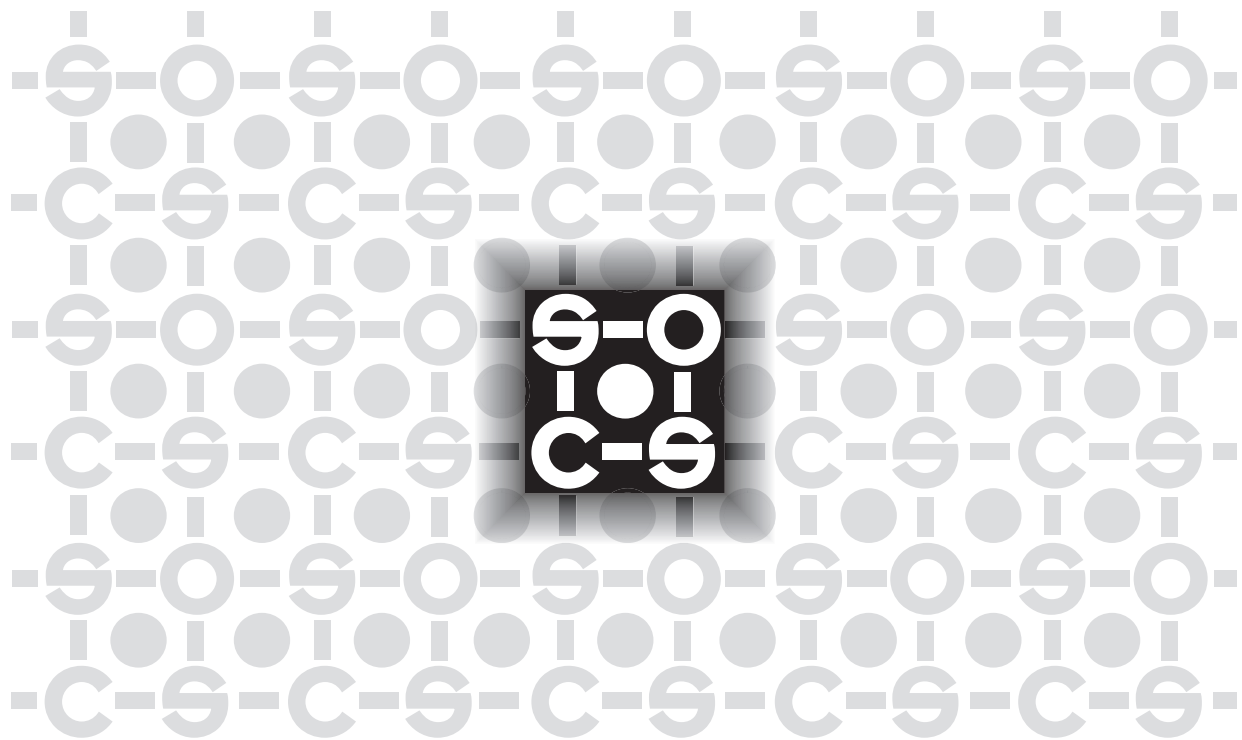


SOA Design Patterns



The Prentice Hall Service-Oriented Computing Series from Thomas Erl aims to provide the IT industry with a consistent level of unbiased, practical, and comprehensive guidance and instruction in the areas of service-oriented architecture, service-orientation, and the expanding landscape that is shaping the real-world service-oriented computing platform.

For more information, visit www.soabooks.com.

SOA Design Patterns

Thomas Erl

(with additional contributors)



PRENTICE HALL

UPPER SADDLE RIVER, NJ • BOSTON • INDIANAPOLIS • SAN FRANCISCO

NEW YORK • TORONTO • MONTREAL • LONDON • MUNICH • PARIS • MADRID

CAPETOWN • SYDNEY • TOKYO • SINGAPORE • MEXICO CITY

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

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

Library of Congress Cataloging-in-Publication Data:

Erl, Thomas.

SOA design patterns / Thomas Erl. — 1st ed.

p. cm.

ISBN 0-13-613516-1 (hardback : alk. paper) 1. Web services.

2. Computer architecture. 3. Software patterns. 4. System design. I. Title.

TK5105.88813.E735 2008

006.7—dc22

2008040488

Copyright © 2009 SOA Systems Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

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

ISBN-13: 978-0-13-613516-6

ISBN-10: 0-13-613516-1

Text printed in the United States on recycled paper at R.R. Donnelley in Crawfordsville, Indiana.

First printing December 2008

The following patterns: Exception Shielding, Threat Screening, Trusted Subsystem, Service Perimeter Guard, Data Confidentiality, Data Origin Authentication, Direct Authentication, Brokered Authentication are courtesy of the Microsoft Patterns & Practices team. For more information please visit <http://msdn.microsoft.com/practices>. These patterns were originally developed by Jason Hogg, Frederick Chong, Dwayne Taylor, Lonnie Wall, Paul Slater, Tom Hollander, Wojtek Kozaczynski, Don Smith, Larry Brader, Sajjas Nasir Imran, Pablo Cibraro, Nelly Delgado and Ward Cunningham

Editor-in-Chief

Mark L. Taub

Managing Editor

Kristy Hart

Copy Editor

Language Logistics

Indexer

Cheryl Lenser

Proofreader

Williams Woods
Publishing

Composition

Jake McFarland
Bumpy Design

Graphics

Zuzana Cappova
Tami Young
Spencer Fruhling

Photos

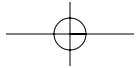
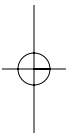
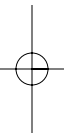
Thomas Erl

Cover Design

Thomas Erl

To the SOA pioneers that blazed the trail we now so freely base our roadmaps on, and to the SOA community that helped me refine the wisdom of the pioneers into this catalog of patterns.

- Thomas Erl



Contents

Foreword	xxxvii
-----------------	---------------

CHAPTER 1: Introduction	1
--------------------------------	----------

1.1 Objectives of this Book	4
1.2 Who this Book is For	4
1.3 What this Book Does Not Cover	4
Topics Covered by Other Books	4
Web Service and REST Service Design Patterns	5
SOA Standardization Efforts	5
1.4 Recommended Reading	6
1.5 How this Book is Organized	7
Part I: Fundamentals	8
Part II: Service Inventory Design Patterns	8
Part III: Service Design Patterns	8
Part IV: Service Composition Design Patterns	9
Part V: Supplemental	10
Part VI: Appendices	10
1.6 Symbols, Figures, Style Conventions	11
Symbol Legend	11
How Color is Used	11
Data Flow and Directionality Conventions	11
Pattern Documentation Conventions	11
1.7 Additional Information	11
Updates, Errata, and Resources (www.soabooks.com)	11
Visio Stencil (www.soabooks.com)	12

Community Patterns Site (www.soapatterns.org)	12
Master Glossary (www.soaglossary.com)	12
Supplementary Posters (www.soaposters.com)	12
The SOA Magazine (www.soamag.com)	12
Referenced Specifications (www.soaspecs.com)	12
Notification Service	13
Contact the Author	13

CHAPTER 2: Case Study Background 15

2.1 Case #1 Background: Cutit Saws Ltd.	17
History	18
Technical Infrastructure and Automation Environment	18
Business Goals and Obstacles	18
2.2 Case #2 Background: Alleywood Lumber Company	19
History	19
Technical Infrastructure and Automation Environment	20
Business Goals and Obstacles	20
2.3 Case #3 Background: Forestry Regulatory Commission (FRC)	21
History	21
Technical Infrastructure and Automation Environment	21
Business Goals and Obstacles	22

PART I: FUNDAMENTALS

CHAPTER 3: Basic Terms and Concepts 25

Purpose of this Introductory Chapter	26
3.1 Architecture Fundamentals	26
A Classic Analogy for Architecture and Infrastructure	27
Technology Architecture	27
Technology Infrastructure	30
Software Program	32
Relationship to Design Framework	33

Contents

xv

3.2 Service-Oriented Computing Fundamentals.	35
Service-Oriented Computing	35
Service-Orientation	36
Service-Oriented Architecture (SOA)	37
Service	37
Service Capability	38
Service Consumer	38
Service Composition	40
Service Inventory	42
Service-Oriented Analysis	43
Service Candidate	44
3.3 Service Implementation Mediums	44
Services as Components	45
Services as Web Services	45
REST Services	46

CHAPTER 4: The Architecture of Service-Orientation . . 47

Purpose of this Introductory Chapter	48
4.1 The Method of Service-Orientation	48
Principles of Service-Orientation	48
Strategic Goals of Service-Oriented Computing	51
4.2 The Four Characteristics of SOA	52
Business-Driven	53
Vendor-Neutral	54
Enterprise-Centric	58
Composition-Centric	59
4.3 The Four Common Types of SOA	61
Service Architecture	62
<i>Information Hiding</i>	64
<i>Design Standards</i>	64
<i>Service Contracts</i>	65
<i>Service Agents</i>	67
<i>Service Capabilities</i>	68
Service Composition Architecture	68
<i>Nested Compositions</i>	72
<i>Task Services and Alternative Compositions</i>	73
<i>Compositions and Infrastructure</i>	74

Service Inventory Architecture	74
Service-Oriented Enterprise Architecture	76
Architecture Types and Scope	77
Architecture Types and Inheritance	77
Other Forms of Service-Oriented Architecture	78
<i>Inter-Business Service Architecture</i>	78
<i>Service-Oriented Community Architecture</i>	78
4.4 The End Result of Service-Orientation	79

CHAPTER 5: Understanding SOA Design Patterns 85

Purpose of this Introductory Chapter	86
5.1 Fundamental Terminology	86
What's a Design Pattern?	86
What's a Compound Pattern?	88
What's a Design Pattern Language?	88
What's a Design Pattern Catalog?	89
5.2 Historical Influences	89
Alexander's Pattern Language	90
Object-Oriented Patterns	91
Software Architecture Patterns	92
Enterprise Application Architecture Patterns	93
EAI Patterns	93
SOA Patterns	94
5.3 Pattern Notation	95
Pattern Symbols	95
Pattern Figures	96
<i>Pattern Application Sequence Figures</i>	96
<i>Pattern Relationship Figures</i>	96
<i>Compound Pattern Hierarchy Figures</i>	99
Capitalization	100
Page Number References	100
5.4 Pattern Profiles	100
Requirement	101
Icon	101
Summary	102
Problem	102
Solution	102

Contents

xvii

Application	103
Impacts	103
Relationships	103
Case Study Example	103
5.5 Patterns with Common Characteristics	104
Canonical Patterns	104
Centralization Patterns	105
5.6 Key Design Considerations	106
“Enterprise” vs. “Enterprise-wide”	106
Design Patterns and Design Principles	106
Design Patterns and Design Granularity	107
Measures of Design Pattern Application	108

PART II: SERVICE INVENTORY DESIGN PATTERNS

CHAPTER 6: Foundational Inventory Patterns 111

How Inventory Design Patterns Relate to SOA Design	
Characteristics	113
How Foundational Inventory and Service Patterns Relate	114
How Case Studies are Used in this Chapter	114
6.1 Inventory Boundary Patterns	114
Enterprise Inventory	116
Problem	116
Solution	117
Application	118
Impacts	120
Relationships	121
Case Study Example	122
Domain Inventory	123
Problem	123
Solution	124
Application	125
Impacts	126
Relationships	127
Case Study Example	128

6.2 Inventory Structure Patterns	130
Service Normalization	131
Problem	131
Solution	132
Application	132
Impacts	133
Relationships	133
Case Study Example	135
Logic Centralization	136
Problem	136
Solution	137
Application	137
Impacts	139
Relationships	140
Case Study Example	142
Service Layers	143
Problem	143
Solution	144
Application	145
Impacts	147
Relationships	147
Case Study Example	148
6.3 Inventory Standardization Patterns	149
Canonical Protocol	150
Problem	151
Solution	152
Application	153
Impacts	155
Relationships	155
Case Study Example	157
Canonical Schema	158
Problem	158
Solution	159
Application	159
Impacts	159
Relationships	160
Case Study Example	161

CHAPTER 7: Logical Inventory Layer Patterns 163

Combining Layers	164
Business Logic and Utility Logic	166
Agnostic Logic and Non-Agnostic Logic	166
Service Layers and Logic Types	167
Utility Abstraction	168
Problem	168
Solution	169
Application	170
Impacts	171
Relationships	171
Case Study Example	173
Entity Abstraction	175
Problem	175
Solution	176
Application	176
Impacts	178
Relationships	178
Case Study Example	180
Process Abstraction	182
Problem	182
Solution	183
Application	184
Impacts	185
Relationships	185
Case Study Example	187

CHAPTER 8: Inventory Centralization Patterns 191

Process Centralization	193
Problem	193
Solution	194
Application	195
Impacts	196
Relationships	197
Case Study Example	198

Schema Centralization	200
Problem	200
Solution	201
Application	202
Impacts	202
Relationships	203
Case Study Example	203
Policy Centralization	207
Problems	207
Solution	208
Application	209
Impacts	210
Relationships	211
Case Study Example	213
Rules Centralization	216
Problem	216
Solution	217
Application	217
Impacts	218
Relationships	219
Case Study Example	222
 CHAPTER 9: Inventory Implementation Patterns	 225
Dual Protocols	227
Problem	228
Solution	228
Application	228
Impacts	233
Relationships	234
Case Study Example	235
Canonical Resources	237
Problem	238
Solution	238
Application	239
Impacts	239

Contents

xxi

Relationships	239
Case Study Example	241
State Repository	242
Problem	242
Solution	243
Application	244
Impacts	244
Relationships	244
Case Study Example	246
Stateful Services	248
Problem	248
Solution	248
Application	250
Impacts	250
Relationships	250
Case Study Example	251
Service Grid	254
Problem	254
Solution	255
Application	256
Impacts	257
Relationships	258
Case Study Example	259
Inventory Endpoint	260
Problem	260
Solution	261
Application	262
Impacts	263
Relationships	263
Case Study Example	265
Cross-Domain Utility Layer	267
Problem	267
Solution	268
Application	269
Impacts	269
Relationships	270
Case Study Example	270

CHAPTER 10: Inventory Governance Patterns 273

Canonical Expression	275
Problem	275
Solution	275
Application	276
Impacts	277
Relationships	278
Case Study Example	279
Metadata Centralization	280
Problem	280
Solution	281
Application	282
Impacts	283
Relationships	283
Case Study Example	284
Canonical Versioning	286
Problem	286
Solution	287
Application	287
Impacts	288
Relationships	288
Case Study Example	290

PART III: SERVICE DESIGN PATTERNS

CHAPTER 11: Foundational Service Patterns 295

Case Study Background	297
11.1 Service Identification Patterns	299
Functional Decomposition	300
Problem	300
Solution	301
Application	302
Impacts	302
Relationships	303
Case Study Example	303

Service Encapsulation	305
Problem	305
Solution	306
Application	307
Impacts	309
Relationships	309
Case Study Example	310
11.2 Service Definition Patterns	311
Agnostic Context	312
Problem	313
Solution	314
Application	315
Impacts	315
Relationships	316
Case Study Example	317
Non-Agnostic Context	319
Problem	319
Solution	320
Application	321
Impacts	322
Relationships	322
Case Study Example	323
Agnostic Capability	324
Problem	324
Solution	325
Application	326
Impacts	327
Relationships	327
Case Study Example	328
 CHAPTER 12: Service Implementation Patterns	 331
Service Façade	333
Problem	333
Solution	334
Application	335

Impacts	341
Relationships	342
Case Study Example	343
Redundant Implementation	345
Problem	345
Solution	346
Application	346
Impacts	347
Relationships	348
Case Study Example	349
Service Data Replication	350
Problem	350
Solution	352
Application	353
Impacts	353
Relationships	353
Case Study Example	354
Partial State Deferral	356
Problem	356
Solution	357
Application	358
Impacts	359
Relationships	359
Case Study Example	360
Partial Validation	362
Problem	362
Solution	363
Application	364
Impacts	364
Relationships	364
Case Study Example	365
UI Mediator	366
Problem	366
Solution	367
Application	368

Impacts	369
Relationships	370
Case Study Example	370

CHAPTER 13: Service Security Patterns 373

Case Study background	374
---------------------------------	-----

Exception Shielding 376

Problem	376
Solution	377
Application	378
Impacts	379
Relationships	379
Case Study Example	380

Message Screening 381

Problem	381
Solution	382
Application	382
Impacts	384
Relationships	385
Case Study Example	385

Trusted Subsystem 387

Problem	387
Solution	388
Application	388
Impacts	391
Relationships	391
Case Study Example	392

Service Perimeter Guard 394

Problem	394
Solution	395
Application	395
Impacts	396
Relationships	396
Case Study Example	397

CHAPTER 14: Service Contract Design Patterns 399

Decoupled Contract	401
Problem	401
Solution	402
Application	403
Impacts	405
Relationships	405
Case Study Example	407
Contract Centralization	409
Problem	409
Solution	410
Application	410
Impacts	411
Relationships	411
Case Study Example	413
Contract Denormalization	414
Problem	414
Solution	415
Application	416
Impacts	417
Relationships	417
Case Study Example	418
Concurrent Contracts	421
Problem	421
Solution	422
Application	423
Impacts	425
Relationships	425
Case Study Example	426
Validation Abstraction	429
Problem	429
Solution	430
Application	431
Impacts	432
Relationships	432
Case Study Example	433

Chapter 15: Legacy Encapsulation Patterns 439**Legacy Wrapper 441**

Problem 441

Solution 442

Application 443

Impacts 444

Relationships 444

Case Study Example 446

Multi-Channel Endpoint 451

Problem 451

Solution 452

Application 453

Impacts 454

Relationships 454

Case Study Example 456

File Gateway 457

Problem 457

Solution 458

Application 458

Impacts 459

Relationships 460

Case Study Example 461

CHAPTER 16: Service Governance Patterns 463**Compatible Change 465**

Problem 465

Solution 466

Application 466

Impacts 469

Relationships 469

Case Study Example 470

Version Identification 472

Problem 472

Solution 473

Application 473

Impacts	474
Relationships	474
Case Study Example	475
Termination Notification	478
Problem	478
Solution	479
Application	480
Impacts	480
Relationships	481
Case Study Example	481
Service Refactoring	484
Problem	484
Solution	485
Application	485
Impacts	486
Relationships	486
Case Study Example	488
Service Decomposition	489
Problem	489
Solution	491
Application	492
Impacts	492
Relationships	494
Case Study Example	495
Proxy Capability	497
Problem	497
Solution	498
Application	498
Impacts	500
Relationships	500
Case Study Example	501
Decomposed Capability	504
Problem	504
Solution	506
Application	507
Impacts	507

Relationships	508
Case Study Example	508
Distributed Capability	510
Problem	510
Solution	511
Application	512
Impacts	513
Relationships	513
Case Study Example	514

PART IV: SERVICE COMPOSITION DESIGN PATTERNS

CHAPTER 17: Capability Composition Patterns 519

Capability Composition	521
Problem	521
Solution	521
Application	523
Impacts	523
Relationships	523
Case Study Example	524
Capability Recomposition	526
Problem	526
Solution	527
Application	527
Impacts	527
Relationships	529
Case Study Example	530

CHAPTER 18: Service Messaging Patterns 531

Service Messaging	533
Problem	533
Solution	533
Application	534
Impacts	534

xxx

Contents

Relationships	535
Case Study Example	536
Messaging Metadata	538
Problem	538
Solution	538
Application	539
Impacts	540
Relationships	541
Case Study Example	542
Service Agent	543
Problem	543
Solution	544
Application	544
Impacts	546
Relationships	546
Case Study Example	548
Intermediate Routing	549
Problem	549
Solution	551
Application	552
Impacts	553
Relationships	553
Case Study Example	556
State Messaging	557
Problem	557
Solution	558
Application	560
Impacts	561
Relationships	561
Case Study Example	562
Service Callback	566
Problem	566
Solution	568
Application	568
Impacts	570

Contents

xxxi

Relationships	570
Case Study Example	571
Service Instance Routing	574
Problem	574
Solution	576
Application	576
Impacts	578
Relationships	578
Case Study Example	579
Asynchronous Queuing	582
Problem	582
Solution	584
Application	584
Impacts	587
Relationships	588
Case Study Example	589
Reliable Messaging	592
Problem	592
Solution	593
Application	593
Impacts	594
Relationships	595
Case Study Example	596
Event-Driven Messaging	599
Problem	599
Solution	600
Application	602
Impacts	602
Relationships	602
Case Study Example	604
 CHAPTER 19: Composition Implementation Patterns . .	 605
Agnostic Sub-Controller	607
Problem	607
Solution	608

Application	610
Impacts	610
Relationships	610
Case Study Example	612
Composition Autonomy	616
Problem	616
Solution	618
Application	619
Impacts	619
Relationships	620
Case Study Example	620
Atomic Service Transaction	623
Problem	623
Solution	624
Application	626
Impacts	626
Relationships	628
Case Study Example	629
Compensating Service Transaction	631
Problem	631
Solution	633
Application	633
Impacts	635
Relationships	635
Case Study Example	636
 CHAPTER 20: Service Interaction Security Patterns . .	 639
Data Confidentiality	641
Problem	641
Solution	643
Application	643
Impacts	644
Relationships	645
Case Study Example	646

Data Origin Authentication	649
Problem	649
Solution	650
Application	651
Impacts	652
Relationships	653
Case Study Example	653
Direct Authentication	656
Problem	656
Solution	657
Application	657
Impacts	658
Relationships	659
Case Study Example	660
Brokered Authentication	661
Problem	661
Solution	662
Application	663
Impacts	665
Relationships	665
Case Study Example	666
 CHAPTER 21: Transformation Patterns	 669
Data Model Transformation	671
Problem	671
Solution	672
Application	673
Impacts	674
Relationships	674
Case Study Example	677
Data Format Transformation	681
Problem	681
Solution	681
Application	683

Impacts	683
Relationships	683
Case Study Example	685
Protocol Bridging	687
Problem	687
Solution	688
Application	688
Impacts	690
Relationships	690
Case Study Example	692

PART V: SUPPLEMENTAL

CHAPTER 22: Common Compound Design Patterns . . . 697

“Compound” vs. “Composite”	698
Compound Patterns and Pattern Relationships	698
Joint Application vs. Coexistent Application	699
Compound Patterns and Pattern Granularity	700
Orchestration	701
Enterprise Service Bus	704
Service Broker	707
Canonical Schema Bus	709
Official Endpoint	711
Federated Endpoint Layer	713
Three-Layer Inventory	715

CHAPTER 23: Strategic Architecture Considerations . . 717

Increased Federation	718
Increased Intrinsic Interoperability	721
Increased Vendor Diversification Options	723

Contents

xxxv

Increased Business and Technology Alignment	725
Increased ROI	727
Increased Organizational Agility	728
Reduced IT Burden	729

**CHAPTER 24: Principles and Patterns at the
U.S. Department of Defense 731**

The Business Operating Environment (BOE)	733
Principles, Patterns, and the BOE	734
Incorporation of Information Assurance (IA)	736
Adherence to Standards	736
Data Visibility, Accessibility, and Understandability to Support Decision Makers	736
Loosely Coupled Services	736
Authoritative Sources of Trusted Data	737
Metadata-Driven Framework for Separation from Technical Details	737
Support Use of Open Source Software	738
Emphasize Use of Service-Enabled Commercial Off-the-Shelf (COTS) Software	738
Participation in the DoD Enterprise	738
Support Mobility — Users & Devices	738
The Future of SOA and the DoD	739
SOADoD.org	739

PART VI: APPENDICES

APPENDIX A: Case Study Conclusion 743

Cutit Saws Ltd.	744
Alleywood Lumber Company	744
Forestry Regulatory Commission (FRC)	745

APPENDIX B: Candidate Patterns 747**APPENDIX C: Principles of Service-Orientation 749**

Standardized Service Contract	751
Service Loose Coupling	753
Service Abstraction	755
Service Reusability	756
Service Autonomy	758
Service Statelessness	760
Service Discoverability	762
Service Composability	764

APPENDIX D: Patterns and Principles**Cross-Reference 767****APPENDIX E: Patterns and Architecture Types****Cross-Reference 775****About the Author 783****About the Contributors 784****Index of Patterns 791****Index 795**

Foreword

The entire history of software engineering can be characterized as one of rising levels of abstraction. We see this in our languages, our tools, our platforms, and our methods. Indeed, abstraction is the primary way that we as humans attend to complexity—and software-intensive systems are among the most complex artifacts ever created.

I would also observe that one of the most important advances in software engineering over the past two decades has been the practice of patterns. Patterns are yet another example of this rise in abstraction: A pattern specifies a common solution to a common problem in the form of a society of components that collaborate with one another. Influenced by the writings of Christopher Alexander, Kent Beck and Ward Cunningham began to codify various design patterns from their experience with Smalltalk. Growing slowly but steadily, these concepts began to gain traction among other developers. The publication of the seminal book *Design Patterns* by Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm marked the introduction of these ideas to the mainstream. The subsequent activities of the Hillside Group provided a forum for this growing community, yielding a very vibrant literature and practice. Now the practice of patterns is very much mainstream: Every well-structured software-intensive system tends to be full of patterns (whether their architects name them intentionally or not).

The emerging dominant architectural style for many enterprise systems is that of a service-oriented architecture, a style that at its core is essentially a message passing architecture. However, therein are many patterns that work (and anti-patterns that should be avoided).

Thomas' work is therefore the right book at the right time. He really groks the nature of SOA systems: There are many hard design decisions to be made, ranging from data-orientation to the problems of legacy integration and even security. Thomas offers wise counsel on each of these issues and many more, all in the language of design patterns. There are many things I like about this work. It's comprehensive. It's written in a very accessible

xxxviii

Foreword

pattern language. It offers patterns that play well with one another. Finally, Thomas covers not just the technical details, but also sets these patterns in the context of economic and other considerations.

SOA Design Patterns is an important contribution to the literature and practice of building and delivering quality software-intensive systems.

—Grady Booch, *IBM Fellow*
September, 2008

Acknowledgments

This book was in development for over three years, a good portion of which was dedicated to external reviews. Patterns were subjected to three review cycles that spanned a period of over twelve months and involved over 200 IT professionals. Pre-release galley of my first and second manuscript drafts were printed and shipped to SOA experts and patterns experts around the world. Additionally, I had the full manuscript published at SOAPatterns.org for an open industry review. Even though these review phases added much time and effort to the development of this book, they ultimately elevated the quality of this work by a significant margin.

Special thanks to Prentice Hall for their patience and support throughout the book development process. Specifically, I'd like to thank Kristy Hart and Jake McFarland for their tremendous production efforts and tireless commitment to achieving printed perfection, Mark Taub who stood by this book project through a whirlwind of changes, reviews, more changes, extensions, and delays, Stephane Nakib and Heather Fox for their on-going guidance, and Eric Miller for his assistance with publishing the online review version of the first manuscript draft. I am fortunate to be working with the best publishing team in the industry.

Special thanks also to Herbjörn Wilhelmsen, Martin Fowler, Ralph Johnson, Bobby Woolf, Grady Booch, Gregor Hohpe, Baptist Eggen, Dragos Manolescu, Frank Buschmann, Wendell Ocasio, and Kevin Davis for their guidance and uninhibited feedback throughout the review cycles.

My thanks and gratitude to the following reviewers that participated in one or more of the manuscript reviews (in alphabetical order by last name):

Mohamad Afshar, Oracle

Sanjay Agara, Wipro

Stephen Bennett, Oracle

Steve Birkel, Intel

Brandon Bohling, Intel

xl**Acknowledgments**

Grady Booch, IBM
Bryan Brew, Booz Allen Hamilton
Victor Brown, CMGC
Frank Buschmann, Siemens
Enrique G. Castro-Leon, Intel
Peter Chang, Lawrence Technical University
Jason “AJ” Comfort, Booz Allen Hamilton
John Crupi, JackBe
Veronica Gacitua Decar, Dublin City University
Ed Dodds, Conmergence
Kevin P. Davis, PhD
Dominic Duggan, Stevens Institute of Technology
Baptist Eggen, Dutch Department of Defense
Steve Elston, Microsoft
Dale Ferrario, Sun Microsystems
Martin Fowler, ThoughtWorks
Pierre Fricke, Red Hat
Chuck Georgo, Public Safety and National Security
Larry Gloss, Information Manufacturing
Al Gough, CACI International Inc.
Daniel Gross, University of Toronto
Robert John Hathaway III, SOA Object Systems
William M. Hegarty, ThoughtWorks
Gregor Hohpe, Google
Ralph Johnson, UIUC
James Kinneavy, University of California
Robert Laird, IBM
Doug Lea, Oswego State University of New York
Canyang Kevin Liu, SAP
Terry Lottes, Northrop Grumman Mission Systems
Chris Madrid, Microsoft
Anne Thomas Manes, Burton Group

Acknowledgments

xli

Dragos Manolescu, Microsoft
Steven Martin, Microsoft
Joe McKendrick
J.D. Meier, Microsoft
David Michalowicz, MITRE Corporation
Per Vonge Nielsen, Microsoft
Wendell Ocasio, DoD Military Health Systems, Agilex Technologies
Philipp Offermann, University of Berlin
Dmitri Ossipov, Microsoft
Prasen Palvakar, Oracle
Parviz Peiravi, Intel
Nishit Rao, Oracle
Ian Robinson, ThoughtWorks
Richard Van Schelven, Ericsson
Shakti Sharma, Sysco Corp
Don Smith, Microsoft
Michael Sor, Booz Allen Hamilton
John Sparks, Western Southern Life
Sona Srinivasan, CISCO
Linda Terlouw, Ordina
Phil Thomas, IBM
Steve Vinoski, IEEE
Herbjörn Wilhelmsen, Objectware
Peter B. Woodhull, Modus21
Bobby Woolf, IBM
Farzin Yashar, IBM
Markus Zirn, Oracle
Olaf Zimmermann, IBM

There were many more individuals who directly or indirectly supported this effort. Amidst the flurry of correspondence over the past three years, I was unable to keep track of all participants. If you were part of the SOA design patterns project and you don't see your name on this list, then do contact me via www.thomaserl.com.

Contributors

In alphabetical order by last name:

Larry Brader

David Chappell, Oracle

Frederick Chong

Pablo Cibraro, Lagash Systems SA

Ward Cunningham

Nelly Delgado, Microsoft

Florent Georges

Charles Stacy Harris, Microsoft

Kelvin Henney, Curbralan

Jason Hogg, Microsoft

Tom Hollander

Anish Karmarkar, Oracle

Sajjas Nasir Imran, Infosys

Berthold Maier, Oracle

Hajo Normann, EDS

Wojtek Kozaczynski

Mark Little, Red Hat

Brian Lokhorst, Dutch Tax Office

Brian Loesgen, Neudesic

Matt Long, Microsoft

Contributors

xliii

David Orchard, Oracle

Thomas Rischbeck, IPT

Chris Riley, SOA Systems

Satadru Roy, Sun Microsystems

Arnaud Simon, Red Hat

Paul Slater, Wadeware

Don Smith

Sharon Smith, Microsoft

Dwayne Taylor

Tina Tech

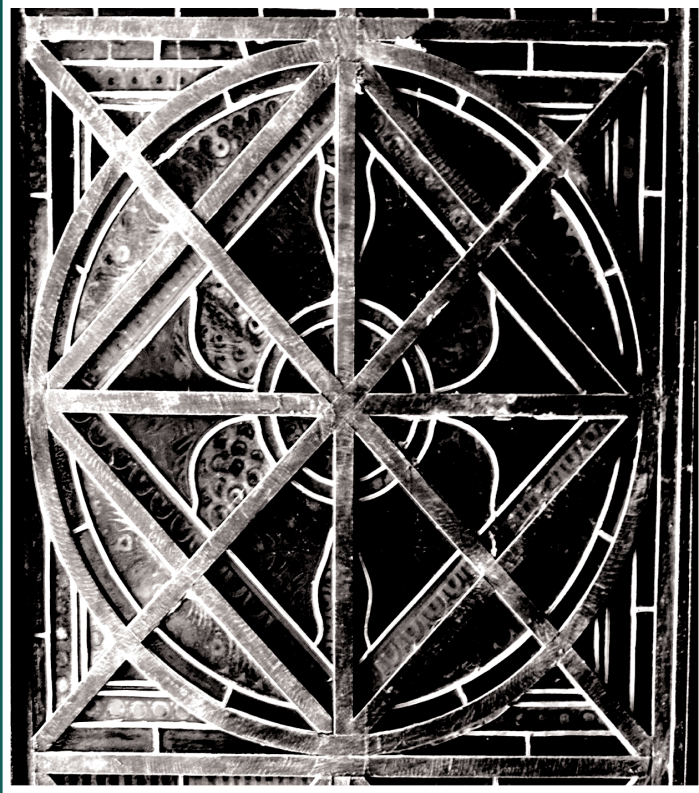
Bernd Trops, SOPER GmbH

Clemens Utschig-Utschig, Oracle

Lonnie Wall, RDA Corporation

Torsten Winterberg, Oracle

Dennis Wisnosky, U.S. Department of Defense



Chapter 5

Understanding SOA Design Patterns

- 5.1 Fundamental Terminology
- 5.2 Historical Influences
- 5.3 Pattern Notation
- 5.4 Pattern Profiles
- 5.5 Patterns with Common Characteristics
- 5.6 Key Design Considerations

The first step to forming an effective working relationship with SOA design patterns is attaining a sound comfort level with pattern-related terminology and notation. This provides us with the knowledge required to navigate through the upcoming chapters with insight as to how the patterns can be applied individually and in various combinations.

Purpose of this Introductory Chapter

This important chapter covers these fundamental topics and further describes how design pattern descriptions are organized into standardized profiles. The remaining sections single out specific pattern types and discuss some common design considerations.

5.1 Fundamental Terminology

What's a Design Pattern?

The simplest way to describe a pattern is that it provides a proven solution to a common problem individually documented in a consistent format and usually as part of a larger collection.

The notion of a pattern is already a fundamental part of everyday life. Without acknowledging it each time, we naturally use proven solutions to solve common problems each day. Patterns in the IT world that revolve around the design of automated systems are referred to as *design patterns*.

Design patterns are helpful because they:

- represent field-tested solutions to common design problems
- organize design intelligence into a standardized and easily “referencable” format
- are generally repeatable by most IT professionals involved with design
- can be used to ensure consistency in how systems are designed and built
- can become the basis for design standards
- are usually flexible and optional (and openly document the impacts of their application and even suggest alternative approaches)

5.1 Fundamental Terminology

87

- can be used as educational aids by documenting specific aspects of system design (regardless of whether they are applied)
- can sometimes be applied prior and subsequent to the implementation of a system
- can be supported via the application of other design patterns that are part of the same collection

Furthermore, because the solutions provided by design patterns are proven, their consistent application tends to naturally improve the quality of system designs.

Let's provide a simple (non SOA-related) example of a design pattern that addresses a user interface design problem:

Problem: How can users be limited to entering one value of a set of predefined values into a form field?

Solution: Use a drop-down list populated with the predefined values as the input field.

What this example also highlights is the fact that the solution provided by a given pattern may not necessarily represent the only suitable solution for that problem. In fact, there can be multiple patterns that provide alternative solutions for the same problem. Each solution will have its own requirements and consequences, and it is up to the practitioner to choose.

In the previous example, a different solution to the stated problem would be to use a list-box instead of a drop-down list. This alternative would form the basis of a separate design pattern description. The user-interface designer can study and compare both patterns to learn about the benefits and trade-offs of each. A drop-down list, for instance, takes up less space than a list box but requires that a user always perform a separate action to access the list. Because a list box can display more field lines at the same time, the user may have an easier time locating the desired value.

NOTE

Even though design patterns provide proven design solutions, their mere use cannot guarantee that design problems are always solved as required. Many factors weigh in to the ultimate success of using a design pattern, including constraints imposed by the implementation environment, competency of the practitioners, diverging business requirements, and so on. All of these represent aspects that affect the extent to which a pattern can be successfully applied.

What's a Compound Pattern?

A compound pattern is a coarse-grained pattern comprised of a set of finer-grained patterns. Compound patterns are explained in detail at the beginning of Chapter 22.

What's a Design Pattern Language?

A *pattern language* is a set of related patterns that act as building blocks in that they can be carried out in *pattern sequences* (or *pattern application sequences*), where each subsequent pattern builds upon the former. As explained shortly in the *Historical Influences* section, the notion of a pattern language originated in building architecture as did the term “pattern sequence” used in association with the order in which patterns can be carried out.

Some pattern languages are open-ended, allowing patterns to be combined into a variety of creative sequences, while others are more structured whereby groups of patterns are presented in a suggested application sequence. In this case, the pattern sequence is generally based on the granularity of the patterns, in that coarser grained patterns are applied prior to finer-grained ones that then build upon or extend the foundation established by the coarse-grained patterns. In these types of pattern languages, the manner in which patterns can be organized into pattern sequences may be limited to how they are applied within their groups.

Structured pattern languages are helpful because they:

- can organize groups of field-tested design patterns into proposed, field-tested application sequences
- ensure consistency in how particular design goals are achieved (because by carrying out sets of inter-dependent patterns in a proven order, the quality of the results can be more easily guaranteed)
- are effective learning tools that can provide insight into how and why a particular method or technique should be applied as well as the effects of its application
- provide an extra level of depth in relation to pattern application (because they document the individual patterns plus the cumulative effects of their application)
- are flexible in that the ultimate pattern application sequence is up to the practitioner (and also because the application of any pattern within the overall language can be optional)

This book in its entirety provides an open-ended, master pattern language for SOA. The extent to which different patterns are related can vary, but overall they share a common

objective and endless pattern sequences can be explored. The relationship diagrams explained in the upcoming *Pattern Relationship Figures* section will often hint at common application sequences for a given pattern.

Chapters 6 and 11 single out sets of closely related patterns and structure them into groups organized into recommended application sequences that essentially establish primitive design processes. As a result, these collections of patterns could be classified as “mini” structured pattern languages that are still part of the overall master pattern language.

What’s a Design Pattern Catalog?

A design pattern catalog is simply a documented collection of related design patterns. Therefore, this book is also referred to as a catalog for design patterns associated with SOA and service-orientation.

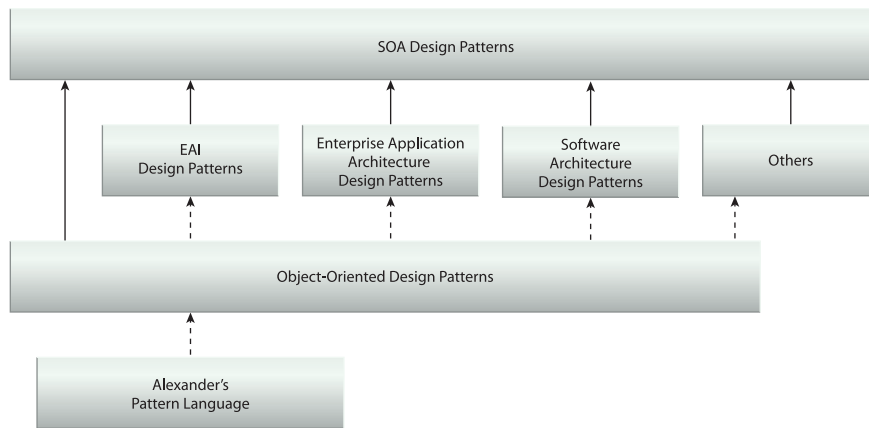
SUMMARY OF KEY POINTS

- A design pattern is a formal documentation of a proven solution to a common problem.
- A design pattern language is a group of related design patterns that can be applied in a variety of creative application sequences.
- A design pattern catalog is a collection of related design patterns documented together.
- This book contains a design pattern catalog that documents a master pattern language for SOA.

5.2 Historical Influences

Because service-orientation has deep roots in past distributed computing design platforms, many of the SOA design patterns have origins and influences that can be traced back to established design concepts, approaches, and previously published design pattern catalogs.

As illustrated in Figure 5.1, object-orientation, EAI, enterprise application architecture, and software architecture in general represent areas for which well-recognized design pattern catalogs exist, each of which has influenced design patterns in this book. Starting with the original pattern language created by Christopher Alexander, let’s briefly discuss these influences separately.

**Figure 5.1**

The primary influences of SOA design patterns.

Alexander's Pattern Language

It's been well documented how the notion of the design pattern owes its existence to the work of Christopher Alexander. Just about every design pattern publication pays tribute to Alexander's pattern language as a fundamental influence and source of inspiration.

Alexander pioneered the concept of patterns in relation to building architecture and related areas, such as city and community structure. He documented a collection of patterns and organized them into a pre-defined series he called a "sequence." The result was an architectural pattern language that inspired the IT community to create their own patterns for the design of automated systems.

Alexander's work is more than just a historical footnote for design patterns; it provides insight into how patterns in general should and should not be structured and organized.

For example, some lessons learned from Alexander's work include:

- *Pattern language sequences need to add value.* Often related patterns are better documented independently from each other even if there is some potential for them to be organized into a sequence. The primary purpose of any application sequence established by a pattern language is *not* to provide a logical organization for a set of related patterns but to demonstrate a proven process that provides value on its own.
- *Patterns do not need to be normalized.* There is often a perception that each design pattern should own an individual domain. In reality, the problem and solution space represented by individual patterns sometimes naturally overlaps. For example, you can easily have two patterns that propose different solutions to the same problem.

Beyond just the idea of organizing solutions into a pattern format, Alexander helped advocate the importance of clarity in how pattern catalogs need to be documented. He preached that patterns need to be individually clear as to their purpose and applicability and that pattern languages need to further communicate the rationale behind any sequences they may propose.

NOTE

As provided by research from Dr. Peter H. Chang from Lawrence Technological University, earlier origins also exist. For example, George Polya published the book *How to Solve It* (Princeton University Press) back in 1945, which included a “problem solving plan” that can be viewed at www.math.utah.edu/~pa/math/polya.html (based on the second edition released in 1957). Furthermore, Marvin Minsky published the paper *Steps Toward Artificial Intelligence* for MIT in 1960 that included coverage of pattern recognition and made further reference to Polya’s work.

Object-Oriented Patterns

A variety of design patterns in support of object-orientation surfaced over the past 15 years, the most recognized of which is the pattern catalog published in *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma, Helm, Johnson, Vlissides; Addison-Wesley, 1995). This set of 23 patterns produced by the “Gang of Four” expanded and helped further establish object-orientation as a design approach for distributed solutions. Some of these patterns have persisted within service-orientation, albeit within an augmented context and new names.

For example, the following patterns in this book are related:

- Capability Composition (521) is associated with Composite
- Service Façade (333) is derived from Façade
- Legacy Wrapper (441) is derived from Adapter
- Non-Agnostic Context (319) is associated with Mediator
- Decoupled Contract (401) is associated with Bridge

Concepts established by several additional object-orientation patterns have factored into other SOA patterns. The incorporation of these patterns within service-orientation is a testament to their importance and evidence of how object-orientation as a whole has influenced the evolution of SOA.

Another relevant object-oriented-related influence is the paper “Using Pattern Languages for Object-Oriented Programs” published by Kent Beck and Ward Cunningham for the 1987 OOPSLA conference. This paper is notable not only for its brevity, but for its vision and its explicit emphasis on the use of sequences in organizing patterns.

NOTE

The comparative analysis in Chapter 14 of *SOA Principles of Service Design* provides a study of how object-oriented design concepts and principles relate to service-orientation.

Software Architecture Patterns

As design patterns became a mainstream part of IT, a set of important books emerged establishing formal conventions for pattern documentation and providing a series of common design patterns for software architecture in general. These pattern catalogs were developed in five separate volumes over a period of a dozen years as part of the *Pattern-Oriented Software Architecture* series (F. Buschmann, K. Henney, M. Kircher, R. Meunier, H. Rohnert, D. Schmidt, P. Sommerlad, M. Stal, Wiley 1996–2007).

Because of the general nature of the patterns, the contributions made by this series are too voluminous to document individually. Here are some examples of how SOA design patterns relate:

- Service Layers (143) is associated with Layers
- Service Broker (707) compound pattern is associated with Broker
- Concurrent Contracts (421) is associated with Extension Interface
- Metadata Centralization (280) is associated with Lookup
- Event-Driven Messaging (599) is derived from Publisher-Subscriber
- Process Abstraction (182) is associated with Whole-Part
- Asynchronous Queuing (582) is associated with Messaging and Message Channel
- Atomic Service Transaction (623) is associated with Coordinator and Task Coordinator
- Inventory Endpoint (260) is a specialization of Message Endpoint
- Partial State Deferral (356) is associated with Partial Acquisition

It is also worth noting that Volume 4 of the series (entitled *A Pattern Language for Distributed Computing*) focuses on connecting existing patterns relevant to building distributed systems into a larger pattern language. This book documents the roots of various previously published patterns, including those that are part of other pattern catalogs listed in this section.

Enterprise Application Architecture Patterns

As distributed computing became an established platform for solution design, an emphasis on enterprise architecture emerged bringing with it its own set of design patterns, many of which built upon object-oriented concepts and patterns. A respected pattern catalog in this field was published in *Patterns of Enterprise Application Architecture* (Fowler, Addison-Wesley, 2003).

You might notice that many of the influences originating from enterprise architecture patterns are located in the two pattern languages provided in Chapters 6 and 11. Service-orientation is, at heart, a design paradigm for distributed computing, and although distinct, it still relies and builds upon the fundamental patterns and concepts associated with enterprise application architecture in general.

For example, the following patterns in this book are related:

- Service Encapsulation (305) is associated with Gateway and Service Layer
- Decoupled Contract (401) is associated with Implementation Separated Interface
- Service Façade (333) is derived from Remote Façade
- Stateful Services (248) is derived from Server Session State
- Partial State Deferral (356) is derived from Lazy State
- State Repository (242) is derived from Database Session State

Studying these types of influences can lead to further revelations as to how SOA has evolved into a unique architectural model.

EAI Patterns

Several pattern catalogs centered around the use of messaging to fulfill integration requirements emerged during the EAI era. These patterns establish sound approaches for robust messaging-based communication and address various integration-related challenges.

A recognized publication in this field is *Enterprise Integration Patterns* (Hohpe, Woolf, Addison-Wesley, 2004).

Because EAI is one of the primary influences of service-orientation, this book contains service interaction patterns based on the use of messaging primarily in support of service composition scenarios.

Some examples of SOA patterns related to design patterns documented in *Enterprise Integration Patterns*:

- Service Messaging (533) is derived from Message, Messaging, and Document Message
- Data Model Transformation (671) is derived from Message Translator
- Canonical Schema (158) is associated with Canonical Data Model
- Service Agent (543) is associated with Event-Driven Consumer
- Process Centralization (193) is associated with Process Manager
- Intermediate Routing (549) is derived from Content-Based Router

Several references to additional EAI patterns are interspersed in the upcoming chapters (Chapter 18, in particular).

SOA Patterns

The intention behind this collection of SOA patterns is not to replace or compete with the catalogs provided by previous publications, but instead to build upon and complement them with a catalog focused solely on attaining the strategic goals associated with service-orientated computing.

This catalog is comprised of new patterns, existing patterns that have been augmented, and patterns that are intentionally similar to patterns in other catalogs. The latter group is included so that these patterns can be explained within the context of SOA and to also formally highlight them as a supporting part of the service-orientation design paradigm.

Learning about the design solutions and techniques provided by SOA design patterns can provide insight into the mechanics required to enable service-orientation and also help clarify exactly how SOA represents a distinct architectural model. When exploring these distinctions, it is important to take into account:

5.3 Pattern Notation

95

- which of the past design techniques are preserved and emphasized
- which of the past design techniques are de-emphasized
- new design techniques
- new approaches to carrying out existing techniques

Note that there are several more useful design patterns in the previously mentioned books which are not mentioned in this pattern catalog. Some provide detailed solutions that are not necessarily specific to SOA, but still very helpful.

SUMMARY OF KEY POINTS

- The pattern language invented by Christopher Alexander inspired the use of design patterns in the IT world.
- The object-orientation platform has an established set of design patterns that are at the root of several of the patterns in this catalog. Additional influences can be traced back to patterns created for enterprise application architecture, EAI, and general software architecture pattern catalogs.

5.3 Pattern Notation

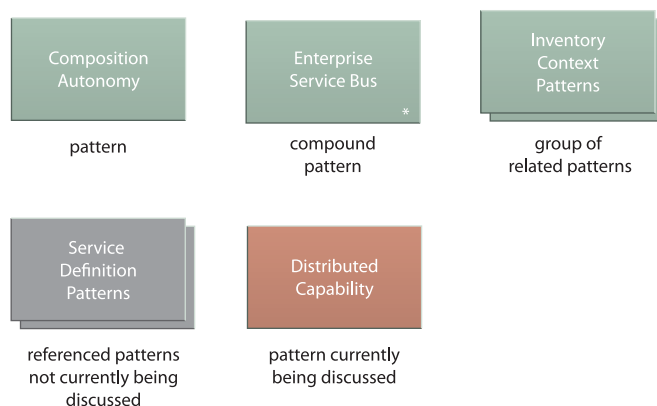
Throughout this book design patterns need to be referenced and explained in text and illustrations. A simple notation is used to consistently represent different types of patterns.

Pattern Symbols

As shown in Figure 5.2, specific symbols are used to represent:

- a design pattern
- a compound design pattern
- a group of related design patterns

Additionally, colors are incorporated to indicate if a displayed design pattern is just being referenced and not actually discussed, versus one that is the current topic of discussion.

**Figure 5.2**

The standard symbols used to represent different types of design patterns and how design patterns relate to the current subject being covered.

Pattern Figures

The symbols displayed in Figure 5.2 are used in the following three primary types of diagrams:

- pattern application sequence figures
- pattern relationship figures
- compound pattern hierarchy figures

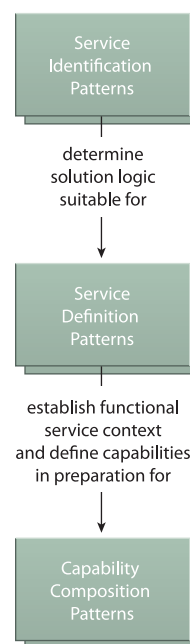
Let's take a closer look at each:

Pattern Application Sequence Figures

When documenting design pattern languages, it is helpful to display the suggested sequence in which patterns should be applied. Figures 5.3 and 5.4 show pattern application sequences for groups of related patterns and for individual patterns belonging to a particular group, respectively.

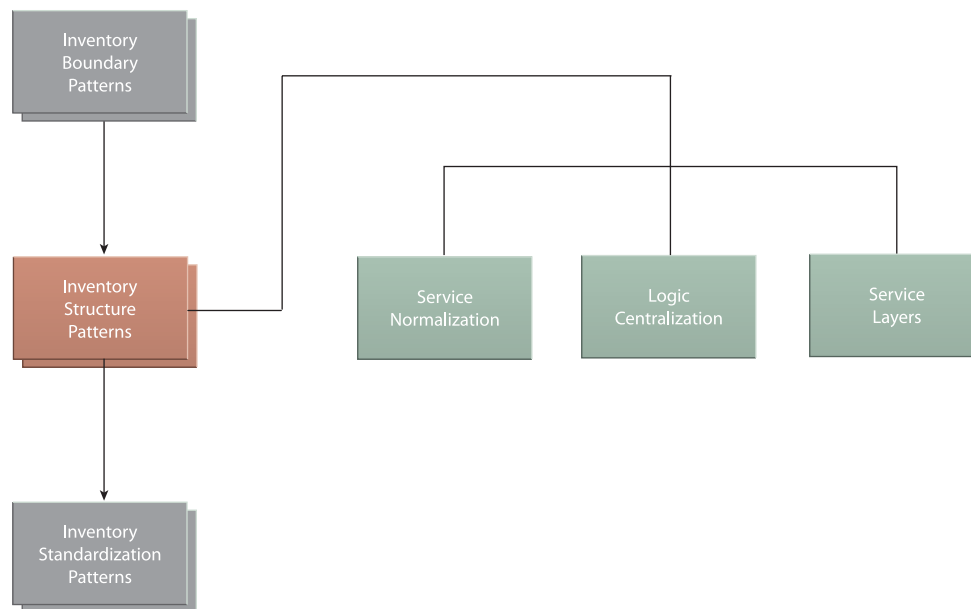
Pattern Relationship Figures

As explained in the upcoming *Pattern Profiles* section, this book explores numerous inter-pattern relationships and provides one pattern relationship diagram (Figure 5.5) for each documented design pattern.

**Figure 5.3**

The pattern groups from Chapters 11 and 17 displayed in a recommended application sequence.

5.3 Pattern Notation

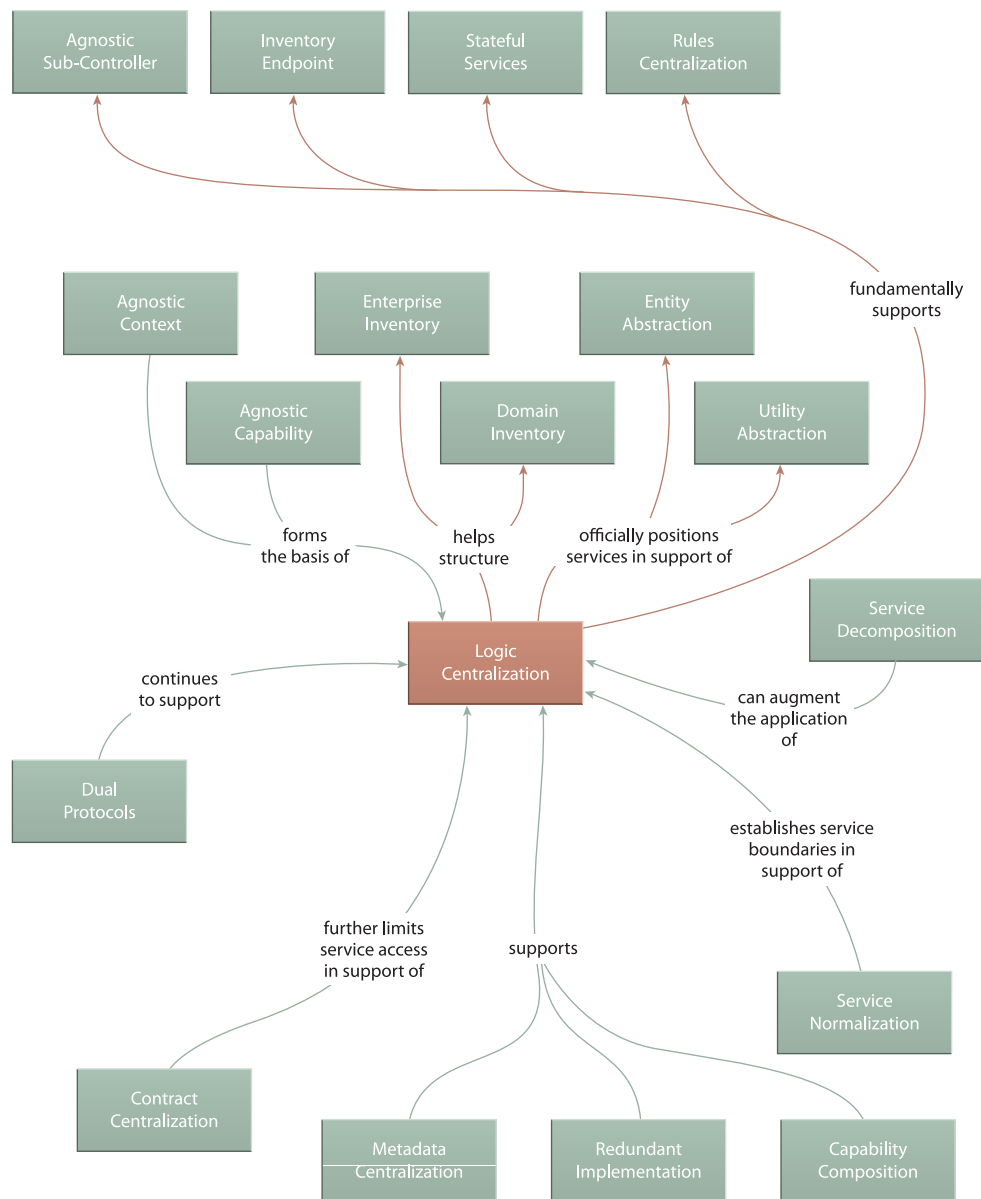
**Figure 5.4**

The inventory structure patterns group from Chapter 6 is highlighted in this diagram. In this case, there is no recommendation as to the order in which the three patterns on the right should be applied.

A style convention applied to all pattern relationship diagrams is the use of color, as follows:

- Each pattern relationship diagram explores the relationships of one pattern. Therefore, that design pattern is highlighted in red, as per the previously established symbol notation.
- Pattern relationships are documented in a unidirectional manner. For relationships where the pattern currently being discussed affects or relates to other patterns, a red line is used along with an arrow pointing to the other pattern. When the relationship line documents how other patterns relate to the current pattern, the lines are green, and the arrows are reversed.

Note that directionality of relationships is preserved in different diagrams. For example, the green relationship line emitting from Service Normalization (131) and pointing to Logic Centralization (136) in the preceding figure would be reversed (and colored red) in the pattern relationship figure for Service Normalization (131).

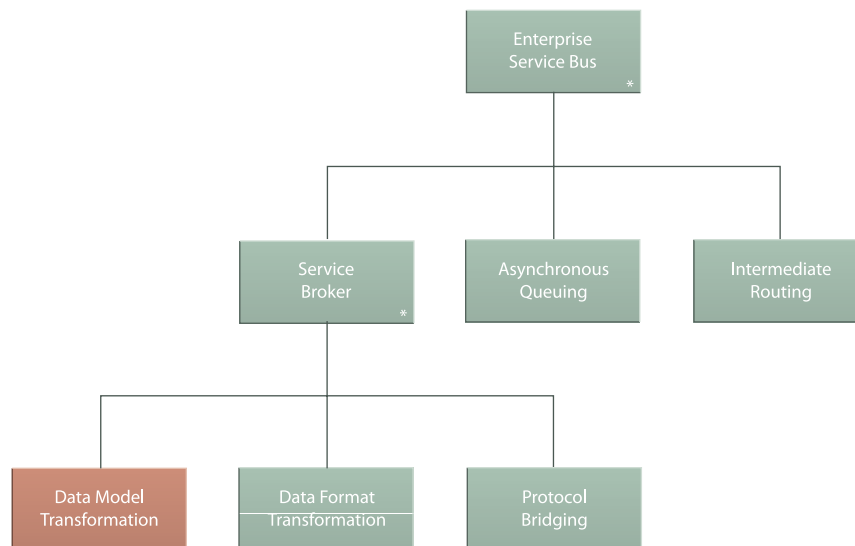
**Figure 5.5**

An example of a pattern relationship diagram.

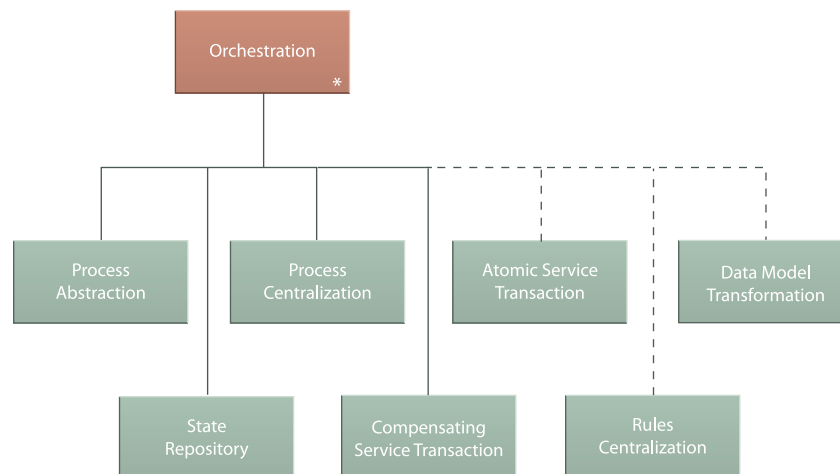
Compound Pattern Hierarchy Figures

Compound patterns are comprised of combinations of design patterns. When illustrating a compound pattern, a hierarchical representation is usually required, where the compound pattern name is displayed at the top, and the patterns that comprise the compound are shown underneath.

These types of diagrams (Figures 5.6 and 5.7) can be considered simplified relationship figures in that they only identify which patterns belong to which compound, without getting into the details of how these patterns relate. Compound patterns are documented separately in Chapter 22, but compound hierarchy figures are displayed throughout the upcoming chapters.

**Figure 5.6**

Enterprise Service Bus (704) is a compound pattern comprised of several core patterns, one of which is a compound pattern in its own right and therefore represents a nested pattern hierarchy. In this case, Data Model Transformation (671) is highlighted, indicating that it is the current pattern being discussed.

**Figure 5.7**

There are additional patterns associated with Orchestration (701) that can be considered optional extensions. In this case, the hierarchy lines are dashed.

NOTE

Another notation used for some forms of compound patterns involves showing patterns combined with a plus (“+”) symbol. These diagrams are limited to Chapter 22 and are formally described there.

Capitalization

All design pattern names (including names of compound patterns) are capitalized throughout this book. The names for groups of related patterns are capitalized when displayed in Figures but not when referenced in body text.

Page Number References

As you may have already noticed in earlier parts of this chapter, each pattern name is followed by a page number in parentheses. This number, which points to the first page of the corresponding pattern profile, is provided for quick reference purposes. Its use has become a common convention among pattern catalogs. The only time the number is not displayed is when a pattern name is referenced within that pattern’s profile section.

5.4 Pattern Profiles

Each of the patterns in this catalog is described using the same profile format and structure based on the following parts:

5.4 Pattern Profiles

101

- Requirement
- Icon
- Summary
- Problem
- Solution
- Application
- Impacts
- Relationships
- Case Study Example

The following sections describe each part individually.

Requirement

This is a concise, single-sentence statement that presents the fundamental requirement addressed by the pattern in the form of a question. Every pattern description begins with this statement.

For example:

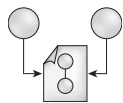
How can a service be designed to minimize the chances of capability logic deconstruction?

Note that the inside cover of this book lists all of the patterns together with their respective requirement statements.

Icon

Each pattern description is accompanied by an icon image that acts as a visual identifier.

An example of a pattern icon:



The icons are displayed together with the requirement statements in each pattern profile as well as on the inside book cover.

Summary

Following the requirement statement, a summary table is displayed, comprised of statements that collectively provide a concise synopsis of the pattern for quick reference purposes.

The following parts of the profile are summarized in this table:

- Problem
- Solution
- Application
- Impacts

Additionally, the profile table provides references to related service-orientation design principles and service-oriented architectural types via the following sections:

- Principles
- Architecture

The parts of the pattern description not represented in the summary table are the *Relationships* and *Case Study Example* sections.

NOTE

All pattern summary tables in this book are also published online at SOAPatterns.org.

Problem

The issue causing a problem and the effects of the problem are described in this section, typically accompanied by a figure that further illustrates the “problem state.” It is this problem for which the pattern provides a solution. Problem descriptions may also include common circumstances that can lead to the problem (also known as “forces”).

Solution

This represents the design solution proposed by the pattern to solve the problem and fulfill the requirement. Often the solution is a short statement followed by a diagram that concisely communicates the final solution state. “How-to” details are not provided in this section but are instead located in the *Application* section.

Application

This part is dedicated to describing how the pattern can be applied. It can include guidelines, implementation details, and sometimes even a suggested process.

Impacts

Most patterns come with trade-offs. This section highlights common consequences, costs, and requirements associated with the application of a pattern.

Note that these consequences are common but not necessarily predictable. For example, issues related to typical performance requirements are often raised; however, these issues may not impact an environment with an already highly scalable infrastructure.

Relationships

The use of design patterns can tie into all aspects of design and architecture. It is important to understand the requirements and dependencies a pattern may have and the effects of its application upon other patterns.

These diagrams are not exhaustive in that not all possible relationships a given design pattern can have are shown. Through the use of pattern relationship figures, this section merely highlights common relationships with an emphasis on how patterns support or depend on each other.

NOTE

Because there are two patterns in each relationship, almost every relationship is shown twice in this book: once in the *Relationships* section of each of the two patterns. To avoid content redundancy, most relationships are only described once. Therefore, if you find a relationship shown in a diagram that is not described in the accompanying text, refer to the description for the other pattern involved in that relationship. Note, however, that some relationships are considered self-explanatory and are therefore not described at all.

Details regarding the format of pattern relationship figures are provided in the *Pattern Notation* section earlier in this chapter.

Case Study Example

Most pattern profiles conclude with a case study example that demonstrates the sample application of a pattern in relation to the storylines established in Chapter 2.

SUMMARY OF KEY POINTS

- Each design pattern is documented with the same profile structure.
- Design pattern profiles begin with a requirements statement and an icon and then provide a summary table followed by sections with detailed descriptions.

5.5 Patterns with Common Characteristics

Each pattern in this book is distinct and unique and is considered an equal member of the overall pattern catalog. However, it is worth highlighting certain groups of similar patterns to better understand how they were named and why they share common characteristics.

NOTE

The following sections do not attempt to group patterns into formal categories. The upcoming chapters in Parts II, III, and IV already are subdivided by chapters representing specific pattern types. These sections here only point out that within and across these types, collections of patterns share common qualities and were labeled to reflect this.

Canonical Patterns

Canonical design patterns propose that the best solution for a particular problem is to introduce a design standard. The successful application of this type of pattern results in a canonical convention that guarantees consistent design across different parts of an inventory or solution.

The canonical design patterns in this book are:

- Canonical Protocol (150)
- Canonical Schema (158)
- Canonical Expression (275)
- Canonical Resources (237)
- Canonical Versioning (286)

Centralization Patterns

Centralization simply means limiting the options of something to one. Applying this concept within key parts of a service-oriented architecture establishes consistency and fosters standardization and reuse and, ultimately, native interconnectivity.

The following centralization patterns are covered in the upcoming chapters:

- Logic Centralization (136)
- Metadata Centralization (280)
- Process Centralization (193)
- Rules Centralization (216)
- Schema Centralization (200)
- Contract Centralization (409)
- Policy Centralization (207)

A common characteristic across centralization patterns is a trade-off between increased architectural harmony and increased governance and performance requirements. As explained shortly in the *Measures of Pattern Application* section, patterns can be applied to different extents. A key factor when assessing the application measure for centralization patterns is at what point the benefit outweighs the architectural impact.

NOTE

Centralization patterns are also very much related to the use of design standards. To constantly require that certain parts of a service-oriented architecture are centralized requires that supporting conventions be regularly followed.

SUMMARY OF KEY POINTS

- Canonical and centralization patterns need to be consistently applied to realize their benefits.
- Canonical and centralization patterns require the use of supporting design standards.

5.6 Key Design Considerations

“Enterprise” vs. “Enterprise-wide”

Having discussed the notion of services as enterprise resources back in Chapter 4, it is important that there is a clear distinction between something that exists as a resource as part of an enterprise and something that is actually an *enterprise-wide* resource.

- An enterprise resource is not a resource that is necessarily made available across the entire enterprise. Instead, it is a resource positioned for use within the enterprise, outside of and beyond any one particular application boundary. In other words, it is a “cross-silo” resource.
- An enterprise-wide resource, on the other hand, is truly intended for use across all service inventories within an enterprise.

This difference in terminology is especially relevant to design patterns associated with specific enterprise boundaries, such as Domain Inventory (123). Note also that a service positioned as an enterprise resource is expected to be an inventory-wide resource, meaning that it is interoperable from anywhere within the inventory boundary.

Design Patterns and Design Principles

Most of the upcoming design patterns reference design principles where appropriate to highlight a dependency or relationship or perhaps to describe the effect a design pattern may have on service-orientation.

Specifically, the relationship between service-orientation design principles and patterns can be defined as follows:

- Design principles are applied collectively to solution logic in order to shape it in such a manner that it fosters key design characteristics that support the strategic goals associated with service-oriented computing.

- Design patterns provide solutions to common problems encountered when applying design principles—and—when establishing an environment suitable for implementing logic designed in accordance with service-orientation principles.

In many ways, design principles and patterns are alike. Both provide design guidance in support of achieving overarching strategic goals. In fact, it would not be unreasonable to think of the eight service-orientation principles as super patterns that are further supported by the patterns in this book.

Service-orientation design principles have another role in that they collectively define service-orientation as a design paradigm. Ultimately, it is best to view design patterns as providing support for the realization of design principles and their associated goals. (Design principles were introduced in the *Principles of Service-Oriented* section in Chapter 4.)

NOTE

We just stated that design principles could be thought of as super patterns. Why then weren't they documented as such? When the manuscript for this book was undergoing a review by Ralph Johnson and his pattern review group at UIUC, the question came up as to how to determine whether something is a legitimate pattern.

Ralph responded by stating, "When people ask me, 'Is this a pattern?' I usually say, 'That is not the right question.' The right question is whether pattern form is the best way to communicate this material." This is a good way to think of the purpose of this book.

Each pattern provides a specific solution to a distinct problem. The guidance provided by a design principle is much broader and can, in fact, end up solving a variety of problems. Therefore, design principles are better off documented in their form.

Design Patterns and Design Granularity

Design granularity, as it pertains to service-orientation, is itself something worth being familiar with prior to reading the upcoming chapters. Provided here are brief descriptions of common granularity-related terms:

- *Service Granularity* – The overall quantity of functionality encapsulated by a service determines the service granularity. A service's granularity is set by its functional context, which is usually established during the service modeling phase.
- *Capability Granularity* – The quantity of functionality encapsulated by a specific service capability determines the level of corresponding capability granularity.

- *Data Granularity* – The quantity of data exchanged by a specific service capability determines the level of its data granularity.
- *Constraint Granularity* – The extent of validation logic detail defined for a given service capability within the service contract determines the capability's level of constraint granularity. Generally, the more specific the constraints and the larger the amount of constraints, the more fine-grained the capability's constraint granularity is.

The effect of design patterns on service-related design granularity can vary. For example, when applying multiple patterns (or compound patterns) to the same service, the end-levels of design granularity may be distinctly defined by that combination of patterns (and they may fluctuate between the application of one pattern to another).

Measures of Design Pattern Application

It is important to acknowledge that most patterns do not propose a black or white design option. Design patterns can often be applied at different levels. Although the effectiveness of a given pattern will generally be equivalent to the extent to which it is realized, there may be practical considerations that simply limit the degree to which a pattern can be applied in the real world (as is often the case when designing service logic that is required to encapsulate legacy functionality).

This consideration affects both design patterns and design principles. For example, individual service-orientation design principles can rarely be applied to their maximum potential. The point is to pursue the design goals of a design pattern or principle to whatever extent feasible and to strive for an end-result that realizes the pattern or principle to a meaningful extent.

SUMMARY OF KEY POINTS

- Some specific terminology is used within design pattern profiles. The distinction between “enterprise” and “enterprise-wide” is especially important.
- Design pattern profiles contain references to related design principles, revealing links between the patterns and the realization of service-orientation itself.
- As with design principles, most design patterns can be applied to various measures. Sometimes it isn't possible to fully apply a design pattern due to environmental constraints.

Index of Patterns

Agnostic Capability (Erl), 324
Agnostic Context (Erl), 312
Agnostic Sub-Controller (Erl), 607
Asynchronous Queuing (Little, Rischbeck, Simon), 582
Atomic Service Transaction (Erl), 623
Brokered Authentication (Hogg, Smith, Chong, Hollander, Kozaczynski, Brader, Delgado, Taylor, Wall, Slater, Imran, Cibraro, Cunningham), 661
Canonical Expression (Erl), 275
Canonical Protocol (Erl), 150
Canonical Resources (Erl), 237
Canonical Schema (Erl), 158
Canonical Schema Bus (Utschig, Maier, Trops, Normann, Winterberg, Erl), 709
Canonical Versioning (Erl), 286
Capability Composition (Erl), 521
Capability Recomposition (Erl), 526
Compatible Change (Orchard, Riley), 465
Compensating Service Transaction (Utschig, Maier, Trops, Normann, Winterberg, Loesgen, Little), 631
Composition Autonomy (Erl), 616
Concurrent Contracts (Erl), 421
Contract Centralization (Erl), 409
Contract Denormalization (Erl), 414
Cross-Domain Utility Layer (Erl), 267
Data Confidentiality (Hogg, Smith, Chong, Hollander, Kozaczynski, Brader, Delgado,

Taylor, Wall, Slater, Imran, Cibraro, Cunningham), 641

Data Format Transformation (Little, Rischbeck, Simon), 681

Data Model Transformation (Erl), 671

Data Origin Authentication (Hogg, Smith, Chong, Hollander, Kozaczynski, Brader, Delgado, Taylor, Wall, Slater, Imran, Cibraro, Cunningham), 649

Decomposed Capability (Erl), 504

Decoupled Contract (Erl), 401

Direct Authentication (Hogg, Smith, Chong, Hollander, Kozaczynski, Brader, Delgado, Taylor, Wall, Slater, Imran, Cibraro, Cunningham), 656

Distributed Capability (Erl), 510

Domain Inventory (Erl), 123

Dual Protocols (Erl), 227

Enterprise Inventory (Erl), 116

Enterprise Service Bus (Erl, Little, Rischbeck, Simon), 704

Entity Abstraction (Erl), 175

Event-Driven Messaging (Little, Rischbeck, Simon), 599

Exception Shielding (Hogg, Smith, Chong, Hollander, Kozaczynski, Brader, Delgado, Taylor, Wall, Slater, Imran, Cibraro, Cunningham), 376

Federated Endpoint Layer (Erl), 713

File Gateway (Roy), 457

Functional Decomposition (Erl), 300

Intermediate Routing (Little, Rischbeck, Simon), 549

Inventory Endpoint (Erl), 260

Legacy Wrapper (Erl, Roy), 441

Logic Centralization (Erl), 136

Message Screening (Hogg, Smith, Chong, Hollander, Kozaczynski, Brader, Delgado, Taylor, Wall, Slater, Imran, Cibraro, Cunningham), 381

Messaging Metadata (Erl), 538

Metadata Centralization (Erl), 280

Multi-Channel Endpoint (Roy), 451

Non-Agnostic Context (Erl), 319

Official Endpoint (Erl), 711

Orchestration (Erl, Loesgen), 701

Partial State Deferral (Erl), 356
Partial Validation (Orchard, Riley), 362
Policy Centralization (Erl), 207
Process Abstraction (Erl), 182
Process Centralization (Erl), 193
Protocol Bridging (Little, Rischbeck, Simon), 687
Proxy Capability (Erl), 497
Redundant Implementation (Erl), 345
Reliable Messaging (Little, Rischbeck, Simon), 592
Rules Centralization (Erl), 216
Schema Centralization (Erl), 200
Service Agent (Erl), 543
Service Broker (Little, Rischbeck, Simon), 707
Service Callback (Karmarkar), 566
Service Data Replication (Erl), 350
Service Decomposition (Erl), 489
Service Encapsulation (Erl), 305
Service Façade (Erl), 333
Service Grid (Chappell), 254
Service Instance Routing (Karmarkar), 574
Service Layers (Erl), 143
Service Messaging (Erl), 533
Service Normalization (Erl), 131
Service Perimeter Guard (Hogg, Smith, Chong, Hollander, Kozaczynski, Brader, Delgado, Taylor, Wall, Slater, Imran, Cibraro, Cunningham), 394
Service Refactoring (Erl), 484
State Messaging (Karmarkar), 557
State Repository (Erl), 242
Stateful Services (Erl), 248
Termination Notification (Orchard, Riley), 478
Three-Layer Inventory (Erl), 715
Trusted Subsystem (Hogg, Smith, Chong, Hollander, Kozaczynski, Brader, Delgado, Taylor, Wall, Slater, Imran, Cibraro, Cunningham), 387

794

Index of Patterns

UI Mediator (Utschig, Maier, Trops, Normann, Winterberg), 366

Utility Abstraction (Erl), 168

Validation Abstraction (Erl), 429

Version Identification (Orchard, Riley), 472

THE PRENTICE HALL SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL


**Service-Oriented Architecture:
A Field Guide to Integrating XML and Web Services**

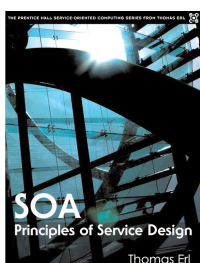
ISBN 0131428985

This top-selling field guide offers expert advice for incorporating XML and Web services technologies within service-oriented integration architectures.


**Service-Oriented Architecture:
Concepts, Technology, and Design**

ISBN 0131858580

Widely regarded as the definitive “how-to” guide for SOA, this best-selling book presents a comprehensive end-to-end tutorial that provides step-by-step instructions for modeling and designing service-oriented solutions from the ground up.


SOA Principles of Service Design

ISBN 0132344823

Published with over 240 color illustrations, this hands-on guide contains practical, comprehensive, and in-depth coverage of service engineering techniques and the service-orientation design paradigm. Proven design principles are documented to help maximize the strategic benefit potential of SOA.


Web Service Contract Design and Versioning for SOA

ISBN: 9780136135173

For Web services to succeed as part of SOA, they require balanced, effective technical contracts that enable services to be evolved and repeatedly reused for years to come. Now, a team of industry experts presents the first end-to-end guide to designing and governing Web service contracts.


SOA Design Patterns

ISBN 0136135161

Software design patterns have emerged as a powerful means of avoiding and overcoming common design problems and challenges. This new book presents a formal catalog of design patterns specifically for SOA and service-orientation. All patterns are documented using full-color illustrations and further supplemented with case study examples.

Several additional series titles are currently in development and will be released soon. For more information about any of the books in this series, visit www.soabooks.com.

