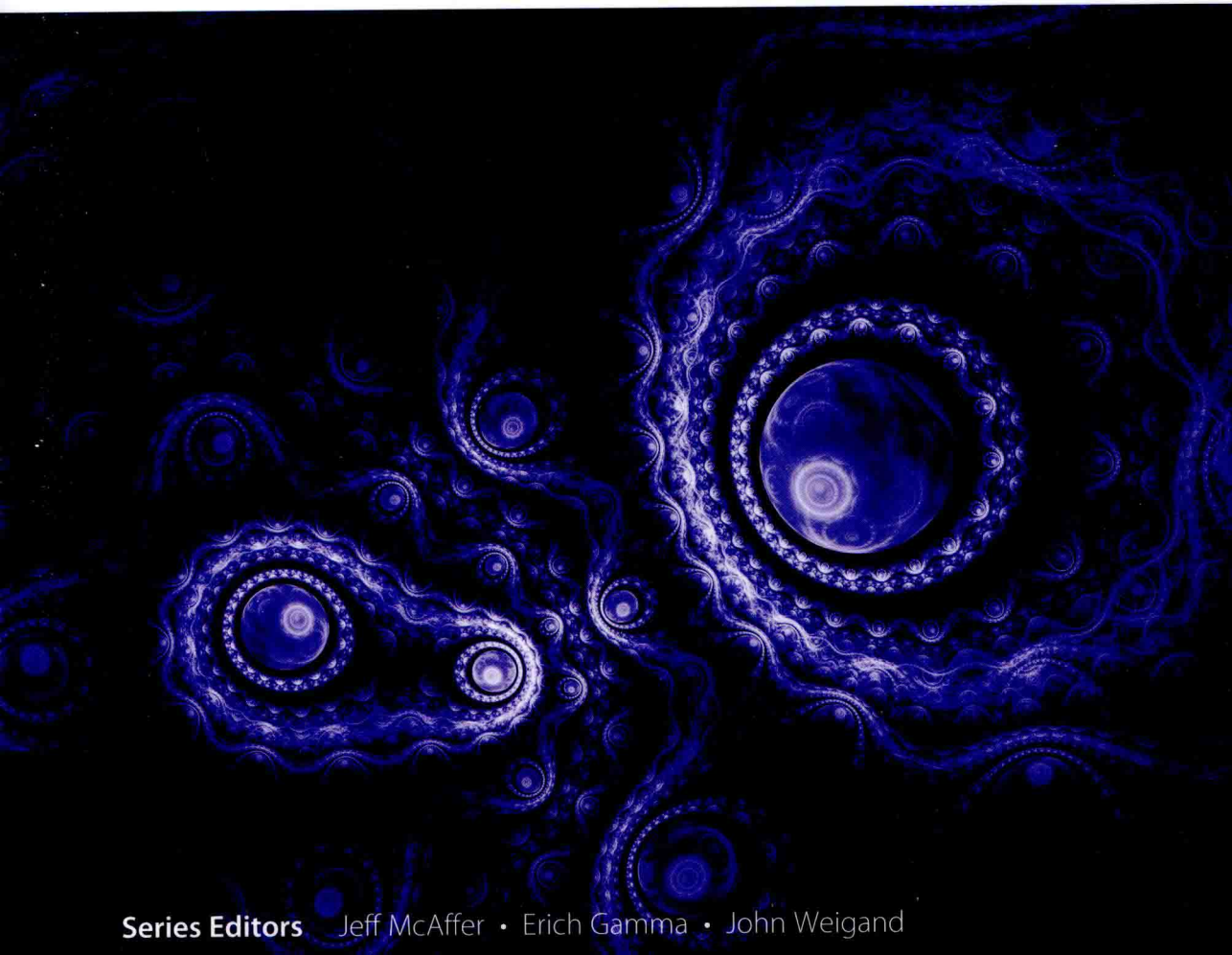
eclipseRT

the eclipse series

OSGi and Equinox

Creating Highly Modular Java™ Systems

Jeff McAffer • Paul VanderLei • Simon Archer



Series Editors Jeff McAffer • Erich Gamma • John Weigand

OSGi and Equinox

Creating Highly Modular Java™ Systems

Jeff McAffer
Paul VanderLei
Simon Archer

Addison-Wesley

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

Library of Congress Cataloging-in-Publication Data
OSGi and Equinox : creating highly modular Java systems / Jeff McAffer, Paul VanderLei, Simon Archer.
p. cm.
Includes index.
ISBN 0-321-58571-2 (pbk. : alk. paper)
1. Java (Computer program language) 2. Computer software—Development.
I. VanderLei, Paul. II. Archer, Simon (Simon J.) III. Title.
QA76.73.J38M352593 2010
005.2'762—dc22

2009047201

ISBN-13: 978-0-321-58571-4
ISBN-10: 0-321-58571-2
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
First printing February 2010

OSGi and Equinox

To my brother Ray
—Jeff McAffer

To Elizabeth and our four bundles:
Andrew, Bryant, Maria, and Josie
—Paul VanderLei

To my parents for their continual support
and encouragement in all my endeavors
—Simon Archer

Foreword

My role as the Chief Technology Officer of SpringSource brings me into frequent contact with companies building enterprise applications: many familiar names from the Fortune 500, and a whole host of others besides. If there is one thing you quickly learn, it is that the world of enterprise applications is messy and complex. Even four to five years ago, customers adopting Spring were asking us for ways to help them manage the size and complexity of the applications they were building. Large team sizes and applications with hundreds or thousands of internal components (Spring beans) were not uncommon. The pressures on enterprises to deliver increasingly sophisticated applications, in shorter and shorter time frames, have only been growing since then. In many cases applications are now always live and are constantly evolving. The move to deliver software "as a service"—internally or externally—can only accelerate this trend.

In the enterprise Java landscape, the traditional unit of deployment for an enterprise application is a web application archive (WAR) file. A number of common themes arise in my discussions with enterprise development teams:

- The WAR file as a single large unit of packaging and deployment is slowing down development processes and making it more difficult to structure large development teams since everything must come together in a single packaging step before anything can be deployed.
- WAR files are getting too large and unwieldy—a typical enterprise application may have literally hundreds of third-party dependencies, all packaged inside the WAR file. This has an adverse effect on upload and deployment times.
- Attempting to tackle complexity by deploying multiple WAR files side by side in the same container leads to problems with heap usage in the JVM since each WAR file has its own copy of all the dependencies, even though many of them could in theory be shared.
- When deploying WAR files side by side, there is no easy way to share common services.

- The WAR file as the smallest unit of change means that changes in large enterprise applications cannot be easily isolated and contained.
- Attempts to introduce "self-policed" (i.e., unenforced) modularity constraints into a design typically fail, despite best intentions.

To help manage the large team sizes and complex requirements of modern enterprise applications, it is clear that we need a more principled way to "divide and conquer." Something that lets us encapsulate well-defined parts of the system as modules with hidden internals and carefully managed externals. Something that enables those modules to be packaged and deployed individually without forcing us to revise the whole universe. Something that provides a principled mechanism for bringing those modules together in a running system, and that can cope with the changes introduced by continuous evolution.

Facing these requirements back in 2005, it was an easy decision at SpringSource (then Interface21) to turn to OSGi, the "dynamic module system for Java," as the foundation technology for modular enterprise applications. Even then, the OSGi Service Platform was already mature and proven in industrial settings, as well as being lightweight through its heritage in embedded systems.

The modularity layer of OSGi provides a mechanism for dividing a system into independent modules, known as bundles, that are independently packaged and deployed and have independent lifecycles. This solved a part of the problem for us—helping to keep the implementation types of a module private, and exposing only types that form part of the public interface of a module. We wanted enterprise developers to continue developing their applications using Spring, of course, and through the Spring Dynamic Modules' open-source project created a simple model whereby each module had its own set of components (Spring beans). Some of those components are private to the module, but some should be made public so that components in other modules can use them. The OSGi service layer provides an answer to this problem, promoting an in-memory service-oriented design. Components from a module can be published in the OSGi service registry, and from there other modules can find and bind to those services. OSGi also provides the necessary primitives to track services that may come and go over time as modules are installed, uninstalled, and upgraded.

The next stage in our journey with OSGi was the introduction of the SpringSource dm Server: an enterprise application server that is not only built on top of OSGi, but critically also supports the deployment of applications developed as a set of OSGi bundles. Spring Dynamic Modules works with any compliant OSGi Service Platform implementation, but for the dm Server we had to choose an OSGi Service Platform as the base on which to build. We chose to build on Equinox, the Eclipse implementation of the OSGi Service Platform, and also the reference

implementation for the core OSGi specification. The open-source nature of Equinox fit well with our own open-source philosophy and has been invaluable in enabling us to work closely with the developers of Equinox and submit patches and change requests over time. The widespread adoption of Equinox (as the underpinnings of Eclipse, to name but one example) gave us confidence that it would be battle-hardened and ready for enterprise usage.

I am seeing a strong and growing serious interest in OSGi among companies large and small. Building on OSGi will provide a firm foundation for dividing your application into modules, which in turn will help you structure the team(s) working on it more effectively. "Organization follows architecture" in the sense that your ability to divide a complex application into independent pieces also facilitates the structuring of team responsibilities along the same lines. In other scenarios, your teams may be fixed, and you need an architecture that enables those teams to work together most effectively. Again, a principled basis for dividing a system into modules can facilitate that. With OSGi as a basis, your unit of packaging and deployment can become a single module, removing bottlenecks in the process and helping to minimize the impact of change. OSGi is also incredibly well suited to product-line engineering, and to situations where you need to provide an extension or plug-in mechanism to enable third parties to extend your software.

The future for OSGi looks bright. Version 4.2 of the specification has just been released, and the OSGi Core Platform and Enterprise Expert Groups are very active. A glance at the membership of the OSGi Alliance and the composition of the expert groups tells you just how seriously enterprise vendors are taking it. I am confident that the investment of your time in reading and studying this book will be well rewarded. It is my belief that OSGi is here to stay. A firm grasp of the strengths—and the weaknesses—of the OSGi Service Platform will prove invaluable to you on your journey toward creating agile, modular software.

—Adrian Colyer
CTO, SpringSource
October 2009

Preface

OSGi is a hot topic these days; all the major Java application server vendors have adopted OSGi as their base runtime, Eclipse has been using OSGi as the basis of its modularity story and runtime for at least the past five years, and countless others have been using it in embedded and "under the covers" scenarios. All with good reason.

The success of Eclipse as a tooling platform is a direct result of the strong modularity enshrined in OSGi. This isolates developers from change, empowers teams to be more agile, allows organizations to change the way that they develop software, and lubricates the formation and running of ecosystems. These same benefits can be realized in any software domain.

The main OSGi specification is remarkably concise—just 27 Java types. It is well designed, and specified to be implemented and used in real life. Adoption of OSGi is not without challenges, however. Make no mistake: Implementing highly modular and dynamic systems is hard. There is, as they say, no free lunch. Some have criticized OSGi as being complicated or obtuse. In most cases it is the problem that is complex—the desire to be modular or dynamic surfaces the issues but is not the cause. Modularizing existing monolithic systems is particularly challenging.

This book is designed to both highlight such topics and provide knowledge, guidance, and best practices to mitigate them. We talk heavily of modularity, components, and dynamism and show you techniques for enhancing your system's flexibility and agility.

Despite using OSGi for many years, participating in writing the OSGi specifications, and implementing Equinox (the OSGi framework specification reference implementation), during the writing of this book we learned an incredible amount about OSGi, Equinox, and highly modular dynamic systems. We trust that in reading it you will, too.

About This Book

This book guides up-and-coming and established OSGi developers through all stages of developing and delivering an example OSGi-based telematics and fleet management system called *Toast*.

We develop Toast from a blank workspace into a full-featured client and server system. The domain is familiar to most everyone who has driven a car or shipped a package. Telematics is, loosely speaking, all the car electronics—radio, navigation, climate control, and so on. Fleet management is all about tracking and coordinating packages and vehicles as they move from one place to another.

The set of problems and opportunities raised allows us to plausibly touch a wide range of issues from modularity and component collaboration to server-side programming and packaging and delivery of highly modular systems. We create stand-alone client applications, embedded and stand-alone server configurations, and dynamic enhancements to both. This book enables you to do the same in your domain.

Roughly speaking, the book is split into two sections. The first half, Parts I and II, sets the scene for OSGi and Equinox and presents a tutorial-style guide to building Toast. The tutorial incrementally builds Toast into a functioning fleet management system with a number of advanced capabilities. The tutorial is written somewhat informally to evoke the feeling that we are there with you, working through the examples and problems. We share some of the pitfalls and mishaps that we experienced while developing the application and writing the tutorial.

The second half of the book looks at what it takes to "make it real." It's one thing to write a prototype and quite another to ship a product. Rather than leaving you hanging at the prototype stage, Part III is composed of chapters that dive into the details required to finish the job—namely, the refining and refactoring of the first prototype, customizing the user interface, and building and delivering products to your customers. This part is written as a reference, but it still includes a liberal sprinkling of step-by-step examples and code samples. The goal is both to dive deep and cover most of the major stumbling blocks reported in the community and seen in our own development of professional products.

A final part, Part IV, is pure reference. It covers the essential aspects of OSGi and Equinox and touches on various capabilities not covered earlier in the book. We also talk about best practices and advanced topics such as integrating third-party code libraries and being dynamic.

OSGi, despite being relatively small, is very comprehensive. As such, a single book could never cover all possible topics. We have focused on the functions and services that we use in the systems we develop day to day under the assumption that they will be useful to you as well.

OSGi, Equinox, and EclipseRT

The OSGi community is quite vibrant. There are at least three active open-source framework implementation communities and a wide array of adopters and extenders. The vast majority of this book covers generic OSGi topics applicable to any OSGi system or implementation. Throughout the book we consistently use Equinox, the OSGi framework specification reference implementation, as the base for our examples and discussions. From time to time we cover features and facilities available only in Equinox. In general, these capabilities have been added to Equinox to address real-world problems—things that you will encounter. As such, it is prudent that we discuss them here.

Throughout the book we also cover the Eclipse Plug-in Development Environment (PDE) tooling for writing and building OSGi bundles. PDE is comprehensive, robust, and sophisticated tooling that has been used in the OSGi context for many years. If you are not using PDE to create your OSGi-based systems, perhaps you should take this opportunity to find out what you are missing.

Finally, Eclipse is a powerhouse in the tooling domain. Increasingly it is being used in pure runtime, server-side, and embedded environments. This movement has come to be known as *EclipseRT*. EclipseRT encompasses a number of technologies developed at Eclipse that are aimed at or useful in typical runtime contexts. The Toast application developed here has been donated to the Eclipse Examples project and is evolving as a showcase for EclipseRT technologies. We encourage you to check out http://wiki.eclipse.org/Toast to see what people have done to and with Toast.

Audience

This book is targeted at several groups of Java developers. Some Java programming experience is assumed, and no attempt is made to introduce Java concepts or syntax.

For developers new to OSGi and Equinox, there is information about the origins of the technology, how to get started with the Eclipse OSGi bundle tooling, and how to create your first OSGi-based system. Prior experience with Eclipse as a development tool is helpful but not necessary.

For developers experienced with writing OSGi bundles and systems, the book formalizes a wide range of techniques and practices that are useful in creating highly modular systems using OSGi—from service collaboration approaches to server-side integration and system building as part of a release engineering process, deployment, and installation.

For experienced OSGi developers, this book includes details of special features available in Equinox and comprehensive coverage of useful facilities such as

Declarative Services, buddy class loading, Google Earth integration, and the Eclipse bundle tooling that make designing, coding, and packaging OSGi-based systems easier than ever before.

Sample Code

Reading this book can be a very hands-on experience. There are ample opportunities for following along and doing the steps yourself as well as writing your own code. The companion download for the book includes code samples for each chapter. Instructions for getting and managing these samples are given in Chapter 3, "Tutorial Introduction," and as needed in the text. In general, all required materials are available online at either http://eclipse.org or http://equinoxosgi.org. As mentioned previously, a snapshot of Toast also lives and evolves as an open-source project at Eclipse. See http://wiki.eclipse.org/Toast.

Conventions

The following formatting conventions are used throughout the book:

Bold—Used for UI elements such as menu paths (e.g., **File > New > Project**) and wizard and editor elements

Italics—Used for emphasis and to highlight terminology

`Lucida`—Used for Java code, property names, file paths, bundle IDs, and the like that are embedded in the text

`Lucida Bold`—Used to highlight important lines in code samples

Notes and sidebars are used often to highlight information that readers may find interesting or helpful for using or understanding the function being described in the main text. We tried to achieve an effect similar to that of an informal pair-programming experience where you sit down with somebody and get impromptu tips and tricks here and there.

Feedback

The official web site for this book is http://equinoxosgi.org. Additional information and errata are available at informit.com/title/0321585712. You can report problems or errors found in the book or code samples to the authors at book@equinoxosgi.org. Suggestions for improvements and feedback are also very welcome.

Acknowledgments

It is impossible to write a book such as this without the cooperation and help of a vast number of people. In our case, virtually the entire Equinox team contributed directly to the end result through conversations, help with code and concepts, bug fixes, manuscript review, or just general support.

A few individuals contributed exceptional amounts of time and brain-power to this project, and we extend our heartfelt thanks to them here:

Tom Watson—Tom is the driving force behind Equinox and is very active in the OSGi specification community. His pragmatic approach and level head have brought you Equinox and us a guiding hand in the creation of this material.

Chris Aniszyzck—Chris has brought his diverse passions to bear on PDE, the tooling that makes OSGi and Equinox a pleasure to program. The creation of this book drove many new use cases and requirements. Chris eagerly pushed PDE to be even more of a bundle development environment, making life easier for all of us.

Ian Bull—Ian applied his pedagogical skill and attention to detail on all things related to p2, packaging, and building, making the whole process of building and delivering OSGi functionality tractable.

Stoyan Boshev—Stoyan is the guiding hand behind the Equinox Declarative Services implementation. DS figures heavily in this book and the sample code. Stoyan spent countless hours implementing DS and working with us to bring its power to you.

A number of people provided portions of the book's sample code or in-depth review and guidance on technical elements of the content. In particular, DJ Houghton and Scott Admiraal completed exhaustive testing and review of the tutorial sections, saving our behinds in the process. Rafael Oliveira Nóbrega and Chris Aniszyzck contributed hugely to the creation of Declarative Services tooling, making DS usable by mere mortals. Andrew Niefer, Pascal Rapicault, Simon Kaegi,

and Scott Lewis all contributed fixes, samples, and guidance on technologies ranging from PDE Build and p2 to server-side OSGi to ECF. Patrick Dempsey contributed the Crust code and offered tireless support on all things Mac-related. BJ Hargrave, the steady hand of OSGi, patiently discussed any number of design points, best practices, and coding approaches.

We were also fortunate to have the Eclipse community and a number of people who reviewed chapters or provided valuable input and help. These include Joel Rosi-Schwartz, Benjamin Muskalla, Kevin Barnes, Grant Gayed and the SWT team, Ralf Sternberg, Matt Flaherty, the readers of the early drafts on Rough Cuts, and all the people involved in developing the Toast example code.

Of course, no book project is possible without a publishing team. We were lucky to have Greg Doench as the enduring editor of the Eclipse Series along with Michelle Housley, Barbara Wood, Elizabeth Ryan, and the whole crew at Addison-Wesley who made this a relatively painless and quite enjoyable experience.

The authors would like to individually acknowledge the following people:

Jeff McAffer: Nancy, Sydney, and Toby, you are the loves of my life. Mom, Dad, and Val, I love you fiercely; you made me what I am today and I am thankful. The entire EclipseSource team, thanks for giving me the room to move and being generally enthusiastic around Toast and this project.

Paul VanderLei: I'd like to thank my partners at Band XI International—John Cunningham, Brett Hackleman, Patrick Dempsey, and James Branigan—for generously providing me the time to complete this project. Thanks, too, to my wife and children for their patience and love. Finally, I'm forever grateful to my father, whose encouragement and sage counsel have shaped my entire career.

Simon Archer: Undertaking to write a book such as this involves a huge amount of time, dedication, and sacrifice. While I am grateful for my coauthors, Jeff and Paul, for their time and dedication to this project, it is to my wife, Lisa, and my children, Thomas and Emma, that I owe the most gratitude since they are the ones who made all the sacrifices. Thank you for your constant love and support and for allowing me the time to work on the book—I am forever in your debt.

Beyond this book, OSGi would not be what it is today without the following people:

BJ Hargrave—BJ is the CTO at the OSGi Alliance and has been driving the technology since the beginning. He was the lead for the IBM OSGi implementation, SMF, that was donated to Eclipse as the forerunner of Equinox. He continues to promote and guide OSGi as it evolves beyond its original domain.

Peter Kriens—Peter is the OSGi Evangelist and a longtime leader of the OSGi community. He fulfills his evangelical role with style and energy that are inspiring. The continuity and clarity that we see in the OSGi specifications are a direct result of Peter's editorial and design skill.

Tom Watson—Tom is the co-lead and heavy lifter in the Equinox OSGi project at Eclipse and a valued member of the OSGi expert groups. He is responsible for the entire framework implementation and many of the add-on facilities. His pragmatism and thoroughness have made Equinox what it is today.

Richard Hall—Richard is the lead of the Apache Felix project and is very active in the OSGi specification process. Felix is an evolution of the Oscar project, the first open-source OSGi framework implementation and an inspiration to the Equinox team as they looked to adopt OSGi. The alternative viewpoint provided by the Felix project continues to enrich the specification and implementation process.

About the Authors

Jeff McAffer co-leads the Eclipse RCP and Equinox OSGi projects and is CTO and cofounder of EclipseSource. He is one of the architects of the Eclipse Platform and a coauthor of *The Eclipse Rich Client Platform* (Addison-Wesley). He co-leads the RT PMC and is a member of the Eclipse Project PMC, the Tools Project PMC, and the Eclipse Architecture Council and a former member of the Eclipse Foundation Board of Directors. Jeff is currently interested in all aspects of Eclipse components, from developing and building bundles to deploying, installing, and ultimately running them. Previous lives include being a Senior Technical Staff Member at IBM; being a team lead at Object Technology International covering work in Smalltalk, distributed/parallel OO computing, expert systems, and metalevel architectures; and getting a Ph.D. from the University of Tokyo.

Paul VanderLei is a partner at Band XI International. He has more than twenty-five years of software engineering experience with an emphasis on object-oriented design and agile practices. He is well known for his innovative yet straightforward engineering solutions to complex problems. After earning his M.S. in computer science from Arizona State University, he joined Object Technology International and worked on a wide range of Smalltalk-based systems. After OTI's acquisition by IBM, Paul developed embedded Java applications and user interfaces for the automotive and medical industries as a founding member of IBM's Embedded Java Enablement Team. He has been using OSGi in commercial applications for over ten years. He lives in Grand Rapids, Michigan, with his wife and four children.

Simon Archer has more than sixteen years of software engineering experience with an emphasis on object-oriented design, agile practices, and software quality. After earning his B.Sc. in computer science from the University of Portsmouth, UK, he worked as a Smalltalk developer at Knowledge System Corporation and later at Object Technology International. While at OTI in 2000, Simon began working

with and teaching OSGi in areas such as telematics and RFID. Today he works for IBM Rational, using OSGi to build collaborative development tools for the Jazz Foundation project. He lives in Cary, North Carolina, with his wife and two children.