UML Components

A Simple Process for Specifying Component-Based Software

John Cheesman
John Daniels

Addison-Wesley
Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

LOC 000-61851

ISBN 0-201-70851-5
Text printed on recycled paper
1 2 3 4 5 6 7 8 9 10—CRS—0403020100
First printing, October 2000

Contents

Foreword

Component software is finally taking off in a big way. Several companies now focus solely on resale, brokering, and consulting services around software components. Yet the practitioner is left with a confusing array of technologies and with mismatching methods and processes. Geared toward object-oriented analysis and design, the all-important component concept has fallen by the wayside in many object-centric approaches. Worse yet, there is the occasional thesis that components are special objects. Clearly, there is nothing wrong with objects—but it is components, not objects, that promise industrial leverage of software production and composition.

John Cheesman and John Daniels address the practical problem of architecting and specifying component-based systems head-on. This compact and very approachable text will help many who find themselves wedged between the need to utilize component technologies—currently mostly EJB and COM+—and the desire to apply the rich concepts of object modeling in general and UML in particular. Their pragmatic extension of UML captures important component concepts: component specifications, component interfaces, component implementations, and finally (generated by installed components) component objects. Only the last of these are objects.

Design by contract is the fundamentally simple and compelling idea to design systems as cooperating abstract boxes that achieve their common goal by relying on contracts. A contract specifies what each of the participants buying into collaboration has to do in order to benefit from the promised results. Contracts are formalizations of fair give-and-take

among strangers and it is not surprising that a contractual approach is nowhere as natural as in a component world. Cheesman and Daniels take the formalization of contracts to the necessary level of precision, using the Object Constraint Language (OCL) that forms part of the UML foundation. Component specifications name the interfaces that a component adhering to the specification must implement.

Contracts enrich the operations of an interface with pre- and postconditions. However, the interface level does not constrain how components may interact. A typical example is a requirement that a component instance, whenever it receives a call on one of its interfaces, should then call a certain method of some other instance. Another common example is restrictions on the order in which operations on interfaces of a component instance may be called. Component specifications provide the required additional information.

The main contribution of this book, however, is a very workable process that builds on a clear conceptualization and rests on the author's extensive practical experience. John Cheesman and John Daniels have been drivers of the field for many years. This book is based on their experience with contributions to many influential methods and approaches and their application in practice. Readers will enjoy their direct, no-nonsense style as much as they will appreciate the many practical words of advice.

Clemens Szyperski, July 2000

Preface

This book describes how to architect and specify enterprise-scale component-based systems.

It is a practical and technical book. The business benefits of a component-based approach to building systems have been well documented in many theoretical books and we don't repeat these. Our focus is on helping people move from the theory to the detailed reality.

It seems to us that people who want to take a model-based approach to the design and construction of enterprise-scale component-based software face two big problems. First, what tasks and techniques can they use that will both produce a good system and be compatible with whatever project management process is in use? Little has been written to date about processes that can support the construction of large component systems. Second, how should they use the wide range of notations and techniques found in the Unified Modeling Language (UML)? The UML has become the de facto standard for pretty much all application development modeling, but its application to component-based approaches isn't obvious.

If you flick through the pages it might seem to you that we've concentrated mainly on the second of these problems—there are lots of UML diagrams—but a deeper examination will show, we hope, that the primary emphasis is on explaining a simple process by which components can be identified and specified, and robust but flexible application architectures can be produced.

Of course, the full development process covers more than just specification; it covers all activities from requirement gathering to system deployment. But this book focuses on specification. It explains how to represent

requirements in a way that will facilitate the construction of specifications, it shows how to create specifications, and it gives guidance on implementing the specifications in software. We make no apology for focusing on specification. The main challenge that a component approach can meet is dealing with change, but the substitutability of parts this requires can be achieved only if components are properly specified.

Underpinning the process are a set of principles and definitions that organize and structure our thinking about software components. We have found these ideas to be a great help, and we urge you to take the time to understand and appreciate them. You'll find them set out in Chapter 1.

Who Should Read This Book?

We have written this book for practitioners—people who need to architect systems and specify components in UML today, using today's tools. We describe a clear process for moving from business requirements to system specifications and architectures. This will be helpful for those struggling with business-IT alignment in today's e-business world. The architecture pieces will assist those focusing on system architectures and assemblies, from city planning through detailed application architectures. The emphasis on unambiguous interface specification will be useful for those trying to establish software factories, those defining clear buy-and-build software policies, and those involved in application integration and legacy migration. It should also appeal to testing and validation teams.

We also think this book contains sufficient conceptual clarity and succinct explanations of techniques to make it of interest to both academics and educators. We certainly hope they will buy it.

How Best to Read This Book

Start at page 1 and keep going. When you reach a page that's thicker than the rest and shiny on one side, you're done. Seriously though, this isn't a big book, and we think you'll get most from it if you read it all. We think

it's all important, so we can't suggest sections to skip on first reading, although you might find it useful the first time through only to skim some of the detailed specification examples, especially in Chapter 7. In fact, we tried to write the kind of book we like to read ourselves—lean and mean, with no unnecessary asides to distract from the main message.

Having read it all once, however sketchily, you will probably want to dip in and out of particular chapters as you're dealing with specific issues on your projects.

If you want to dig deeper into the examples we have provided, you can find the full case study at *http://www.umlcomponents.com*.

Where Did These Ideas Come From?

We'd like to think that the ideas in this book are all our own, but they're not. The component concepts and the process ideas we've used have been formed over a number of years and derive from a great many sources. We've relied heavily on the expertise of others who have struggled with—and solved, at least partially—related problems.

On John Cheesman's side the ideas come from his early work on the Microsoft Repository Open Information Model (OIM), in the mid-1990s; his work with Desmond D'Souza and Alan Cameron Wills on the Catalysis meta-model [D'Souza99]; UML, of course, to which he was a direct contributor; and Sterling Software's Advisor method for component-based development [Advisor], developed mainly by John Dodd and itself influenced by Catalysis.

John Daniels is one of the pioneers of object-oriented concepts and practices. In the early 1990s he developed, together with Steve Cook, the Syntropy method [Cook94]. This work has been a forerunner and common ancestor of many of the later developments mentioned above, especially Catalysis. The UML's Object Constraint Language (OCL) is directly descended from Syntropy, and several ideas first seen in Syntropy have found their way into the UML.

Figure P.1, although inevitably a simplification, gives some insight into how the ideas have influenced each other. Of course each of these areas

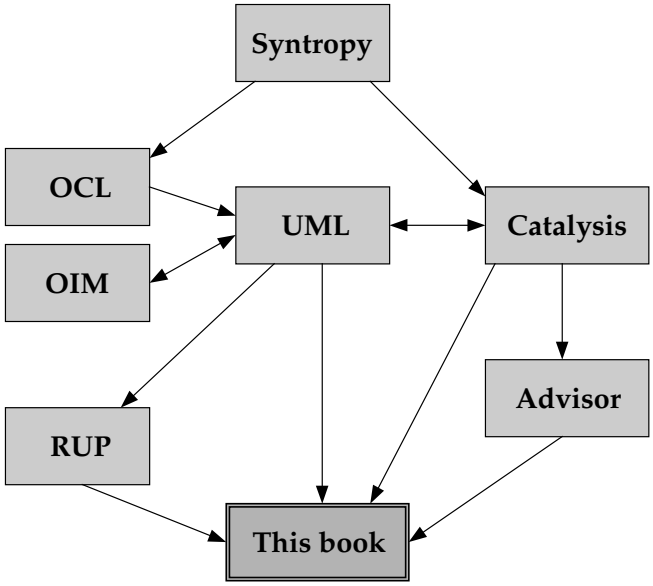

Figure P.1 Family tree

has its own set of influences, which we haven't shown, although OMT [Rumbaugh91] and Bertrand Meyer's notions of design by contract [Meyer00] deserve special mention.

We joined forces in 1999 to refine the concept models, tighten the process ideas, and align them with the workflows and terminology of the Rational Unified Process (RUP) [Jacobson99]. This book is the result.

Practical Experience

We have greatly benefited from studying the experiences of the Sterling Software component-based development (CBD) Customer Advisory Board (CAB). This is a set of companies who have been developing enterprise-scale component-based applications since 1996 in a variety of vertical domains, from telecoms to transportation, and finance to manu-

facturing. At the time of writing, the CBD CAB has around one hundred member companies.

These companies have helped to separate what works in practice and has genuine, practical added value from what sounds good but is impractical on a real project. They've kept our feet firmly on the ground.

What works in practice is often a function of how well a particular process or technique is supported by application development tools. And it has to be said, most UML tools don't do a great job of supporting component-based development. We avoid references to specific tools in this book since we want it to have a broad appeal, and we try to keep as close to standard UML as we can, defining a relatively small set of extensions. But clearly, the better your tool of choice supports these concepts and processes, the more practical you will find them.

Acknowledgments

As detailed earlier, we acknowledge the work of others that has provided the source for many of the ideas that we've brought together here.

We condensed all of these ideas into a cohesive whole during 1999, while we were both working at Sterling Software's Chertsey Lab in England. We would like to thank all the members of that team for their support and help. In particular, we would like to acknowledge the contribution of John Dodd, with whom we spent many hours debating the details of particular approaches and considering the practicability of some of the techniques.

The prize for keeping us practical must go to the Sterling Software CBD CAB.

Paul Harmon from the Cutter Consortium provided useful independent analysis of our CBD approach and our concept models.

Our formal reviewers provided timely, insightful, and constructive feedback, as well as encouragement that we were saying something useful and new. We would like to thank Paul Allen; John Dodd; Chris Lamela, IntellectMarket, Inc.; Pete McBreen; and Alan Cameron Wills, TriReme International Ltd.

Additional helpful comments and support came from Laura Hill, Bruce Anderson, and Richard Mitchell. Thanks also to James Noble for suggesting the title.

Thanks to Kristin Erickson at Addison-Wesley for her enthusiasm and support throughout, and to the team there for processing the book on a tight schedule.

John Cheesman
Surrey, England
johnc@componentsource.com

John Daniels
London, England
john@syntropy.co.uk

UML - Component Diagrams - Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, wh.Â Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.