- XML over JMS

For this synchronous variation, the assumption is that both parties are active for the duration of the exchange. The loss of communications or

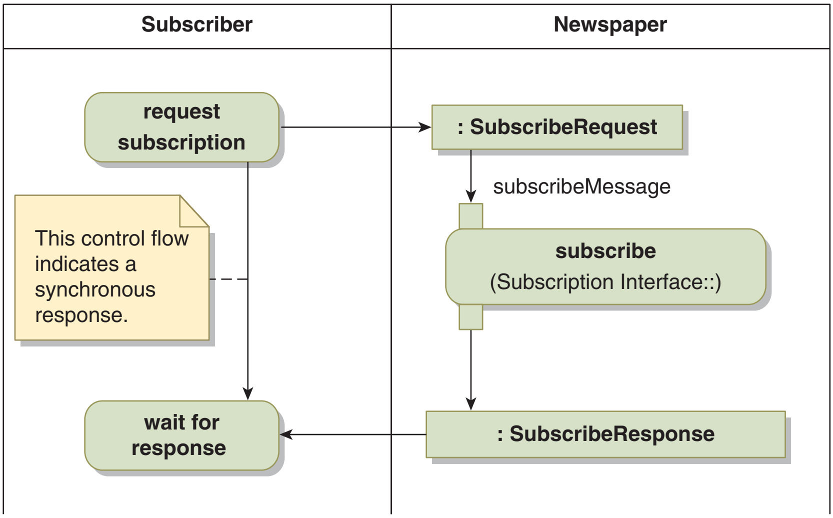

Figure 11-9: *Synchronous Subscribe Process*

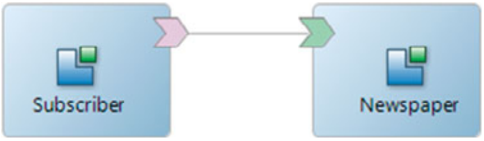

Figure 11-10: *ActiveMatrix Design for Subscribe Process*

the restart of either party may cause exceptions, and both parties should be designed to handle these exceptions gracefully.

There are five implementation types that would be appropriate for the subscriber: BusinessWorks, Java, C++, Spring, and WebApp. There are four that would be appropriate for the Newspaper: BusinessWorks, Java, C++, and Spring. Note that WebApp would not be appropriate since its input is just the raw HTTP protocol.

Asynchronous Variations

There are actually two asynchronous variations for a request-reply exchange. One is the checkpoint pattern shown in Figure 11-11. In this pattern the requestor does not necessarily wait for the reply, but generally must take steps to ensure that, when the reply arrives, it is in a position to handle it. This generally means creating a *checkpoint*, a

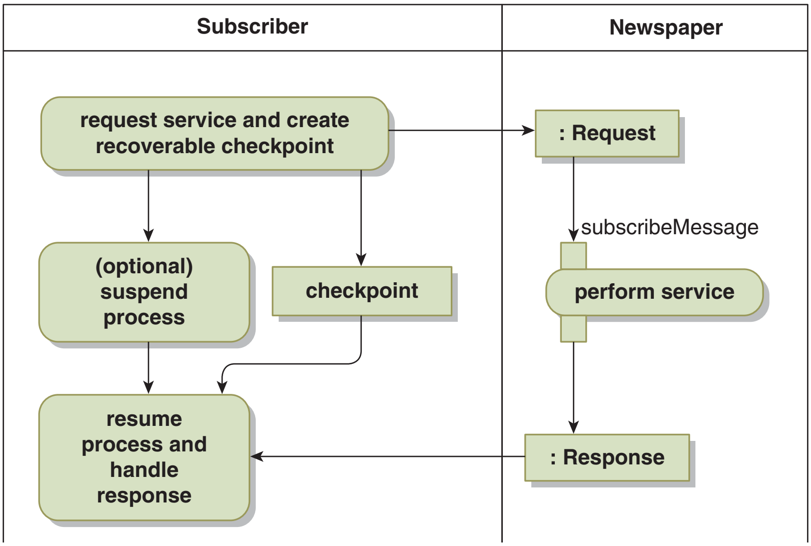
Subscriber
Newspaper
request service and create recoverable checkpoint
: Request
subscribeMessage
(optional) suspend process
checkpoint
perform service
resume process and handle response
: Response

Figure 11-11: *Checkpoint Asynchronous In-Out Pattern*

recoverable snapshot of the requestor's state. In addition, the requestor (in this case the Subscriber) must be implemented in such a way that, should the requestor be halted for any reason, it is resurrected from the checkpoint and is ready to receive the response. Optionally, the process may be suspended to free up resources while waiting for the response.

The checkpoint asynchronous In-Out pattern is typically used when the performance of the requested service is expected to take significant time (minutes or longer). The idea is that, because of the long wait, there is a reasonable possibility that the requestor may be interrupted and you do not want the interruption to adversely impact the execution of the business process. Note, however, that this pattern ties up some resources for each outstanding request.

The other major variation is the third-party asynchronous In-Out pattern shown in Figure 11-12. Here the response is handled by a third party, either a different thread in the requesting process or a completely independent application. In this case there is usually a need for some additional communications between the party sending the request and the party receiving the response.

This additional communication conveys the context information required to handle the response. The content of this context varies from solution to solution, but typically includes information such as:

- Notification that there is an outstanding request. This information (in conjunction with a response-time SLA) enables the response handler to determine when responses are missing or overdue.

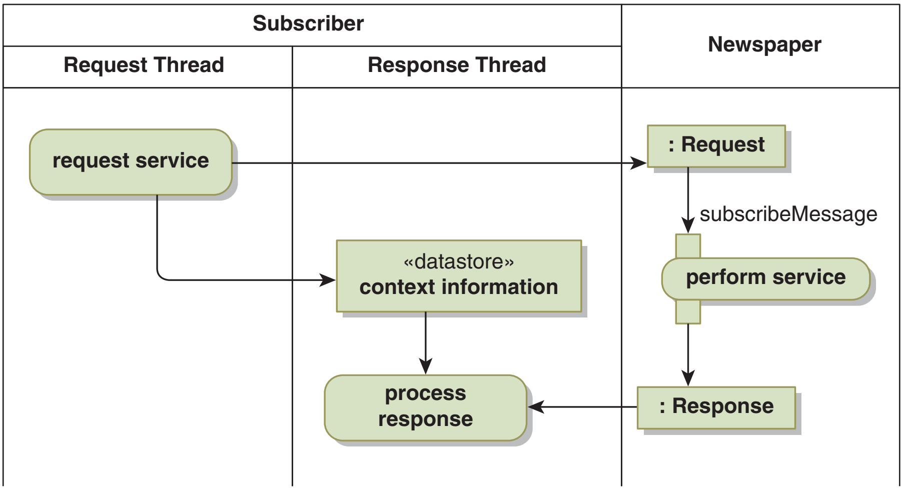

Figure 11-12: *Third-Party Asynchronous In-Out Pattern*

- An identifier for the request that will be returned as part of the response. This allows the response handler to correlate a particular request-response pair.
- Information about the nature of the request needed to properly handle the response. This can be the information itself or a reference to a location (database, file, etc.) in which this information can be found.

The communication of the context information is a design task that should not be overlooked when selecting this pattern. It always requires design and implementation work.

At present the only transport in ActiveMatrix that can support these asynchronous interaction patterns is XML over JMS. When using this transport, the JMSCorrelationID should be used in the request to uniquely identify the request. The value for this field is provided by the requestor and should be returned in the JMSCorrelationID field of the response. Also required is the JMSReplyTo field in the request. Its value should indicate the JMS destination to which the response should be sent.

There are four implementation types that would be appropriate for the subscriber: BusinessWorks, Java, C++, and Spring. The Business Works implementation type is particularly well suited to implementing the request side (e.g., the subscriber) of the checkpoint asynchronous In-Out pattern, as all the mechanisms required for checkpointing and recovery are provided as part of the product. There are four that would be appropriate for the Newspaper: BusinessWorks, Java, C++, and Spring. Note that WebApp would not be appropriate for either role since its input is just the raw HTTP protocol.

Out-Only Example and Implementation Options

The process for delivering the newspaper is shown in Figure 11-13. This Out-Only interaction is inherently asynchronous—the Subscriber is not actively waiting for the paper to be delivered.

The only ActiveMatrix transport that can support this pattern today is XML over JMS.

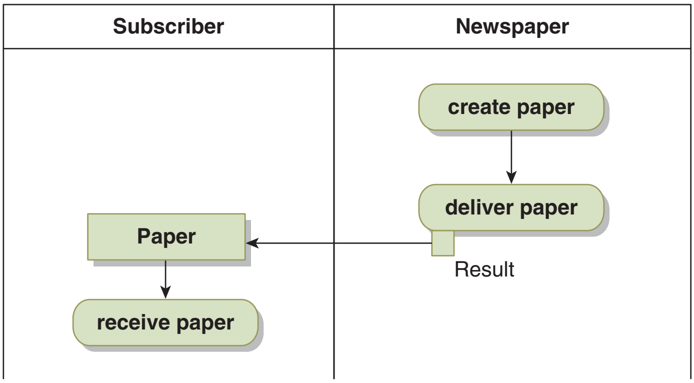

Figure 11-13: *Deliver Paper Process*

The Out-Only pattern is, unfortunately, not well represented in the current version of the SCA notation.[1] The closest you can come in the present notation is the design shown in Figure 11-14. There are two problems with this representation. One is that the diagram implies that it is the subscriber providing the service and the newspaper referencing the service, when in reality the opposite is true. The other is that, for most publications, it is unlikely that the wiring between the Out-Only service provider and service consumer would be done at design time. In other words, it is unlikely that you would ever show both the service provider and service consumer in the same SCA composite. Instead, this wiring would be done either at deployment time or at run time.

What you would create in ActiveMatrix today (until such time as the SCA Event Processing Specification is completed) is a composite containing just the service provider (Figure 11-15). Note that the

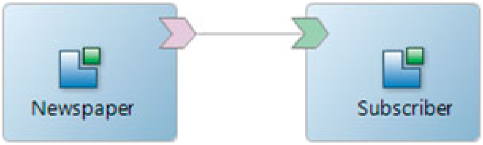

Figure 11-14: *Inappropriate Attempt to Represent Out-Only Pattern in Present SCA Notation*

[1] The SCA Event Processing Specification is presently under development (see www.osoa.org).

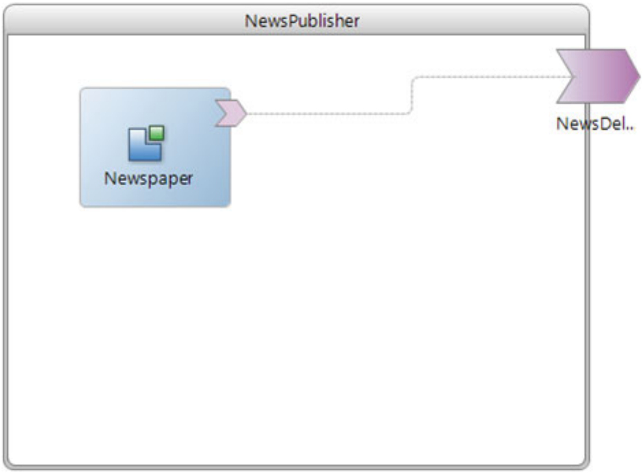

Figure 11-15: *SCA Approximation of an Out-Only Service Provider*

composite shows a reference to the service; the reason is that when you generate implementations, references generate outbound calls, which is consistent with the design intent. This structure (the component referencing the service) can be incorporated into any composite wishing to send Out-Only notifications.

Similarly, you would create the service consumer as a composite with a promoted service (Figure 11-16). From an implementation perspective, this is appropriate since, when you generate the implementation, the generated structure will be appropriate for an inbound call. This structure (the service and its association with a component) can be incorporated into any composite that wishes to receive Out-Only notifications from a service provider.

There is a bit of hidden JMS administrative configuration required to connect the two parties in this pattern. The JMS destination must be created (or the JMS server must be configured to auto-create destinations), and both parties must be configured to use the same destination. This is generally straightforward when the configuration is done at deployment time, but dynamic connection at runtime will require extra design work. For example, if you wanted to have a subscriber dynamically create the subscription, the `subscribe()` operation would have to return the JMS destination and the subscriber would have to have code to alter its configuration to receive messages from this destination.

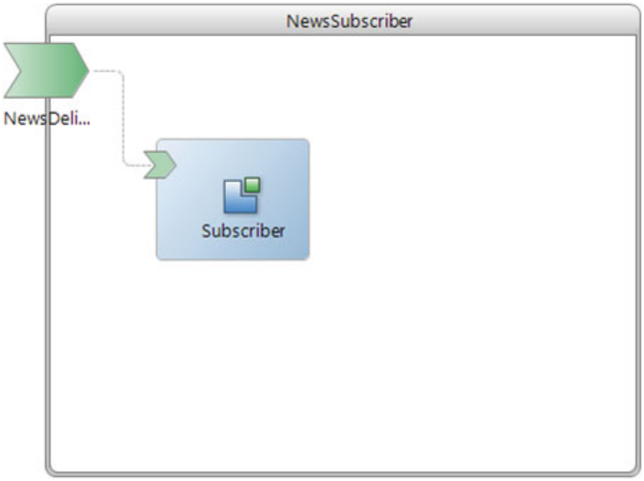

Figure 11-16: *SCA Approximation of an Out-Only Service Consumer*

There are five implementation types that would be appropriate for the newspaper: BusinessWorks, Java, C++, Spring, and WebApp. There are four that would be appropriate for the subscriber: BusinessWorks, Java, C++, and Spring. Note that WebApp would not be appropriate for the subscriber since its input is just the raw HTTP protocol.

Out-In Example and Implementation Options

The process of the newspaper making an offer to a subscriber and then handling the response is shown in Figure 11-17. The interactions here are, by definition, asynchronous: Neither party is actively waiting for an input. Furthermore, the service provider (the newspaper) will likely have separate threads (or applications) for sending the offers and processing the responses. Consequently, there will likely be a need to communicate context information between these two threads as was discussed in the earlier asynchronous In-Out example.

As with the Out-Only pattern, the only suitable protocol and transport combination available in ActiveMatrix is XML over JMS. The SCA design would be similar to that discussed in the Out-Only example, and the JMSCorrelationID and JMSReplyTo properties would have to be used as described in the Asynchronous In-Out example.

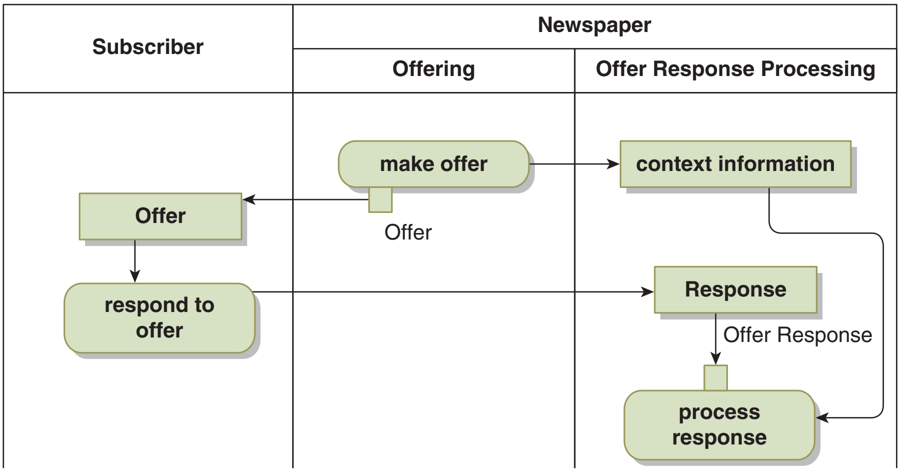

Figure 11-17: *Make Offer Process*

There are five implementation types that would be appropriate for the newspaper: BusinessWorks, Java, C++, and Spring. There are four that would be appropriate for the subscriber: BusinessWorks, Java, C++, and Spring. Note that WebApp would not be appropriate for either role since its input is just the raw HTTP protocol.

Summary

There are four basic message exchange patterns between two parties: In-Only, In-Out, Out-Only, and Out-In. The In-Only pattern and the synchronous variation of the In-Out pattern have many protocol and transport options in ActiveMatrix, including SOAP over HTTP, JMS, and ActiveMatrix Virtualization as well as XML over JMS. The BusinessWorks, Java, C++, Spring, and WebApp implementation types are all suitable for the service-consumer side of these interactions, while the BusinessWorks, Java, C++, and Spring implementation types are appropriate for the service-provider side.

The asynchronous variation of the In-Out pattern and the Out-Only and Out-In patterns all involve asynchronous interactions. At present, the only suitable protocol and transport combination in ActiveMatrix for asynchronous interactions is XML over JMS. For the asynchronous In-Out and Out-In, the JMSCorrelationID and JMSReplyTo properties should be used to correlate the request and reply messages and indicate the JMS destination to which the replies should be sent. For these patterns, the BusinessWorks, Java, C++, and Spring implementation types are all suitable for both parties.

Index

A

Abstraction. *See* Reference architecture
Access control
  Aspect-Oriented Design considerations, 141–143
  direct interaction via ActiveMatrix-supported protocols and, 178
  policy enforcement points in, 78–79, 148
  standardizing using services, 212
  straight-wire mapping mediation pattern and, 164
Accidental architecture, 7
ActiveMatrix adapters, 178–179
ActiveMatrix composite implementation type, 76
ActiveMatrix hosts
  administration organization of, 84–86
  architecture pattern and, 88–89
  configuration folder, 87
  creating with TIBCO Configuration Tool, 86–87
  folders for, 87
  overview of, 80
  physical environment, 83–84
  SOAP over ActiveMatrix Virtualization used only with, 178
  solution life cycle and, 88–91
ActiveMatrix nodes
  deploying SCA designs on, 91–96
  enforcing policies in, 148, 159
  example of, 78–79
  within internal structure, 74–75
  in logical environments, 83–84
  overview of, 78
  in physical environments, 83–84
  as Service Bus element, 80
ActiveMatrix policy framework
  accessing external systems, 150–153
  accessing LDAP, 153–157
  approach to, 143–144
  Aspect-Oriented Design, 141–143
  associating policy sets with design elements, 148–150
  policy applicability, 148
  policy enforcement points, 148
  policy intents, 157–158
  policy set templates, 146–148
  policy sets, 144–146
  summary review, 158–159
ActiveMatrix Virtualization transport, 123
Activities
  ATM withdraw cash process example, 21–22
  implementing in parallel, 23–24
  process-pattern mapping, 18–19
  structuring through process models, 13–16
Adapter binding type, 77
Adapter SDK, TIBCO®, 180–181
Adapters. *See* TIBCO ActiveMatrix adapters
Add Resource dialog, policy in LDAP, 155–156
Administration
  stand-alone EMS tool for, 69
  using Administrator. *See* TIBCO ActiveMatrix® Administrator
Advice, in Aspect-Oriented Design, 142
Agile development process, 37
AMX hosts. *See* ActiveMatrix hosts
Announcements
  bridge delivery semantics for, 137–138
  Out-Only message pattern, 120–121
  requests vs., 133
  topic delivery semantics for, 137

API (application programming interface)
 accessing external systems, 174
 combining Database interactions with, 177
Application program, two-phase commit transactions, 191–193
Applications
 policy, 148
 reference architecture, 32–33
Architects
 architecture and, 7
 maintaining total perspective, 11
 project vs. enterprise, 7–8
Architects, roles of
 avoiding policeman approach, 40
 business processes and organizational silos, 35–36
 creating architecture steps, 38–39
 development processes, 36–37
 enterprise architects, 47–49
 importance of vision, 50–51
 improving project schedules, 42–44
 integration test step, 42
 project architects, 44–46
 project charter, 40–42
 summary review, 51–52
Architecture, aspects of
 architecture patterns, 17–18
 ATM architecture example, 20–25
 ATM architecture example with services, 25–26
 overview of, 13
 process models, 13–16
 process-pattern mapping, 18–19
 reasons to care about architecture, 19–20
 summary review, 26–27
Architecture concepts
 architects. *See* Architects
 business process management, 5–6
 collaborative business process design focus, 5
 service-oriented architecture design focus, 3–5
 summary review, 8
 system-centric design focus of past, 3–4
Architecture patterns
 accessing external systems, 151
 ActiveMatrix administrative, 88–89
 data augmentation, 166–167
 data transformation, 165–166
 direct interaction via ActiveMatrix-supported protocols, 178
 evaluating for breakdown detection, 207–208
 mediation, 161
 Membership Validation Service, 222
 multicast message delivery, 69–71
 overview of, 17–18
 process-pattern mapping. *See* Mapping, process-pattern
 pub-sub, 134–135
 reference, 31–33
 reference architecture used as entire solution, 230
 reference architecture used as fragment of, 232
 routing, 168–169
 sketch of, 13–14
 solution architecture, 219, 224–226
 straight-wire mapping, 162
 system-initiated direct interaction via non-ActiveMatrix protocol, 181–182
 system-initiated indirect interaction via adapters, 179
 TIBCO ActiveMatrix®, 74–78
 TIBCO BusinessEvents deployment, 111
 TIBCO BusinessEvents™ life cycle, 113
 two-party interactions, 119
 why you should care about, 19–20
Architecture, solution. *See* Solution architecture
Architecture step, 37–39
Architecture vision, of enterprise architect, 47
Aspect-Oriented Design, 141–143
Asynchronous delegation with confirmation pattern, 189–190
Asynchronous In-Out message pattern, 125–127
Asynchronous Out-In message pattern, 130–131

Asynchronous request-reply coordination, 188
ATM (automated teller machine) architecture example
  architecture pattern, 20–21
  architecture pattern refinement, 224–226
  Aspect-Oriented Design in, 141–142
  defined, 20
  process-pattern mapping after refinement, 226–228
  with services, 25–26
  withdraw cash process model, 21–24
  withdraw cash process-pattern mapping, 24–25
Authentication
  ATM withdraw cash process example, 21–22
  EMS supporting JAAS for, 72
  Service Bus policy templates for, 147
  using policies for, 144
Authorization
  ATM withdraw cash process example, 21–22
  disbursal, 24–25
  EMS supporting JAAS and JACI for, 72
  Service Bus policy templates for, 147
  using policies for, 144
Automated teller machine. *See* ATM (automated teller machine) architecture example
Automobile recall notice example, Out-In pattern, 121

B

Back-end systems
  in architecture step, 38–39
  placing validation in, 204
Balancing Agility and Discipline: A Guide for the Perplexed (Boehm and Turner), 37
Basic route, mediation flow, 169
Behavior, addressing concern in design with, 142
Binding types, 77, 80
BPM (business process management). *See also* TIBCO ActiveMatrix® BPM
  business processes and organizational silos, 35–36
  design focus, 5–6
  TIBCO product suite for, 64–65
BPM composite, TIBCO ActiveMatrix® BPM, 98
Breakdown detection, multi-party
  adding feedback to improve, 205
  coordination patterns, 205–208
  delegation with confirmation pattern, 201–202
  evaluating architecture, 207–208
  request-reply confirmation pattern, 200–201
  third-party process monitoring for, 206–207
Breakdown detection, two-party
  compensating transactions, 195
  delegation pattern, 189
  impossible with fire-and-forget coordination, 186
  overview of, 185
  request-reply confirmation pattern, 187
Bridge semantics, event-driven interaction patterns, 137–138
Browser-based interfaces, TIBCO ActiveMatrix® BPM, 97
Business expectations, project charter, 40–42
Business process management. *See* BPM (business process management); TIBCO ActiveMatrix® BPM
Business processes
  in architecture step, 38–39
  collaborative, 5
  identifying in solution architecture, 219
  Membership Validation solution architecture, 221–222
  organizational silos and, 35–36
  project architect responsibilities, 45
  project charter quantifying risks in, 41–42
  scope of total architecture, 9–11
  TIBCO product suite for, 64
Business Studio. *See* TIBCO Business Studio™

Business Works. *See* TIBCO ActiveMatrix™ BusinessWorks™
BusinessEvents. *See* TIBCO BusinessEvents™
BWSE (TIBCO Business Works™ Service Engine), 76

C

C++ implementation type
- defined, 75
- TIBCO ActiveMatrix® Service Grid and, 81–82
- TIBCO supporting, 56

C programming language, EMS client library for, 68–69
C# programming language, EMS client library for, 68–69
Categories, Service Bus policy templates, 147–148
Central Administration server, 69
Change Data Capture, 177
check order status
- process model, 15–16
- process-pattern mapping, 18

Checkpoint asynchronous In-Out pattern, 125–126
Cloud platform, TIBCO product suite for, 64
COBOL, EMS client library for, 68–69
Collaborative business processes, 5–6
Combining API and database interactions, 177
Communication
- of architectural vision, 50–51
- TIBCO Architecture Fundamentals, 63

Compensating transactions, 195–197
Complex composites, SCA, 59–60, 94–96
Complex designs, and enterprise architect, 49
Complex event processing, BusinessEvents
- basic solution role of, 106
- capturing technical events in, 138
- defined, 65
- event channels, 104–105
- information extraction, caching and persistence, 103
- overview of, 101–102
- queries, 105
- rules and decisions, 105
- state machine modeling, 103–104
- visualization, 105

Component type, SCA, 58–59
Components
- administrator, 86
- TIBCO ActiveMatrix® BPM, 98
- TIBCO Enterprise Message Service™, 67–69

Components, SCA
- deploying SCA designs on ActiveMatrix nodes, 91–94
- overview of, 55–56
- with reference, 57–58
- services, 56–57
- summary review, 60

Composites, SCA
- ActiveMatrix composite implementation type, 76
- associating policy sets with, 154–155
- BPM solution, 98
- complex, 59–60
- components contained within, 55–56
- mediation flow in, 162
- with promoted services, 56–57
- summary review, 60
- TIBCO Active Matrix product suite for, 64–65

Concerns
- Aspect-Oriented Design addressing, 141–142
- Service Bus policies addressing, 143–144

Configuration folder, ActiveMatrix, 86–87
Content transformation, mediation, 165–166, 171
Conventional delivery, TIBCO Enterprise Message Service™, 69–70
Coordination patterns
- multi-party. *See* Multi-party coordination patterns
- two-party. *See* Two-party coordination patterns

Cost, business expectations for project, 41
Credential mapping, Service Bus policy templates for, 148

Credential server keystore, administrator, 86
Credentials, ATM withdraw cash process, 21–22
Crosscutting concern, in Aspect-Oriented Design, 141–142

D

DAA (distributed application archive), 89–90, 151
Data augmentation, mediation, 166–168, 171–172
Data transformation mediation flow, 165–166
Data validation, multi-party coordination patterns, 202–205
Database
  administrator, 86
  implementing two-phase commit transactions with, 191
Database adapter, 176, 182–183
Database interactions
  accessing external systems, 174–175
  combining API interactions with, 177
  strategies for, 182–183
Database triggers, 174, 176–177
Delegation pattern, two-party coordination, 188–189, 198
Delegation with confirmation pattern
  multi-party, 201–202
  two-party, 189–190
Design focus
  of business process management, 5–6
  of collaborative business processes, 5
  of service-oriented architecture, 3–5
  system-centric, 3–4
  why you should care about architecture, 19–20
Design patterns, TIBCO ActiveMatrix®
  basic interaction. *See* Interaction patterns
  event-driven interaction. *See* Event-driven interaction patterns
  mediation. *See* Mediation patterns
  overview of, 117
  policy framework. *See* ActiveMatrix policy framework
  system access. *See* System access patterns
  two-party coordination. *See* Two-party coordination patterns
Development processes, 36–37
Direct interaction
  with databases, 182–183
  with files, 183
  via ActiveMatrix-supported protocols, 178
  via non-ActiveMatrix-supported protocols, 181–182
Director role, BusinessEvents, 106–107
Disbursal authorization, process-pattern mapping, 24–25
Distributed application archive (DAA), 89–90, 151
Distributed systems, TIBCO products for, 63
Distributed transactions
  limitations of, 193–194
  messaging and, 193
  overview of, 190–191
  summary review, 198
  two-phase commit protocol, 191–193
Documentation
  advantages of, 27
  project architect responsibilities, 45
  reference architecture advantages, 33
Dynamic routing, mediation flow, 169–170

E

E-mails, In-Only pattern, 120
EJB binding type, 77
EJBs (enterprise java beans), 77
EMS (Enterprise Message Service). *See* TIBCO Enterprise Message Service™ (EMS)
EMS servers
  ActiveMatrix architecture pattern, 88–89
  administration, 69, 86
  EMS client libraries interacting with, 68–69
  overview of, 67–68
  as Service Bus element, 80

End-to-end business process, defined by project architect, 45
Enterprise architects
  overview of, 7–8
  responsibilities of, 47–49
  role of, 44
Enterprise java beans (EJBs), 77
Enterprise Message Service. *See* TIBCO Enterprise Message Service™ (EMS)
Environments, TIBCO ActiveMatrix®, 82–84
ETL (extract-transform-load) interactions, 176
Event collector, TIBCO ActiveMatrix® BPM, 97
Event-driven interaction patterns
  bridge semantics, 137–138
  defined, 134–135
  other sources of events, 139
  overview of, 133–134
  pub-sub architecture pattern, 134–135
  queue semantics, 135–136
  summary review, 139
  topic semantics, 137
Events. *See* TIBCO BusinessEvents™
Evolution strategy, enterprise architect defining, 47–48
Execution environment, AMX, 89
External reference checks, data validation, 203–205
External systems, accessing
  API interaction, 174
  combining API and Database interactions for, 177
  database interaction, 174–175
  direct interaction via ActiveMatrix-supported protocols, 177–178
  event recognition challenge, 175–177
  file-based interaction, 175
  indirect interaction via ActiveMatrix adapters, 179–181
  overview of, 173–174
  policies for, 150–153
  protocol-based interaction, 175
Extract-transform-load (ETL) interactions, 176

F

Failures, transaction. *See also* Breakdown detection, 193–194
Fault mapping, mediation flow design, 163–164
Fault tolerance, EMS servers for, 67–68
Feedback, improving breakdown detection with, 205–206
File Adapter, 183
File-based interactions
  accessing external systems, 175
  overview of, 183
File system folder structure, ActiveMatrix, 86–87
Fire-and-forget coordination pattern
  delegation pattern using, 189
  delegation with confirmation pattern using, 189–190
  multi-party, 200
  two-party, 186, 197
Flexibility of services, 213
folder structures, ActiveMatrix, 86–87
Front-end systems
  architecture step examining, 38–39
  placing validation in, 203–204
Functional organization, TIBCO ActiveMatrix® BPM, 96–97

G

Google Web Toolkit (GWT), OpenSpace BPM client, 97
Governance applications
  accessing external systems, 150–152
  implementing policy accessing LDAP, 153–155, 157
  summary review, 159
Granularity, service, 216

H

High-fanout message delivery, EMS, 69
HTTP (Hypertext Transfer Protocol), 63
HyperSQL database, Service Bus, 80–81

I

Identification, ATM withdraw cash process, 21–22
Implementation types
  ActiveMatrix, 75–76

content transformation, 165–166
In-Only message pattern, 123
Out-In message pattern, 131
In-Out message pattern, 125, 127
Out-Only message pattern, 130
SCA, 55–56, 59
Service Grid, 81–82
straight-wire mapping, 162
Implementing SOA (Brown), 38, 41
In-Only message exchange pattern
defined, 120
delegation with confirmation pattern, 190
example and implementation options, 122–123
summary review, 131
In-Out message exchange pattern
examining interactions of, 120
example and implementation options, 123–127
fire-and-forget coordination pattern using, 186
summary review, 131–132
Inbound to external system, 173–175, 177
Indirect interaction
with databases, 182–183
via ActiveMatrix adapters, 179–181
Input
ATM withdraw cash process, 21–22
mediation flow design for mapping, 163–164
Installation folder, ActiveMatrix, 86
Integration test step, 42
Interaction patterns
event-driven. *See* Event-driven interaction patterns
mediation. *See* Mediation patterns
newspaper case study example, 121–122
In-Only implementation, 122–123
In-Out implementation, 123–127
Out-In implementation, 130–131
Out-Only implementation, 127–130
overview, 120–121
overview of, 119
summary review, 131–132
system access. *See* System access patterns
TIBCO Architecture Fundamentals, 63
Interface
investment required for stability of, 215–216
mediation flow design, 163–164
Membership Validation Service, 220–221
services standardizing, 212
stability of SOA, 213
in system-centric design focus, 3–4
TIBCO ActiveMatrix® BPM browser-based, 97
Investment, interface stability, 215–216
Isolation, services benefiting, 213

J

JAAS, EMS supporting, 72
JACI, EMS supporting, 72
Java, EMS client library for, 68–69
Java implementation type
defined, 75
TIBCO ActiveMatrix® Service Grid, 81–82
TIBCO supporting, 56
JDBC interactions, 182, 187
JMS (Java Messaging Service). *See also* XML over JMS
binding type, 77
as communication mechanism, 63
conventional message delivery, 69–70
high-fanout message delivery, 69
pub-sub architecture pattern and, 135
queue semantics, 135–136
JMX commands, ActiveMatrix solution life cycle, 88–89

L

LDAP access policies, 150–157
Leadership, proactive architectural, 40
Libraries, EMS client, 68–69
Life cycle
solution, 88–91
TIBCO ActiveMatrix® Lifecycle Governance, 73–74
TIBCO BusinessEvents™ solution, 112–114
Logical environment structure, ActiveMatrix, 83, 89–90

M

Machine model, AMX execution environment, 89
Manager role, distributed transactions, 191
Mapping, process-pattern
  advantages of, 27
  after refinement, 226–228
  Aspect-Oriented Design and, 141–142
  ATM withdraw cash examples, 24–26
  documentation of, 27
  Membership Validation, 222–223
  overview of, 18–19
  reference architecture, 32–34
  reference architecture used as entire solution, 230–231
  reference architecture used as fragment of, 232–235
  solution architecture, 219
  why you should care about, 19–20
Mediation flow design interface, 163–164
Mediation Flow implementation type
  ActiveMatrix, 75
  content transformation, 165
  data augmentation, 166–167
  features, 164
  routing, 169
  as Service Bus element, 80
  straight-wire mediation, 162
  summary review, 172
Mediation patterns
  content transformation, 165–166
  data augmentation, 166–168
  flow capabilities and limitations, 170–171
  mediation flow design, 163–164
  overview of, 161
  routing, 168–170
  straight-wire mapping, 162
  summary review, 171
  use case: access control, 164
  use case: transport mapping, 164–165
Membership Validation Service example
  reference architecture used as entire solution, 231–235
  requirements, 220–221
  solution architecture, 221–223
Mentoring, enterprise architect role, 49
Message delivery transports, 69–72, 144
Messages
  distributed transactions and, 193
  event-drive interaction patterns. *See* Event-driven interaction patterns
  as pub-sub communications channel, 135
Migration strategy, enterprise architect defining, 47–48
Multi-party coordination patterns
  breakdown detection, 205–208
  data validation, 202–205
  delegation with confirmation, 201–202
  fire-and-forget, 200
  overview of, 199–200
  request-reply, 200–201, 222–223
  summary review, 207–208
Multicast message delivery, EMS, 70–72
Multiple message storage options, EMS, 72

N

Newspaper case study example
  In-Only implementation, 122–123
  In-Out implementation, 123–127
  Out-In implementation, 130–131
  Out-Only implementation, 127–130
  overview of, 121–122
Nodes. *See* ActiveMatrix nodes
Notifications
  delivered to multiple parties, 133–134
  requests vs., 133
  topic delivery semantics for, 137

O

OpenSpace client, 97
Operation semantics, 212–213
Optimization, TIBCO product suite for, 64
Organizational silos, business processes and, 35–36
OSGI Plugins, AMX execution environment, 89, 92
Out-In message exchange pattern
  asynchronous variation of, 131–132
  defined, 121

example and implementation options, 130–131
fire-and-forget coordination pattern, 186
Out-Only message exchange pattern
asynchronous variation of, 131–132
defined, 120–121
delegation with confirmation pattern, 190
example and implementation options, 127–131
Outbound from external system, 173–175, 177
Output mapping, mediation flow design, 163–164

P

Parallelism, implementing processes with, 23–24
Patterns
architecture. *See* Architecture patterns
design. *See* Design patterns, TIBCO ActiveMatrix®
event-driven interaction. *See* Event-driven interaction patterns
mediation. *See* Mediation patterns
system access. *See* System access patterns
two-party coordination. *See* Two-party coordination patterns
People, in scope of total architecture, 9–11
PEPs (policy enforcement points)
ActiveMatrix nodes, 78–79
direct interaction via ActiveMatrix-supported protocols, 178
overview of, 148
straight-wire mapping for access control, 164
Physical environment structure
administration organization of, 86
architecture pattern for, 88
overview of, 83–84
Platform neutrality, of services, 213
Point-of-view interfaces, system-centric design, 3–4
Policeman approach, architects avoiding, 40
Policies
concerns addressed by Service Bus, 143–144
framework. *See* ActiveMatrix policy framework
governing node behavior, 78
overview of, 144
Policy agent, 78, 80
Policy enforcement points. *See* PEPs (policy enforcement points)
Policy intents
overview of, 157–158
summary review, 159
Policy set templates. *See* Policy templates
Policy sets
applicability of, 148
associating with design elements, 148–150
implementing policy accessing LDAP, 153–154
overview of, 144–146
summary review, 158–159
Policy templates
accessing external system from policy set, 151
concerns addressed by ActiveMatrix Service Bus using, 143–144
overview of, 146–148
summary review, 158–159
Port types, in straight-wire mapping, 162
Practical evolution strategy, enterprise architect defining, 47–48
Problem-solving, with reference architecture, 33
Process-centric design
coordinating changes to multiple systems, 37
IT moving from system-centric to, 5–6
Process coordinator, third-party, 194–195
Process manager, 5–6, 96–97
Process models
ATM withdraw cash example, 21–24
check order status, 15
overview of, 13–16
process-pattern mapping, 18–19, 24–25

Process models (*continued*)
reference architecture, 30–31, 228–229
refining solution architecture, 224–225
why you should care about, 19–20
Process-pattern mapping. *See* Mapping, process-pattern
Product structure, TIBCO Enterprise Message Service™, 67–69
Product suites
BusinessEvents, 107–110
TIBCO, 63–65
TIBCO ActiveMatrix®, 73–74
Production BusinessEvents deployment example, 111–112
Project architects
enterprise architect role in training, 49
overview of, 7–8
responsibilities of, 45–46
role of, 44
Project charter, 40–42
Project teams, 48–49
Promoted reference, SCA, 57–58
Promoted service, SCA, 56–57
Protocols, Active-Matrix supported
accessing external systems, 175, 177–178
accessing external systems with non, 181–182
advantages, 182
fire-and-forget coordination using, 186
synchronous request-reply coordination using, 187
provides attribute, policy sets, 158
Pub-sub architecture pattern, 134–135

Q

QA (quality assurance), development process, 36
Quantification of business expectations, project charter, 40–41
Queue delivery semantics, event-driven interactions, 135–138

R

Reference architecture
applications of, 32–33
architecture pattern, 31–32
essential aspects of, 29–30
process model, 30–31
process-pattern mapping, 32
role between project and enterprise architects, 45–46, 48, 52
summary review, 33–34
using as entire solution, 228–231
using as fragment of solution, 231–235
References, SCA
associating policy sets with, 149–150
defining component type with, 58–59
deploying SCA designs on ActiveMatrix nodes, 92–94
overview of, 57–58
straight-wire mapping for mediation, 162
summary review, 60
Refinement process, solutions
overview of, 224–228
reference architecture as entire solution, 228–231
reference architecture as solution fragment, 232–235
Reliability, policy intents associated with, 158
Request-reply coordination pattern. *See also* In-Out message exchange pattern
delegation pattern using, 189
multi-party, 200–201
two-party, 187–188
Requests
bridge message delivery semantics for, 137–138
notifications vs., 133
queue delivery semantics for, 135–136
Resource managers, two-phase commit transactions, 191–193
Resource templates, ActiveMatrix solution life cycle, 90–91
Resource Templates dialog, accessing LDAP, 155
Results, ATM withdraw cash process, 21–22
Return on investment (ROI), services, 215–216

Reuse, of services, 213
Risks, quantifying business process, 41–42
ROI (return on investment), services, 215–216
Roles
- architect. *See* Design patterns, TIBCO ActiveMatrix®
- process coordinator, 194–195
- TIBCO BusinessEvents™, 106–107
- transaction manager, 191

Rollbacks, transaction, 193
Routing, mediation, 168–170, 172
Rules, ActiveMatrix policy, 144
Run-time environments, 82–84

S

SCA (service-component architecture)
- architectural decisions, 38–39
- business processes/organizational silos, 35–36
- components and composites, 55–56, 58–60
- deploying designs on ActiveMatrix nodes, 91–96
- design focus, 3–5
- example service design, 54–55
- implementation type, 59
- Out-Only pattern not well represented in, 128–130
- overview of, 53–54
- policy intents, 158
- policy sets, 144
- references, 57–58
- Service Bus as foundation of, 73–74
- Service Bus policy framework, 143–144
- services, 56–57
- summary review, 60

Schedule, project, 41–44
Scope, 7–11
Security, policy intents, 158
Select/Create a Policy Set dialog, 149–150
Self-consistency checks, data validation, 203
Sequencing activities, process models, 22–23
Servers. *See* EMS servers
Service Bus. *See* TIBCO ActiveMatrix® Service Bus
Service-component architecture. *See* SCA (service-component architecture)
Service Grid. *See* TIBCO ActiveMatrix® Service Grid
service-level agreements (SLAs)
- multi-party request-reply coordination, 200
- two-party request-reply coordination, 187

service-oriented architecture. *See* SOA (service-oriented architecture)
Service providers, for data validation, 204
Services
- accessing via two different transports, 164–165
- ATM withdraw cash example, 25–26
- benefits of, 213–214
- defined, 211
- granularity of, 216
- overview of, 211
- policy governing access to, 144
- policy sets associated with, 154–155
- practical evolution strategy for, 47–48
- project architect identifying, 46
- SCA, 56–59, 91–94
- situations warranting investment in, 215–216
- SOA approach to, 212–214
- straight-wire mapping and, 162
- summary review, 217
- traditional approach vs., 211–212

SLAs (service-level agreements)
- multi-party request-reply coordination, 200
- two-party request-reply coordination, 187

SOA (service-oriented architecture)
- approach to services, 212–213
- business processes in, 35–36
- design concept of, 3–4
- requiring service interface stability, 214
- SCA based on. *See* SCA (service-component architecture)
- service-centric design focus and, 4–5
- TIBCO product suite for, 64

SOAP
- fire-and-forget coordination using, 186
- protocol binding type, 77
- synchronous request-reply coordination using, 187

SOAP faults, 204, 222–223
SOAP over ActiveMatrix Virtualization, 124
SOAP over HTTP
- direct interaction via ActiveMatrix-supported protocols, 178
- interactions of, 119
- In-Only message pattern, 122–123
- In-Out message pattern, synchronous, 124

SOAP over JMS
- direct interaction via ActiveMatrix-supported protocols, 178
- interactions of, 119
- In-Only message pattern, 122–123
- In-Out message pattern, synchronous, 124

Software Architecture in Practice, Second Edition (Bass et al.), 29
Solution architecture
- architecture pattern refinement, 224–226
- mapping refinement, 226–227
- Membership Validation Service example, 221–222
- overview of, 219–220
- process model refinement, 224–225
- refinement, 224

Solution composite, TIBCO ActiveMatrix® BPM, 98
Solution life cycle, 88–91
Solutions
- adding refinement to, 224–228
- BusinessEvents life cycle, 112–114
- deploying BusinessEvents, 110–112
- Membership Validation Service example, 220–223
- overview of, 219
- reference architecture as fragment of, 231–235
- reference architecture defining, 228–231
- solution architecture. *See* Solution architecture
- summary review, 235
- TIBCO ActiveMatrix® BPM, 98

Spring implementation type
- defined, 75
- TIBCO ActiveMatrix® Service Grid, 81–82
- TIBCO supporting, 56

Standardized data semantics, 213
Straight-wire mapping
- mediation flow design, 163–164
- overview of, 162
- summary review, 171
- use case: access control, 164
- use case: transport mapping, 164–165

Succeeding with SOA (Brown), 42
Synchronous In-Out pattern, 124–125
Synchronous request-reply coordination, multi-party, 200–201
Synchronous request-reply coordination, two-party, 187
Syntactic validation, 203
System access patterns
- accessing external systems, 173–177
- database interactions, 182–183
- direct interaction via non-ActiveMatrix-supported protocols, 181–182
- direct interaction vs. ActiveMatrix-supported protocols, 177–178
- file interactions, 183
- general considerations, 182
- indirect interaction via ActiveMatrix adapters, 179–181
- overview of, 173
- summary review, 183–184

System-centric design
- accidental architecture based on, 7
- business process change using, 35–36
- development process, 36–37
- no longer sufficient for today's projects, 3–6

System Environment, 86
System Host, 86
System integration test step, 42
System Node, 86, 88–89
Systems, scope of total architecture, 9–11

T

- Target architecture, 47–48
- TAS (total architecture synthesis) methodology, 41
- TCT (TIBCO Configuration Tool), 86–87
- Technology, access
  - standardizing using services, 212–213
  - traditional vs. service approaches, 211–212
- Templates, policy set, 146–148
- Text messages, In-Only message patterns, 120
- Third-party asynchronous In-Out pattern, 126–127
- Third-party process coordinators, 194–195
- Third-party process monitoring, 206–207
- Threads, EMS server, 68
- TIBCO ActiveMatrix Adapter for Database, 176, 182–183
- TIBCO ActiveMatrix adapters
  - for Database, 176, 182–183
  - defined, 73–74
  - deploying implementations in, 76
  - for Files, 183
  - indirect interaction via, 179–181
  - other components playing role of, 181
  - solving event recognition using, 176–177
- TIBCO ActiveMatrix Business Works™, 73–76
- TIBCO ActiveMattrix® product suite, 73-74
- TIBCO ActiveMatrix® Administrator
  - accessing external system from policy set, 151
  - of ActiveMatrix nodes, 78
  - administration organization, 84–86
  - deployment and run-time management, 80–81
  - implementing policy accessing LDAP, 154–156
  - plugins for EMS, 69
  - solution life cycle, 88–91
- TIBCO ActiveMatrix® BPM
  - defined, 65, 73–74
  - functional organization, 96–97
  - overview of, 96
  - process coordinator role of, 195
  - solution deployment, 98
- TIBCO ActiveMatrix® Lifecycle Governance Framework, 73–74
- TIBCO ActiveMatrix® Service Bus
  - administration organization, 84–86
  - architecture patterns, 74–78
  - associating policy sets with design elements, 148–150
  - defined, 73
  - deploying SCA designs on ActiveMatrix nodes, 91–96
  - design patterns. *See* Design patterns, TIBCO ActiveMatrix®
  - file system folder structures, 86–87
  - implementing policy for accessing LDAP, 153–157
  - logical environments, 83
  - Mediation Flow implementation type in, 56, 75
  - overview of, 78–81
  - physical environments, 83–84
  - policies accessing external systems, 151–153
  - policy applicability, 148
  - policy enforcement points, 148
  - policy framework, 143–144
  - policy intents, 157–158
  - policy set templates, 146–148
  - policy sets, 144–146
  - references, 57–58
  - run-time environments, 82–83
  - Service Grid built on, 81
  - services, 56–57
  - solution life cycle, 88–91
  - summary review, 98–100
- TIBCO ActiveMatrix® Service Grid
  - defined, 64, 73–74
  - deploying implementations in, 75
  - overview of, 81–82
- TIBCO ActiveMatrix™ BusinessWorks™
  - architecture pattern example, 17
  - defined, 64
  - deploying implementations in, 75–76
  - direct interaction with databases, 182–183
  - direct interaction with files, 183
  - messaging and transactions in, 193
  - process coordinator role of, 195
  - taking role of adapter, 181

TIBCO Business Studio™, 56, 80
TIBCO Business Works™ Service Engine (BWSE), 76
TIBCO BusinessEvents™. *See also* Complex event processing, BusinessEvents
  director role, 106–107
  overview of, 101
  process coordinator role of, 195
  product suite, 107–110
  solution deployment, 110–112
  solution life cycle, 112–114
  solution role of complex event processor, 106
  summary review, 114–115
TIBCO Configuration Tool (TCT), 86–87
TIBCO Enterprise Message Service™ (EMS)
  conventional message delivery, 69–70
  defined, 63
  feature highlights, 72
  high-fanout message delivery, 69
  multicast message delivery, 70–72
  overview of, 67
  product structure, 67–69
  as pub-sub communications channel, 135
  queue delivery semantics, 136
  TIBCO ActiveMatrix® architecture patterns, 74
TIBCO product suite
  overview of, 63–65
TIBCO Rendezvous™, 72
TIBCO SmartSockets™, 72
TIBCO® Adapter SDK, 180–181
TIBCO™ General Interface, 97
Topic delivery semantics, event-driven interaction patterns, 137–138
Total architecture, scope of, 9–10
total architecture synthesis (TAS) methodology, 41
Traditional approach, to services, 211–212
Training, enterprise architect role, 49
Transactions
  implementing distributed, 190–194
  policy intents associated with, 158
Transport mapping, in straight-wire mapping, 164–165
Two-party coordination patterns
  architecture pattern for, 119
  compensating transactions, 195–197
  delegation, 188–189
  delegation with confirmation, 189–190
  distributed transactions, 190–194
  fire-and-forget coordination, 186
  overview of, 185
  request-reply coordination, 187–188
  summary review, 197–198
  third-party process coordinator, 194–195
Two-phase commit transactions
  approximating with compensation patterns, 195–196
  compensating transactions vs., 195
  implementing distributed transactions, 191–193

U

UML (Unified Modeling Language) notations, using, 54
Understanding SCA (Marino and Rowley), 53
Unified Modeling Language (UML) notations, using, 54

V

Validation, data., 202–204
Virtualization, policy intents and, 158
Vision, architectural
  communicating, 50–51
  creating reference architectures for, 45–46
  enterprise architect's role in, 44, 47–49
  project architect's role in, 44

W

WebApp implementation type, 56, 75, 81–82
Work manager, TIBCO ActiveMatrix® BPM, 96–97
Work patterns, project architect responsibilities, 46
WorkSpace client, TIBCO ActiveMatrix® BPM, 97

WSDL portType, SCA services and references, 56–57, 60
WSS Consumer, Service Bus policy templates for, 148
WSS Provider, Service Bus policy templates for, 148

X

XML files, defining policy sets with, 144
XML over JMS
  direct interaction via ActiveMatrix-supported protocols, 178
  fire-and-forget coordination, 187–188
  interactions of, 119, 123
  Out-In message pattern, 130–131
  In-Out message pattern, asynchronous, 127
  In-Out message pattern, synchronous, 124
  Out-Only message pattern, 127–129
  request-reply coordination, synchronous, 187
XPath Route, mediation flow, 169