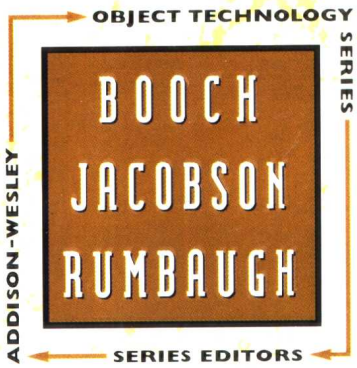
UML与面向对象设计影印丛书

EXECUTABLE UML 技术内幕

EXECUTABLE UML

A FOUNDATION FOR MODEL-DRIVEN ARCHITECTURE

STEPHEN J. MELLOR
MARC J. BALCER
编著



科学出版社
www.sciencep.com

UML与面向对象设计影印丛书

Executable UML 技术内幕

Stephen J. Mellor
Marc J. Balcer 编著

科学出版社
北京

内 容 简 介

Executable UML 是软件开发领域的一项重大发明，这方面的著作尚不多见。本书对这一技术做了深入的介绍，比如，怎样用 UML 将需求和用况物化成为直观的图表，如何用 UML 产生可执行、可测试的模型，如何将模型直接翻译成代码，以及如何用 Executable UML 模型编译器将分散的系统域编译在一起。为加深读者对有关概念和技巧的理解，书中还提供了一个开发成功的大型案例。另外，还提供了两个网址，以便于读者下载有关的模型以及翻译和运行这些模型的工具。

本书适合软件系统分析、设计人员阅读。

ISBN 0-201-74804-5

图字:01-2003-2537

图书在版编目(CIP)数据

Executable UML 技术内幕=Executable UML：A Foundation for Model-Driven Architecture/（美）梅勒（Mellor,S.J.），（美）巴尔塞（Balcer,M.J.）著.—影印本，—北京：科学出版社，2003

ISBN 7-03-011401-9

Ⅰ.E… Ⅱ.①…②巴… Ⅲ.面向对象语言—UML—程序设计—英文 Ⅳ.TP312

中国版本图书馆 CIP 数据核字（2003）第 030824 号

策划编辑：李佩乾/责任编辑：李佩乾
责任印制：吕春珉/封面设计：东方人华平面设计室

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com

双青印刷厂 印刷
科学出版社发行 各地新华书店经销

*

2003 年 5 月第 一 版 开本：787×960 1/16
2003 年 5 月第一次印刷 印张：25
印数：1—2 000 字数：478 000

定价：45.00 元

（如有印装质量问题，我社负责调换<环伟>）

影印前言

随着计算机硬件性能的迅速提高和价格的持续下降，其应用范围也在不断扩大。交给计算机解决的问题也越来越难，越来越复杂。这就使得计算机软件变得越来越复杂和庞大。20 世纪 60 年代的软件危机使人们清醒地认识到按照工程化的方法组织软件开发的必要性。于是软件开发方法从 60 年代毫无工程性可言的手工作坊式开发，过渡到 70 年代结构化的分析设计方法、80 年代初的实体关系开发方法，直到面向对象的开发方法。

面向对象的软件开发方法是在结构化开发范型和实体关系开发范型的基础上发展而来的，它运用分类、封装、继承、消息等人类自然的思维机制，允许软件开发者处理更为复杂的问题域和其支持技术，在很大程度上缓解了软件危机。面向对象技术发端于程序设计语言，以后又向软件开发的早期阶段延伸，形成了面向对象的分析和设计。

20 世纪 80 年代末 90 年代初，先后出现了几十种面向对象的分析设计方法。其中，Booch, Coad/Yourdon、OMT 和 Jacobson 等方法得到了面向对象软件开发界的广泛认可。各种方法对许多面向对象的概念的理解不尽相同，即便概念相同，各自技术上的表示法也不同。通过 90 年代不同方法流派之间的争论，人们逐渐认识到不同的方法既有其容易解决的问题，又有其不容易解决的问题，彼此之间需要进行融合和借鉴；并且各种方法的表示也有很大的差异，不利于进一步的交流与协作。在这种情况下，统一建模语言(UML)于 90 年代中期应运而生。

UML 的产生离不开三位面向对象的方法论专家 G. Booch、J. Rumbaugh 和 I. Jacobson 的通力合作。他们从多种方法中吸收了大量有用的建模概念，使 UML 的概念和表示法在规模上超过了以往任何一种方法，并且提供了允许用户对语言做进一步扩展的机制。UML 使不同厂商开发的系统模型能够基于共同的概念，使用相同的表示法，呈现彼此一致的模型风格。1997 年 11 月 UML 被 OMG 组织正式采纳为标准的建模语言，并在随后的几年中迅速地发展为事实上的建模语言国际标准。

UML 在语法和语义的定义方面也做了大量的工作。以往各种关于面向对象方法的著作通常是以比较简单的方式定义其建模概念，而以主要篇幅给出过程指导，论述如何运用这些概念来进行开发。UML 则以一种建模语言的姿态出现，使用语言学中的一些技术来定义。尽管真正从语言学的角度看它还有许多缺陷，但它在这方面所做的努力却是以往的各种建模方法无法比拟的。

从 UML 的早期版本开始，便受到了计算机产业界的重视，OMG 的采纳和大公司的支持把它推上了实际上的工业标准的地位，使它拥有越来越多的用户。它被广泛地用

于应用领域和多种类型的系统建模，如管理信息系统、通信与控制系统、嵌入式实时系统、分布式系统、系统软件等。近几年还被运用于软件再工程、质量管理、过程管理、配置管理等方面。而且它的应用不仅仅限于计算机软件，还可用于非软件系统，例如硬件设计、业务处理流程、企业或事业单位的结构与行为建模，等等。

在 UML 陆续发布的几个版本中，逐步修正了前一个版本中的缺陷和错误。即将发布的 UML2.0 版本将是对 UML 的又一次重大的改进。将来的 UML 将向着语言家族化、可执行化、精确化等理念迈进，为软件产业的工程化提供更有力的支撑。

本丛书收录了与面向对象技术和 UML 有关的 12 本书，反映了面向对象技术最新的发展趋势以及 UML 的新的研究动态。其中涉及对面向对象建模理论研究与实践的有这样几本书：《面向对象系统架构及设计》主要讨论了面向对象的基本概念、静态设计、永久对象、动态设计、设计模式以及体系结构等近几年来面向对象技术领域中的新的理论知识与方法；《用 UML 进行用况对象建模》主要介绍了面向对象的需求阶段、分析阶段、设计阶段中用况模型的建立方法与技术；《高级用况建模》介绍了在建立用况模型中需要注意的高级的问题与技术；《UML 面向对象设计基础》则侧重于经典的面向对象理论知识的阐述。

涉及 UML 在特定领域的运用的有这样几本：《UML 实时系统开发》讨论了进行实时系统开发时需要对 UML 进行扩展的技术；《用 UML 构建 Web 应用程序》讨论了运用 UML 进行 Web 应用建模所应该注意的技术与方法；《面向对象系统测试：模型、视图与工具》介绍了将 UML 应用于面向对象的测试领域所应掌握的方法与工具；《对象、构件、框架与 UML 应用》讨论了如何运用 UML 对面向对象的新技术——构件-框架技术建模的方法策略。《UML 与 Visual Basic 应用程序开发》主要讨论了从 UML 模型到 Visual Basic 程序的建模与映射方法。

介绍面向对象编程技术的有两本书：《COM 高手心经》和《ATL 技术内幕》，深入探讨了面向对象的编程新技术——COM 和 ATL 技术的使用技巧与技术内幕。

还有一本《Executable UML 技术内幕》，这本书介绍了可执行 UML 的理念与其支持技术，使得模型的验证与模拟以及代码的自动生成成为可能，也代表着将来软件开发的一种新的模式。

总之，这套书所涉及的内容包含了对软件生命周期的全过程建模的方法与技术，同时也对近年来的热点领域建模技术、新型编程技术作了深入的介绍，有些内容已经涉及到了前沿领域。可以说，每一本都很经典。

有鉴于此，特向软件领域中不同程度的读者推荐这套书，供大家阅读、学习和研究。

北京大学计算机系　蒋严冰 博士

Foreword

Creating a modeling language that is also an executable language has long been a goal of the software community. Many years ago, in 1968 to be exact, while working with software components to successfully develop a telecommunications system, we created a modeling language that was the forerunner to UML. To model components we used sequence diagrams, collaboration diagrams, and state transition diagrams (a combination of state charts and activity diagrams). Our modeling language then seamlessly translated the component models into code. Each code component was in its turn compiled into an executable component that was deployed in our computer system. The computer system was a component management system—thus we had "components all the way down."

Thanks to the seamless relation between the modeling language and the programming language, almost 90% of the code could be generated. Changes in the code could translate into models. However, this had to be done manually, because at that time we didn't have the tools to do the job. It was almost a clerical event. We called it a job for monkeys. Of course, I asked myself the question, if we only need the programming environment to create 10% of the code, do we really need two languages? Couldn't we integrate the modeling language and the programming language and make one language with a visual as well as a textual syntax and with common semantics. Having such a powerful language would dramatically

eliminate work and it would make life more enjoyable to the developers. In 1980, I suggested this move in writing as a next step in the development of our products.

Several years latter, the International Telecommunication Union (ITU), headquartered in Geneva, created a standard for object modeling, known as the Specification and Description Language (or SDL). SDL was very much inspired by the modeling language we were using in developing our telecommunication system. We referred to it as "The Ericsson Language". In the early 1980s we added to SDL constructs to formally define algorithms and data structures. It was very simple, but at last we had a language that would allow us to execute our design models long before these models had been translated into a programming environment.

Steve Mellor and Marc Balcer now move the concept of "executableness" a step further. Steve in particular, advances the idea of making an abstract, platform independent, executable model of a software system which then, with specially designed tools (bridges), generates or transforms the model into executable software to run on target computer systems. The specially designed tools transform platform independent models to platform dependent code.

Steve has long been an advocate of this idea. I remember that Steve was on a panel discussing this several years ago. He was met by a lot of skepticism by other panelists. After the panel I told him, "There is in my mind no doubt that you will be proven right—in particular for a small stereotypical class of systems, but it may take 25 years to get there for the majority of software development." After five more years and all the work done by Steve and others, I think we can get there even sooner.

Steve has now moved his work from Shlaer-Mellor notation to UML as the base modeling language. He has been the driving force behind the work on action semantics in UML, which was the missing piece to make UML an executable language. This is now part of the OMG standard for UML. He has gone even further. His work on transforming from a platform independent executable model to executable code via bridges is one of the cornerstones on which rests the OMG's new initiative: Model Driven Architecture (MDA).

However, to quote Winston Churchill, "This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning." Having an executable UML for modeling platform independent software is a great step forward. We can work on abstract models, validate (debug if you want) them early before we introduce the platform dependencies and make sure that the system behaves functionally as expected. However, I believe we need to take one step further to eliminate a very expensive impedance in software development. We should move forward and design the missing link to make UML the language to also execute platform dependent software, to make UML a third generation language that eventually will replace existing programming languages. This is not a technical problem. This could be done today if the big platform vendors wanted to do it. We would eliminate the two-language problem (having both a modeling language, UML, and a programming language like C# or Java). We would get a language that would be both. A language that would be used for use cases, for platform-independent design, for platform dependent designs whether this would be done by transformers as Steve advocates or by doing the job by interconnecting components—some of which being new, some of them already being harvested.

Getting an executable UML—to be used both for platform independent models as Steve and Marc describe in this book and for platform dependent ordinary source code—will be an important step in the future of software development.

Ivar Jacobson

Preface

At one time, the title for this book was *Executable UML For Model-Driven Architectures (MDA) Using Aspect-Oriented (AO) Techniques with Extreme Programming (XP), Agile Modeling (AM), and Other Agile Alliance (AA) Processes as an Instance of the Rational Unified Process (RUP).*

Eventually, we settled instead on *Executable UML: A Foundation for Model-Driven Architecture.* This title is snappier, but it's not as fully buzzword-compliant as the original.

So what is this Executable UML? It is a profile of UML that allows you, the developer, to define the behavior of a single subject matter in sufficient detail that it can be executed. In this sense, the model is like code, but there's no point in writing "code" in UML just to rewrite it in Java or C++, so it's rather more revealing to examine what executable UML *doesn't* say that code might.

An executable UML model doesn't make coding decisions. It makes no statement about tasking structures; it makes no statement about distribution; it makes no statement about classes or encapsulation. An executable UML model describes only the data and behavior, organized into classes to be sure, about the subject matter at hand. In other words, an executable

UML developer describes subject matters at a *higher level of abstraction* than she would in a programming language.

To build a system, we build an executable UML of each subject matter. Typically, the system includes subject matters such as the application, a user interface, and some general services. The executable UML models for each of these subject matters are then woven together by an executable UML *model compiler.*

The model compiler targets a specific implementation embodying decisions about "coding:" tasking structures, distribution, data structures (which may be quite different from that suggested by the class structure), as well as the language. Model compilers can be extremely sophisticated, taking care of cross-cutting concerns such as transaction safety and rollback, or they can be sophisticated in a different way, targeting small footprint embedded systems with no tasking or other system support.

In general, a model compiler compiles several executable UML models, each of which captures a single cross-cutting concern to yield the running system. In this sense, executable UML makes use of the concepts in *aspect-oriented* programming.

Executable UML models support a new Object Management Group initiative, *Model-Driven Architecture* (MDA). This initiative is in its early stages, but its goal is to allow developers to compose complete systems out of models and other components. This goal requires at least an interface as contract and, behind the interface, the ability to express a solution without making coding decisions. That would be executable UML, or some variation.

This book does not describe model-driven architecture or its implications. Rather, this book focuses on one aspect of MDA that we believe to be critical: the ability to model whole subject matters completely and turn these models into systems. This ability, we believe, relies on being able to execute models. Hence executable UML.

Because the developer builds models as executable as a program for each subject matter, all the principles of *extreme* programming and *agile* processes can be applied. Indeed, many of the principles of these processes having nothing to do with coding *per se.*

You can use executable UML in a deliberate process or, because the models are executable, an agile one. Our preference is agile and incremental because it keeps the focus on delivering working software.

And what about RUP? As one of our reviewers, Martin Fowler, so memorably said: "My biggest concern with RUP is that it's so loose that any process seems to qualify as an instance of RUP. As a result, saying you're using RUP is a semantics-free statement." So we can reasonably assert that the process described by this book is an instance of RUP. (And if you want, we do.)

Frequently Asked Questions

Is this the only possible Executable UML? No. This rendition views each object as potentially having a state machine that can execute asynchronously and concurrently. We view this approach as necessary for today's distributed computing environments. However, one could define an executable UML that relies on synchronous method calls between objects to produce a completely synchronous model of the subject matter. Similarly, our particular use of the statechart diagram is not the only possible one.

Is Executable UML a standard? Yes and No. The notational elements you see in this book conform to UML, and so qualify as a *profile* of that standard. In addition, the execution semantics defined here conform to UML, though we do both subset UML and impose certain rules to link the elements together. What is not yet a standard is the exact content of what can and should be interchanged so that we can guarantee that any and all model compilers, irrespective of vendor, can compile any arbitrary executable UML model.

Throughout this book, we use standards as much as they are established. In some areas, the book is intended to provide a basis for discussion of what should ultimately become a standard.

Will there be a standard one day, and how might it differ? Yes, we hope so. Work has begun informally to define a standard and we will encourage and support it. We expect the standard to define an underlying semantics quite similar to that outlined here, and to layer increasingly rich syntax on top.

Does that mean I should wait? Not at all. This technology is taking off, and the basic elements are already established. Get ahead of the learning curve.

I know hardly anything about UML. Is this book too advanced for me? We assume you have an intuitive understanding of the goals behind UML, but nothing more. We will show you all the elements you need to build an executable UML model.

I'm a long-time UML user. Do I need this book? If you want to garner the benefits of Executable UML, then you'll have to learn the elements that make it up. Focus on the definitions we use and the chapters that show how to build and execute models. Skip the notational stuff. Be prepared to unlearn some UML and habits of mind required to model software structure, but not required to specify an executable model.

What happened to adornments such as aggregation or composition? We don't need them for Executable UML. UML enables you to model software structure, but that's not our purpose here, so those adornments, and many others, are not in our profile.

Some of this seems familiar. Is this just Shlaer-Mellor in UML clothing? Shlaer-Mellor focused on execution and specification of an abstract solution, not on specifying software structure. UML can be used for both the expression of software structure and the abstract model. Executable UML brings Shlaer-Mellor and UML together by using UML notation and incorporating concepts of execution. We hope this will make execution accessible to a broader community.

I've used Shlaer-Mellor before. Is this any different? A lot can happen in this industry in ten weeks, let alone the ten years since the publication of *Object Lifecycles*. First of all, of course, we all now use UML notation and vocabulary. (Resistance was futile.) Executable UML takes a more object-oriented perspective, no longer requiring identifiers or referential attributes, or other traces of Shlaer-Mellor's relational roots.

The addition of an action semantics to the UML is a major step forward. We hope the action semantics, and the very concept of an executable and translatable UML may one day be seen as a significant contribution of the Shlaer-Mellor community.

Progress in tools has also made certain conventions, such as event numbering, less critical to model understanding, though they are still helpful in keeping our minds clear.

Why do you say "Action Semantics?" Because UML defines only the semantics of actions, it does not define a language.

But how can you execute without an action language? We use an action semantics–conforming language that is executable today. We show several other action languages to illustrate that syntax is unimportant.

You use an Online Bookstore case study. Can I use this if I'm a real-time developer? Yes. We chose a more IT-oriented case study to increase the reach of the approach. You can find a completely worked out real-time case study in Leon Starr's book *Executable UML: The Elevator Case Study.*

How can I get an Executable UML tool? All of the examples in this book were developed using Project Technology's tool, BridgePoint. A copy of BridgePoint can be downloaded from the book's website, *www.executableUMLbook.com.*

How is this different from the old "draw the pictures, get some code" CASE tools? There are two main differences. First, compiling models produces the whole system, not just interfaces or frameworks. Second, there are many different model compilers available to buy, and even more that can be built, to meet exacting software architecture needs.

Where has Executable UML been used? Executable UML has been used to generate systems as large as two million lines of C++, and as small as handheld drug delivery devices. Executable UML has also been used in lease-origination, web-enabled executive reporting, and intermodal transportation logistics systems.

Why did you write this book? Because we had nothing better to do? No: There are lots of books out there that tell you about UML notation, but few of them focus on the subset you need for executability. Many books use UML to describe software structure. We explicitly spurn this usage.

Why should I buy this book? Because it describes completely everything you need to know about executable UML: It's the Executable UML handbook.

Stephen J. Mellor
San Francisco, California

Marc J. Balcer
San Francisco, California

March 2002

Acknowledgments

First and foremost, we must acknowledge our debt to the late Sally Shlaer. She started this ball rolling in the mid-1970s with a project that generated FORTRAN from a set of primitive data and program files, with daily builds and—perhaps astonishingly—many of the trappings of today's agile processes. The system, a radiation treatment facility, had only five bugs in its first full system test. None lasted over forty-eight hours. And a good job too, given the subject matter. We deeply miss her warmth, her unparalleled concern for people, and especially her steel-trap mind.

In our work together in the late eighties, we focused on objects as an organizing principle for describing data and behavior, culminating in the two Shlaer-Mellor books. Execution was then, and is now, the goal. Since then, of course, the UML has become the *lingua franca* of object modeling, but the UML, until the recent past, has not been executable. This book is intended to link together the executable ideas of Shlaer-Mellor with the universality of UML.

Someone, someday, should write a paper about the four stages of review and how they correspond to the four stage of grief. First there's disbelief, then denial, then bargaining, and finally acceptance. The paper should discuss the correspondence between "denial" and the abuse heaped upon

the reviewers as it becomes all too clear that the work needs to be revised. Fortunately, we didn't send too many of the hate mails we composed.

The table below lists our unfortunate reviewers. Those reviewers marked with a * were a part of the formal review team. We thank them all for their sterling efforts and apologize profusely to their burning ears.

* Conrad Bock	Mark Blackburn	Alistair Blair
* Dirk Epperson	Scott Finnie	* Martin Fowler
* Takao Futagami	Kazuto Horimatsu	Yukitoshi Okamura
* Edwin Seidewitz	* Leon Starr	* William G. Tanner

We would especially like to thank Conrad Bock, Director of Standards at Kabira Technologies, one of the few people worldwide who has the whole UML in his head, who provided a most detailed—and so, if he'll forgive us, an especially annoying—review.

We would like to thank the many talented analysts and developers at ThoughtWorks (http://www.thoughtworks.com), including Chief Scientist Martin Fowler, for keeping our focus on executability and its impact on agile development.

Our thanks, too, to William G. Tanner, Software Development Manager at Project Technology, Inc., who is apparently able to review executable models with a model compiler in his head.

This book would not be possible without the professionalism of the staff at Addison Wesley. Susan Winer, our copy editor, moved better than 50% of our commas and performed countless acts of linguistic hygiene. Kate Saliba, who leads the marketing team, has kept the project on track as it moved into production. John Fuller has been tolerant as we have attempted to do his job as production editor. Finally, our thanks to Paul "Eyebrows" Becker, alternately patient and a pain, who has cajoled us into finishing this project. Authors are not (always) easy to get along with!

Finally, we'd like to thank members of the community who have long understood the benefits of execution and modeling at a high level of abstraction. You know who you are, and we wouldn't still be here if it weren't for you. Thanks.

Contents