



SERVICE-ORIENTED DESIGN WITH RUBY AND RAILS

Foreword by **Obie Fernandez**, *Series Editor*

PAUL DIX

with TROTTER CASHION ■ BRYAN HELMKAMP ■ JAKE HOWERTON

Download from www.wowebook.com

SERVICE-ORIENTED DESIGN WITH RUBY AND RAILS

SERVICE-ORIENTED DESIGN WITH RUBY AND RAILS

Paul Dix

◆ Addison-Wesley

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

Download from www.wowebook.com

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

Visit us on the Web: informat.com/aw

Library of Congress Cataloging-in-Publication Data

Dix, Paul, 1977-

Service-oriented design with Ruby and Rails / Paul Dix.

p. cm.

Includes bibliographical references and index.

ISBN 0-321-65936-8 (pbk. : alk. paper) 1. Web services.

2. Service-oriented architecture (Computer science) 3. Web sites—Design.

4. Ruby on rails (Electronic resource) I. Title.

TK5105.88813.D593 2010

006.7'8—dc22

2010021623

Copyright © 2011 Pearson Education, 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: 978-0-321-65936-1

ISBN-10: 0-321-65936-8

Text printed in the United States on recycled paper at Courier in Stoughton, Massachusetts.

First printing, August 2010

Associate Publisher

Mark Taub

Acquisitions Editor

Debra Williams Cauley

Development Editor

Michael Thurston

Managing Editor

John Fuller

Project Editor

Elizabeth Ryan

Copy Editor

Kitty Wilson

Indexer

Jack Lewis

Proofreader

Carol Lallier

Technical Reviewers

Jennifer Lindner,
Trotter Cashion

Cover Designer

Chuti Prasertsith

Compositor

LaserWords

Download from www.wowebook.com

To Pops, for encouraging my weird obsession with computers.

This page intentionally left blank

Contents

Foreword	xiii
Preface	xv
Acknowledgments	xix
About the Author	xxi
1 Implementing and Consuming Your First Service	1
What's a Service?	1
Service Requirements	2
The Ruby Tool Set	2
Sinatra	2
ActiveRecord	3
JSON	3
Typhoeus	4
RSpec	4
The User Service Implementation	5
Using GET	6
POSTing a User	11
PUTing a User	13
Deleting a User	15
Verifying a User	16
Implementing the Client Library	18
Finding a User	18
Creating a User	21
Updating a User	22
Destroying a User	24
Verifying a User	24
Putting It All Together	26
Conclusion	26

2	An Introduction to Service-Oriented Design	27
	Use of Service-Oriented Design in the Wild	27
	Service-Oriented Design Versus Service-Oriented Architecture Versus RESTful-Oriented Architecture	28
	Making the Case for Service-Oriented Design	29
	Isolation	30
	Robustness	34
	Scalability	35
	Agility	36
	Interoperability	37
	Reuse	38
	Conclusion	38
3	Case Study: Social Feed Reader	41
	A Typical Rails Application	41
	The Rails Social Feed Reader Application	45
	Features	46
	Current Setup	46
	Converting to Services	54
	Segmenting into Services	54
	Breaking Up the Application into Services	54
	Conclusion	58
4	Service and API Design	59
	Partitioning Functionality into Separate Services	59
	Partitioning on Iteration Speed	60
	Partitioning on Logical Function	61
	Partitioning on Read/Write Frequencies	62
	Partitioning on Join Frequency	63
	Versioning Services	64
	Including a Version in URIs	64
	Using Accept Headers for Versioning	65
	URIs and Interface Design	66
	Successful Responses	68
	HTTP Status Codes	68
	HTTP Caching	69
	Successful Response Bodies	70

Error Responses	72
HTTP Status Codes	72
Error Response Bodies	72
Handling Joins	73
Storing References	73
Joining at the Highest Level	74
Beware of Call Depth	75
API Complexity	75
Atomic APIs	76
Multi-Gets	76
Multiple Models	77
Conclusion	78
5 Implementing Services	79
The Vote Service	79
A Multi-Get Interface	81
The Vote Interface	82
API Design Guidelines	85
Models	86
Rails	88
Rails 2.3 Routes	88
Rails 3 Routes	89
The Rails Controller	90
Sinatra	95
Rack	100
Conclusion	106
6 Connecting to Services	107
Blocking I/O, Threading, and Parallelism	107
Asynchronous I/O	108
Multi-threading	108
Typhoeus	109
Making Single Requests	109
Making Simultaneous Requests	111
Multi-threaded Requests	113
JRuby	115
Logging for Performance	117

Handling Error Conditions	118
Testing and Mocking Service Calls	119
Requests in Development Environments	121
Conclusion	121
7 Developing Service Client Libraries	123
Packaging	123
Jeweler	124
Building and Deploying a Library	127
Parsing Logic	127
The JSON Gem	128
YAJL Ruby	129
Wrapping Parsed Results	130
ActiveModel	132
Validations	132
Serialization	134
Connection and Request Logic	136
Data Reads	136
Data Writes	142
Mocks, Stubs, and Tests	143
Conclusion	146
8 Load Balancing and Caching	147
Latency and Throughput	147
Load Balancing	148
Load Balancing Algorithms	148
Implementing Load Balancing	152
Caching with Memcached	155
The Memcached Client and ActiveRecord	156
Time-Based Expiration	158
Manual Expiration	159
Generational Cache Keys	160
HTTP Caching	162
Expiration-Based Caching	162
Validation-Based Caching	163
Implementing HTTP Caching	165
Conclusion	166

-
- 9 Parsing XML for Legacy Services 167**
 - XML 167
 - REXML 170
 - Nokogiri 174
 - SOAP 177
 - Exploring Web Services with a WSDL File 177
 - Making Requests 180
 - Conclusion 184
 - 10 Security 185**
 - Authentication 185
 - HTTP Authentication 186
 - Signing Requests 187
 - SSL for Authentication 198
 - Authentication for Rails Applications 199
 - Authorization 201
 - Firewalls 201
 - An RBAC Authorization Service 203
 - Encryption 209
 - SSL for Encryption 210
 - Public/Private Key Pairs for Encryption 210
 - Conclusion 214
 - 11 Messaging 215**
 - What Is Messaging? 215
 - Synchronous Versus Asynchronous Messaging 216
 - Queues 217
 - Message Formats 217
 - RabbitMQ and AMQP 217
 - Queues in RabbitMQ 218
 - Exchanges and Bindings 218
 - Durability and Persistence 223
 - Client Libraries 224
 - Synchronous Reads, Asynchronous Writes 227
 - HTTP-Based Reads 227
 - Messaging-Based Writes 227
 - The CAP Theorem 230
 - Eventual Consistency 231
 - Designing Around Consistency 232

Data Is the API	234
Operations on Fields	234
Modifications to Field Operations	235
Conclusion	236
12 Web Hooks and External Services	237
Web Hooks	238
PubSubHubbub	239
Receiving Web Hooks	240
Providing Web Hooks	242
Strategies for Dealing with Failure	244
OAuth	245
Implementing an OAuth Consumer	246
Implementing an OAuth Provider	249
Integrating with External Services	251
Consuming Data	251
Pushing Data	253
The Request Lifecycle	254
Worker Processes	254
Ensuring Performance and Reliability	258
Segregating Queues	259
Metrics	259
Throttling and Quotas	260
Conclusion	261
Appendix RESTful Primer	263
Roy Fielding's REST	263
Constraints	264
Architectural Elements	264
Architectural Views	265
REST and Resources	265
URIs and Addressability	266
Representations	267
HTTP and the Uniform Interface	268
HTTP Methods	268
HTTP Headers	271
HTTP Status Codes	274
Conclusion	275
Index	277

Foreword

It's an honor for me to present to you this timely new addition to the Professional Ruby Series, one that fills a crucially important gap in the ongoing evolution of all professional Rubyists and couldn't come a moment sooner! It is authored by one of the brightest minds of our international Ruby community, Paul Dix, described as "genius" and "A-list" by his peers. Paul is no stranger to the Ruby world, a fixture at our conferences and involved in some of the earliest Rails project work dating back to 2005. He's also the author of Typhoeus, a successful high-performance HTTP library that is an essential part of the service-oriented ecosystem in Ruby.

Why is this book so timely? Serious Ruby adoption in large companies and project settings inevitably necessitates service-oriented approaches to system design. Properly designed large applications, partitioned into cooperating services, can be far more agile than monolithic applications. Services make it easy to scale team size. As the code base of an application gets larger, it gets harder to introduce new developers to the project. When applications are split into services, developers can be assigned to a specific service or two. They only need to be familiar with their section of the application and the working groups can remain small and nimble.

There's also the fact that we live in the age of The Programmable Web, the boom of web applications, APIs, and innovation over the past few years that is directly attributable to the rise of interoperable web services like those described in this book. Applications that rely on web resources present unique challenges for development teams. Service-oriented traits impact various aspects of how applications should be designed and the level of attention that needs to be paid to how the application performs and behaves if those services are unavailable or otherwise limited.

My own teams at Hashrocket have run into challenges where we could have used the knowledge in this book, both in our Twitter applications as well as our large client projects, some of which we have been working on for years. In a couple of notable cases, we have looked back in regret, wishing we had taken a service-oriented approach

sooner. I assure you that this book will be on the required-reading list for all Rocketeers in the future.

Like Hashrocket, many of you buying this book already have big monolithic Rails applications in production. Like us, you might have concerns about how to migrate your existing work to a service-oriented architecture. Paul covers four different strategies for application partitioning in depth: Iteration Speed, Logical Function, Read/Write Frequency, and Join Frequency. Specific examples are used to explore the challenges and benefits of each strategy. The recurring case study is referred to often, to ensure the discussion is grounded in real, not imaginary or irrelevant situations.

Paul doesn't limit himself to theory either, which makes this a well-rounded and practical book. He gives us important facts to consider when running in a production environment, from load balancing and caching to authentication, authorization, and encryption to blocking I/O to parallelism, and how to tackle these problems in Ruby 1.8, 1.9, Rubinius, and JRuby.

Overall, I'm proud to assure you that Paul has given us a very readable and useful book. It is accurate and current, bringing in Rack, Sinatra, and key features of Rails 3, such as its new routing and ActiveRecord libraries. At the same time, the book achieves a timeless feeling, via its concise descriptions of service-oriented techniques and broadly applicable sample code that I'm sure will beautifully serve application architects and library authors alike for years to come.

—Obie Fernandez

Author of *The Rails Way*

Series Editor of the Addison-Wesley Professional Ruby Series

CEO & Founder of Hashrocket

Preface

As existing Ruby on Rails deployments grow in size and adoption expands into larger application environments, new methods are required to interface with heterogeneous systems and to operate at scale. While the word *scalability* with respect to Rails has been a hotly debated topic both inside and outside the community, the meaning of the word *scale* in this text is two fold. First, the traditional definition of “handling large numbers of requests” is applicable and something that the service-oriented approach is meant to tackle. Second, *scale* refers to managing code bases and teams that continue to grow in size and complexity. This book presents a service-oriented design approach that offers a solution to deal with both of these cases.

Recent developments in the Ruby community make it an ideal environment for not only creating services but consuming them as well. This book covers technologies and best practices for creating application architectures composed of services. These could be written in Ruby and tied together through a frontend Rails application, or services could be written in any language, with Ruby acting as the glue to combine them into a greater whole. This book covers how to properly design and create services in Ruby and how to consume these and other services from within the Rails environment.

Who This Book Is For

This book is written with web application and infrastructure developers in mind. Specific examples cover technologies in the Ruby programming ecosystem. While the code in this book is aimed at a Ruby audience, the design principles are applicable to environments with multiple programming languages in use. In fact, one of the advantages of the service-oriented approach is that it enables teams to implement pieces of application logic in the programming language best suited for the task at hand. Meanwhile, programmers in any other language can take advantage of these

services through a common public interface. Ultimately, Ruby could serve simply at the application level to pull together logic from many services to render web requests through Rails or another preferred web application framework.

If you're reading this book, you should be familiar with web development concepts. Code examples mainly cover the usage of available open source Ruby libraries, such as Ruby on Rails, ActiveRecord, Sinatra, Nokogiri, and Typhoeus. If you are new to Ruby, you should be able to absorb the material as long as you have covered the language basics elsewhere and are generally familiar with web development. While the topic of service-oriented design is usually targeted at application architects, this book aims to present the material for regular web developers to take advantage of service-based approaches.

If you are interested in how Ruby can play a role in combining multiple pieces within an enterprise application stack, you will find many examples in this book to help achieve your goals. Further, if you are a Rails developer looking to expand the possibilities of your environment beyond a single monolithic application, you will see how this is not only possible but desirable. You can create systems where larger teams of developers can operate together and deploy improvements without the problem of updating the entire application at large.

The sections on API design, architecture, and data backends examine design principles and best practices for creating services that scale and are easy to interface with for internal and external customers. Sections on connecting to web services and parsing responses provide examples for those looking to write API wrappers around external services such as SimpleDB, CouchDB, or third-party services, in addition to internal services designed by the developer.

What This Book Covers

This book covers Ruby libraries for building and consuming RESTful web services. This generally refers to services that respond to HTTP requests. Further, the APIs of these services are defined by the URIs requested and the method (GET, PUT, POST, DELETE) used. While the focus is on a RESTful approach, some sections deviate from a purist style. In these cases, the goal is to provide clarity for a service API or flexibility in a proposed service.

The primary topics covered in this book are as follows:

- REST, HTTP verbs, and response codes
- API design

- Building services in Ruby
- Connecting to services
- Consuming JSON- and XML-based services
- Architecture design
- Messaging and AMQP
- Securing services

What This Book Doesn't Cover

Service-oriented architectures have been around for over a decade. During this time, many approaches have been taken. These include technologies with acronyms and buzzwords such as SOAP, WSDL, WS-*, and XML-RPC. Generally, these require greater overhead, more configuration, and the creation of complex schema files. Chapter 9, “Parsing XML for Legacy Services,” provides brief coverage of consuming XML and SOAP services. However, SOAP, XML-RPC, and related technologies are beyond the scope of this book. The services you'll create in this book are lightweight and flexible, like the Ruby language itself.

This book also does not cover other methods for building complex architectures. For example, it does not cover batch processing frameworks such as MapReduce or communications backends such as Thrift. While these technologies can be used in conjunction with a web services approach, they are not the focus. However, Chapter 11, “Messaging,” briefly covers messaging systems and message queues.

Additional Resources

Code examples are used heavily throughout this book. While every effort has been made to keep examples current, the open source world moves fast, so the examples may contain code that is a little out-of-date. The best place to find up-to-date source code is on GitHub, at the following address:

<http://github.com/pauldix/service-oriented-design-with-ruby>

In addition, you can subscribe to a mailing list to discuss the code, text, services design, and general questions on the topic of service-oriented design. You can join here:

<http://groups.google.com/group/service-oriented-design-with-ruby>

This page intentionally left blank

Acknowledgments

An unbelievable number of people contributed to this book through writing, editing, conversations about the content, or just moral support, so please forgive me if I leave anyone out. First, I need to thank Lindsey for putting up with my ridiculous schedule while writing this book and working a full-time job. Thanks to Trotter Cashion for writing Chapter 10, “Security”; to Bryan Helmkamp for writing Chapter 8, “Load Balancing and Caching”; and to Jake Howerton for writing Chapter 12, “Web Hooks and External Services.” Thanks to Trotter again and Jennifer Linder for providing excellent editing work and making sure that it makes sense. Thanks to Debra, my editor at Addison-Wesley, and to Michael, my development editor at AW. Thanks to the NYC.rb crew for being smart, fun people to hang out and discuss ideas with. Thanks to the entire team at KnowMore for putting up with me while I wrote and helping me refine my thinking. Finally, thanks to my business partner, Vivek, for providing encouragement during the final editing stages.

This page intentionally left blank

About the Author

Paul Dix is co-founder and CTO at Market.io. In the past, he has worked at Google, Microsoft, McAfee, Air Force Space Command, and multiple startups, filling positions as a programmer, software tester, and network engineer. He has been a speaker at multiple conferences, including RubyConf, Goruco, and Web 2.0 Expo, on the subjects of service-oriented design, event-driven architectures, machine learning, and collaborative filtering. Paul is the author of multiple open source Ruby libraries. He has a degree in computer science from Columbia University.

This page intentionally left blank

CHAPTER 1

Implementing and Consuming Your First Service

In the grand tradition of programming books beginning with a “hello, world” example, this book starts off with a simple service. This chapter walks through the creation of a service to store user metadata and manage authentication. This includes building the web service to handle HTTP requests and the client library for interacting with the service.

What’s a Service?

In this book, *service* generally refers to a system that responds to HTTP requests. Such HTTP requests are usually to write, retrieve, or modify data. Examples of public-facing HTTP services include Twitter’s API (<http://apiwiki.twitter.com>), the Amazon S3 service (<http://aws.amazon.com/s3/>), the Delicious API (<http://delicious.com/help/api>), the Digg API (<http://apidoc.digg.com>), and the New York Times APIs (<http://developer.nytimes.com/docs>). Internally, services could exist to contain pieces of data and business logic that are used by one or more applications.

Using a broader scope of definition, *service* can refer to a system that provides functionality through a standard interface. Working at this level of abstraction are services such as relational databases (for example, MySQL), Memcached servers, message queues (for example, RabbitMQ), and other types of data stores, such as Cassandra (<http://incubator.apache.org/cassandra/>).

While this book touches on the broader definition of *service* in a few places, the majority of the material focuses on HTTP-based services. More specifically, this book focuses on services that are designed to roughly follow a RESTful paradigm, as described in the appendix, “RESTful Primer.” Further, this book focuses on using services within an organization and infrastructure to build out applications. These services may or may not be public facing, like the previous examples.

The details of why, when, and how to use services are covered throughout the course of the book. For now the goal is to implement a simple service.

Service Requirements

A simple user management system is an example of something that can be pulled out as a service. After implementation, this service could be used across multiple applications within an organization as a single sign-on point. The goals and requirements of the service are fairly simple:

- Store user metadata, including name, email address, password, and bio.
- Support the full range of CRUD (create, update, delete) operations for user objects.
- Verify a user login by name and password.

In later versions of the user service, features could map which users work with each other, which user each user reports to, and which groups a user is a member of. For now, the basic feature set provides enough to work on.

The Ruby Tool Set

Ruby provides many tools to build both the service and client sides of services. However, this book heavily favors some specific tools due to their aesthetics or performance characteristics. The following libraries appear often throughout this book.

Sinatra

Sinatra is a lightweight framework for creating web applications. It can be described as a domain-specific language for web applications and web services. Built on top of Rack, Sinatra is perfectly suited for creating small web services like the example in this chapter. In addition to encouraging an elegant code style, Sinatra has fewer than 2,000

- [David Starr: Space Ranger pdf](#)
- [Computer Algorithms book](#)
- [click A Little Bit of Buddha: An Introduction to Buddhist Thought pdf](#)
- [Dictionary of Architecture and Building Construction pdf, azw \(kindle\), epub, doc, mobi](#)
- [click Moonseed \(NASA Trilogy, Book 3\) book](#)
- [C++ Quick Syntax Reference for free](#)

- <http://unpluggedtv.com/lib/David-Starr--Space-Ranger.pdf>
- <http://honareavalmusic.com/?books/Fundamentals-of-Kinematics-and-Dynamics-of-Machines-and-Mechanisms.pdf>
- <http://diy-chirol.com/lib/The-Sinister-Pig.pdf>
- <http://www.khoi.dk/?books/Perfidia--A-novel.pdf>
- <http://www.shreesaiexport.com/library/Measure-and-Category--A-Survey-of-the-Analogies-between-Topological-and-Measure-Spaces--Graduate-Texts-in-Mathe>
- <http://reseauplatoparis.com/library/C---Quick-Syntax-Reference.pdf>