UML 2.0 技术手册（影印版）

UML 2.0 IN A NUTSHELL

A Desktop Quick Reference

UML 2.0 技术手册（影印版）

UML 2.0 in a Nutshell

Dan Pilone

with Neil Pitman

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

UML 2.0 技术手册 /（美）皮隆（Pilone, D.），（美）皮特曼（Pitman, N.）著. – 影印本. – 南京：东南大学出版社，2006.4

书名原文：UML 2.0 in a Nutshell

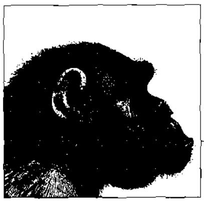
ISBN 7-5641-0273-X

Ⅰ.U… Ⅱ.①皮… ②皮… Ⅲ.面向对象语言，UML – 程序设计 – 技术手册 – 英文 Ⅳ.TP312-62

中国版本图书馆 CIP 数据核字（2006）第 010622 号

江苏省版权局著作权合同登记
图字：10-2006-43 号

书　　名/	UML2.0 技术手册（影印版）
书　　号/	ISBN 7-5641-0273-X
责任编辑/	张烨
封面设计/	Ellie Volckhausen，马冬燕
出版发行/	东南大学出版社（press.seu.edu.cn）
地　　址/	南京四牌楼 2 号（邮政编码 210096）
印　　刷/	扬中市印刷有限公司
开　　本/	787 毫米 × 980 毫米　16 开本　14.75 印张
版　　次/	2006 年 4 月第 1 版　2006 年 4 月第 1 次印刷
印　　数/	0001 – 2000 册
定　　价/	38.00 元（册）



Preface

About This Book

Welcome to *UML 2.0 in a Nutshell*. The Unified Modeling Language (UML) has expanded quite a bit since its inception and can be applied to many different domains, however it still has its roots in software development. We have tried to make this book applicable to as broad an audience as possible, but it's helpful to have at least a cursory knowledge of Object Oriented Programming (OOP) because UML draws much of its terminology from that domain.

Before going any further we'd like to clarify how this book refers to the Unified Modeling Language. Grammatically speaking, "the UML" is correct. However, it sounds weird. This book uses the more colloquial "UML".

UML 2.0 in a Nutshell is a detailed reference for the UML 2.0 Superstructure, from a user's perspective. Whenever it would be helpful to clarify a UML concept with a concrete example, we will present Java code.

In general we assume that you are familiar with OOP and the type of constructs that go with it (classes, methods, inheritance, etc.). However, we make no assumptions about what you know about UML. Each chapter starts with a top-to-bottom discussion of the chapter's topic. This will be fast paced and thorough, meant for those who understand the basics and want to know the "nitty-gritty" of a piece of UML. Subsequent sections are kinder, gentler discussions of the topic. This includes examples that show how the topic may be applied to typical problems, help you further refine your models to eliminate ambiguity, capture details that might otherwise be lost, or add information to your model that aids in tool-based development.

A brief word of warning: UML has a strict terminology for just about every aspect of modeling. This is necessary to reduce ambiguity and confusion as much as possible. However, in everyday use some terms are used interchangeably with others that have completely different meanings in UML. A classic example of this is `operation` and `method`. These are frequently treated as being synonymous in a software development environment but have different meanings when used in the context of UML. We will make a point to use the correct UML term even if it may not be the most colloquial name.

How to Use This Book

This book is divided based on UML diagram type. Obviously there is some crossover, as some diagrams build on concepts from others. Chapter 1, *Fundamentals of UML*, covers the basics of UML and presents some background information that will help you understand the context for the rest of the book. If you are familiar with previous versions of UML, you can probably skim this chapter. If you don't have a strong background in UML, you should definitely start here.

The next set of chapters cover what is called static modeling in UML. Static modeling captures the physical structure of a piece of software (as much as software has a "physical" structure). For example: what operations and attributes a class contains, what interfaces a class realizes, or what packages contain all this mess. The static modeling chapters include:

Chapter 2, *Class Diagrams*
: This chapter introduces the class diagram. It discusses the various elements that can be used on a class diagram, what they represent, and how to extend them. Because class diagrams are often a centerpiece of a UML model, you should know this chapter inside and out. The last part of the chapter discusses how class diagrams fit into the overall UML model and how the diagrams are typically mapped to code.

Chapter 3, *Package Diagrams*
: This chapter introduces packages and grouping within a UML model.

Chapter 4, *Composite Structures*
: This chapter introduces the new UML 2.0 concept of composite structures. Composite structures are specifically designed to represent patterns and are a major new component to the modeling language.

Chapter 5, *Component Diagrams*
: This chapter introduces components and the component diagram. Topics such as the stereotypes used in component diagrams, relationships between components, and component metainformation are discussed. The latter part of this chapter discusses how components are typically realized in a programming language.

Chapter 6, *Deployment Diagrams*
: This chapter introduces the concept of capturing system deployment using deployment diagrams. Deployment fundamentals such as nodes, node stereotypes, and relationships to components are explained. This chapter also includes a discussion on modeling a distributed system using deployment diagrams.

The next set of chapters cover the second half of UML—behavioral modeling. Behavioral modeling captures how the various elements of a system interact during execution. Diagrams such as the use case diagram can capture requirements from an external actor's perspective, and sequence diagrams can show how objects interact to implement a particular use case. The behavioral modeling chapters include:

Chapter 7, *Use Case Diagrams*
: This chapter introduces use cases, actors, and system boundaries. It goes slightly beyond pure UML in that the chapter touches on common practices regarding use cases, such as use case scoping, use case documents, and use case realizations.

Chapter 8, *Statechart Diagrams*
: This chapter introduces state machine modeling using states, actions, and transitions. Statecharts can be used to model a simple algorithm all the way up to a complex system.

Chapter 9, *Activity Diagrams*
: This chapter introduces a close relative to the statechart diagram, the activity diagram. Activity diagrams resemble old-school flowcharts and are typically used to model an algorithm or use case realization.

Chapter 10, *Interaction Diagrams*
: This chapter introduces the large set of interaction diagrams supported by UML 2.0. The two best-known diagrams are sequence and collaboration diagrams. This chapter also discusses the new timing-centric interaction diagram.

The final part of the book covers extension and applications of UML 2.0:

Chapter 11, *Tagged Values, Stereotypes, and UML Profiles*
: This chapter discusses how UML 2.0 may be extended and refined.

Chapter 12, *Effective Diagramming*
: This chapter departs from the specification side of UML 2.0 and offers real-world advice on modeling, what parts of UML 2.0 to use when, and how to effectively convey the right information.

Appendix A, *MDA: Model-Driven Architecture*
: This appendix introduces the Model-Driven Architecture (MDA). While MDA isn't a new idea, UML 2.0 has MDA in mind in several places, and next-generation tools may be able to make MDA a reality.

Appendix B, *The Object Constraint Language*
: This appendix describes the Object Constraint Language (OCL), a simple language defined to express constraints on UML diagrams. It can be applied in countless ways and is introduced here in its basic form.

If you're familiar with the fundamental UML concepts, you can read this book's chapters in nearly any order. However, there is always a certain amount of overlap between chapters because some elements can appear on many diagrams. Instead of repeating the information in each chapter, we fully define elements (and their associated stereotypes, attributes, etc.) the first time they are encountered, and in subsequent chapters, we provide detailed cross references back to the original definition, when needed.

Typographic Conventions

The following typographic conventions are used in this book:

`Constant width`
: Used in the text to refer to class names, stereotypes, and other elements taken from UML diagrams.

`Constant width italic`
: Used in UML diagrams to indicate text that would be replaced by the user.

Italic
: Used when new terms are introduced, and for URLs and file references.

...
: Ellipses indicate nonessential material that has been omitted from a diagram for the sake of readability.

Indicates a tip, suggestion, or general note.

Indicates an aspect of UML that you must be particularly careful about using.

Note that UML makes frequent use of curly braces ({}) and guillemots («»). When these are used in a syntax definition, they are required by UML.

Nearly everything in UML notation is optional, so there is no specific notation to indicate an optional field. If a particular piece of syntax is required, it is noted in the text.

Safari Enabled

When you see a Safari® Enabled icon on the cover of your favorite technology book, it means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at *http://safari.oreilly.com*.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)

707-829-0515 (international/local)
707-829-0104 (fax)

There is a web page for this book that lists errata, examples, or any additional information. You can access this page at:

http://www.oreilly.com/catalog/umlnut2

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:

http://www.oreilly.com

Acknowledgments

From Dan

This book was truly a team effort. Without support, emails, comments, harassment, and suggestions from friends, family, and colleagues, this book would not have happened. First I'd like to thank my editor, Jonathan Gennick, for his astonishing amount of patience. He is fantastic to work with and helped keep this book on track.

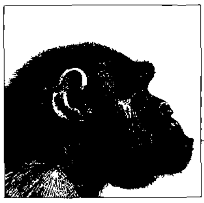
Next, I'd like to thank the technical reviewers who were never short on suggestions or comments. At times I felt like this was the fourth edition of the book, after working in all their ideas. The tech reviewers were: Stephen Mellor, Michael Chonoles, Mike Hudson, Bernie Thuman, Kimberly Hamilton, Russ Miles, and Julie Webster.

Finally, I'd like to thank my family: my parents for supporting me from the start and setting an example that has driven me in both my professional and personal life, and my wife, Tracey, for somehow managing to hold everything together while I wrote this book. Compared to the magic she has been working, writing this book was a piece of cake. Last but not least, I'd like to thank my son Vinny: now we can head to the park!

From Neil

I'd like to thank Ron Wheeler and Jacques Hamel of Artifact-Software for allowing the use of XML examples. Thanks also to Derek McKee of Mindset Corporation for the use of LamMDA examples. Finally, I'd like to especially thank Jonathan Gennick for his depth of patience.

Table of Contents



1 Fundamentals of UML

On the surface, the Unified Modeling Language (UML) is a visual language for capturing software designs and patterns. Dig a little deeper, though, and you'll find that UML can be applied to quite a few different areas and can capture and communicate everything from company organization to business processes to distributed enterprise software. It is intended to be a common way of capturing and expressing relationships, behaviors, and high-level ideas in a notation that's easy to learn and efficient to write. UML is visual; just about everything in it has a graphical representation. Throughout this book we'll discuss the meaning behind the various UML elements as well as their representations.

Getting Started

If you're new to UML, you should be sure to read this chapter all the way through to get acquainted with the basic terminology used throughout the book. If you are a developer, class diagrams tend to be the simplest diagrams to start with because they map closely to code. Pick a program or domain you know well, and try to capture the entities involved using classes. Once you're convinced you've modeled the relationships between your entities correctly, pick a piece of functionality and try to model that using a sequence diagram and your classes.

If you're more of a process person (business or otherwise), you may be more comfortable starting with an activity diagram. Chapter 9 shows examples of modeling business processes with different groups (Human Resources, IT, etc.) and progresses to modeling parallel processes over different geographic regions.

Background

UML has become the de facto standard for modeling software applications and is growing in popularity in modeling other domains. Its roots go back to three distinct methods: the Booch Method by Grady Booch, the Object Modeling

Technique coauthored by James Rumbaugh, and Objectory by Ivar Jacobson. Known as the Three Amigos, Booch, Rumbaugh, and Jacobson kicked off what became the first version of UML, in 1994. In 1997, UML was accepted by the Object Management Group (OMG) and released as UML v1.1.

Since then, UML has gone through several revisions and refinements leading up to the current 2.0 release. Each revision has tried to address problems and shortcomings identified in the previous versions, leading to an interesting expansion and contraction of the language. UML 2.0 is by far the largest UML specification in terms of page count (the superstructure alone is over 600 pages), but it represents the cleanest, most compact version of UML yet.

UML Basics

First and foremost, it is important to understand that UML is a *language*. This means it has both syntax and semantics. When you model a concept in UML, there are rules regarding how the elements can be put together and what it means when they are organized in a certain way. UML is intended not only to be a pictorial representation of a concept, but also to tell you something about its context. How does widget 1 relate to widget 2? When a customer orders something from you, how should the transaction be handled? How does the system support fault tolerance and security?

You can apply UML in any number of ways, but common uses include:

- Designing software
- Communicating software or business processes
- Capturing details about a system for requirements or analysis
- Documenting an existing system, process, or organization

UML has been applied to countless domains, including:

- Banking and investment sectors
- Health care
- Defense
- Distributed computing
- Embedded systems
- Retail sales and supply

The basic building block of UML is a diagram. There are several types, some with very specific purposes (timing diagrams) and some with more generic uses (class diagrams). The following sections touch on some of the major ways UML has been employed. The diagrams mentioned in each section are by no means confined to that section. If a particular diagram helps you convey your message you should use it; this is one of the basic tenets of UML modeling.

Designing Software

Because UML grew out of the software development domain, it's not surprising that's where it still finds its greatest use. When applied to software, UML attempts to bridge the gap between the original idea for a piece of software and its implementation. UML provides a way to capture and discuss requirements at the requirements level (use case diagrams), sometimes a novel concept for developers. There are diagrams to capture what parts of the software realize certain requirements (collaboration diagrams). There are diagrams to capture exactly *how* those parts of the system realize their requirements (sequence and statechart diagrams). Finally there are diagrams to show how everything fits together and executes (component and deployment diagrams).

Books describing previous versions of UML made a point to emphasize that UML was not a visual programming language; you couldn't execute your model. However, UML 2.0 changes the rules somewhat. One of the major motivations for the move from UML 1.5 to UML 2.0 was to add the ability for modelers to capture more system behavior and increase tool automation. A relatively new technique called Model Driven Architecture (MDA) offers the potential to develop executable models that tools can link together and to raise the level of abstraction above traditional programming languages. UML 2.0 is central to the MDA effort.

It is important to realize the UML is *not* a software process. It is meant to be used within a software process and has facets clearly intended to be part of an iterative development approach.

While UML was designed to accommodate automated design tools, it wasn't intended *only* for tools. Professional whiteboarders were kept in mind when UML was designed, so the language lends itself to quick sketches and capturing "back of the napkin" type designs.

Business Process Modeling

UML has an extensive vocabulary for capturing behavior and process flow. Activity diagrams and statecharts can be used to capture business processes involving individuals, internal groups, or even entire organizations. UML 2.0 has notation that helps model geographic boundaries (activity partitions), worker responsibilities (swim lanes), and complex transactions (statechart diagrams).

UML Specifications

Physically, UML is a set of specifications from the OMG. UML 2.0 is distributed as four specifications: the Diagram Interchange Specification, the UML Infrastructure, the UML Superstructure, and the Object Constraint Language (OCL). All of these specifications are available from the OMG web site, *http://www.omg.org*.

The Diagram Interchange Specification was written to provide a way to share UML models between different modeling tools. Previous versions of UML defined an XML schema for capturing what elements were used in a UML diagram, but did not capture any information about how a diagram was laid out. To address this, the Diagram Interchange Specification was developed along with a mapping from a new XML schema to a Scalable Vector Graphics (SVG) representation. Typically the Diagram Interchange Specification is used only by tool vendors, though the OMG makes an effort to include "whiteboard tools."

The UML Infrastructure defines the fundamental, low-level, core, bottom-most concepts in UML; the infrastructure is a metamodel that is used to produce the rest of UML. The infrastructure isn't typically used by an end user, but it provides the foundation for the UML Superstructure.

The UML Superstructure is the formal definition of the elements of UML, and it weighs in at over 600 pages. This is the authority on all that is UML, at least as far as the OMG is concerned. The superstructure documentation is typically used by tool vendors and those writing books on UML, though some effort has been made to make it human readable.

The OCL specification defines a simple language for writing constraints and expressions for elements in a model. The OCL is often brought into play when you specify UML for a particular domain and need to restrict the allowable values for a parameter or object. Appendix B is an overview of the OCL.

It is important to realize that while the specification is the definitive source of the *formal definition* of UML, it is by no means the be-all and end-all of UML. UML is designed to be extended and interpreted depending on the domain, user, and specific application. There is enough wiggle room in the specification to fit a data center through it... this is intentional. For example, there are typically two or more ways to represent a UML concept depending on what looks best in your diagram or what part of a concept you wish to emphasize. You may choose to represent a particular element using an in-house notation; this is perfectly acceptable as far as UML is concerned. However, you must be careful when using nonstandard notation because part of the reason for using UML in the first place is to have a common representation when collaborating with other users.

Putting UML to Work

A UML model provides a view of a system—often just one of many views needed to actually build or document the complete system. Users new to UML can fall into the trap of trying to model everything about their system with a single diagram and end up missing critical information. Or, at the other extreme, they may try to incorporate every possible UML diagram into their model, thereby overcomplicating things and creating a maintenance nightmare.

Becoming proficient with UML means understanding what each diagram has to offer and knowing when to apply it. There will be many times when a concept could be expressed using any number of diagrams; pick the one(s) that will mean the most to your users.

Each chapter of this book describes a type of diagram and gives examples of its use. There are times when you may need to have more than one diagram to capture all the relevant details for a single part of your system. For example, you may need a statechart diagram to show how an embedded controller processes input from a user as well as a timing diagram to show how the controller interacts with the rest of the system as a result of that input.

You should also consider your audience when creating models. A test engineer may not care about the low-level implementation (sequence diagram) of a component, only the external interfaces it offers (component diagram). Be sure to consider who will be using each diagram you produce and make it meaningful to that person.

UML Profiles

In addition to a variety of diagram types, UML is designed to be extended. You can informally extend UML by adding constraints, stereotypes, tagged values, and notes to your models, or you can use the formal UML extension and define a full *UML profile*. A UML profile is a collection of stereotypes and constraints on elements that map the otherwise generic UML to a specific problem domain or implementation. For example, there are profiles for CORBA, Enterprise Application Integration (EAI), fault tolerance, database modeling, and testing. See Chapter 11 for more information on UML 2.0 Profiles.

Modeling

It should go without saying that the focus of UML is modeling. However, what that means, exactly, can be an open-ended question. Modeling is a means to capture ideas, relationships, decisions, and requirements in a well-defined notation that can be applied to many different domains. Modeling not only means different things to different people, but also it can use different pieces of UML depending on what you are trying to convey.

In general a UML model is made up of one or more *diagrams*. A diagram graphically represents things, and the relationships between these things. These things can be representations of real-world objects, pure software constructs, or a description of the behavior of some other object. It is common for an individual thing to show up on multiple diagrams; each diagram represents a particular interest, or *view*, of the thing being modeled.

Diagrams

UML 2.0 divides diagrams into two categories: *structural diagrams* and *behavioral diagrams*. Structural diagrams are used to capture the physical organization of the things in your system—i.e., how one object relates to another. There are several structural diagrams in UML 2.0:

Class diagrams
: Class diagrams use classes and interfaces to capture details about the entities that make up your system and the static relationships between them. Class diagrams are one of the most commonly used UML diagrams, and they vary in detail from fully fleshed-out and able to generate source code to quick sketches on whiteboards and napkins. Class diagrams are discussed in Chapter 2.

Component diagrams
: Component diagrams show the organization and dependencies involved in the implementation of a system. They can group smaller elements, such as classes, into larger, deployable pieces. How much detail you use in component diagrams varies depending on what you are trying to show. Some people

simply show the final, deployable version of a system, and others show what functionality is provided by a particular component and how it realizes its functionality internally. Component diagrams are discussed in Chapter 5.

Composite structure diagrams
Composite structure diagrams are new to UML 2.0. As systems become more complex, the relationships between elements grow in complexity as well. Conceptually, composite structure diagrams link class diagrams and component diagrams; they don't emphasize the design detail that class diagrams do or the implementation detail that composite structures do. Instead, composite structures show how elements in the system combine to realize complex patterns. Composite structures are discussed in Chapter 4.

Deployment diagrams
Deployment diagrams show how your system is actually executed and assigned to various pieces of hardware. You typically use deployment diagrams to show how components are configured at runtime. Deployment diagrams are discussed in Chapter 6.

Package diagrams
Package diagrams are really special types of class diagrams. They use the same notation but their focus is on how classes and interfaces are grouped together. Package diagrams are discussed in Chapter 3.

Object diagrams
Object diagrams use the same syntax as class diagrams and show how actual instances of classes are related at a specific instance of time. You use object diagrams to show snapshots of the relationships in your system at runtime. Object diagrams are discussed as part of class diagrams in Chapter 2.

Behavioral diagrams focus on the behavior of elements in a system. For example, you can use behavioral diagrams to capture requirements, operations, and internal state changes for elements. The behavioral diagrams are:

Activity diagrams
Activity diagrams capture the flow from one behavior or *activity*, to the next. They are similar in concept to a classic flowchart, but are much more expressive. Activity diagrams are discussed in Chapter 9.

Communication diagrams
Communication diagrams are a type of interaction diagram that focuses on the elements involved in a particular behavior and what messages they pass back and forth. Communication diagrams emphasize the objects involved more than the order and nature of the messages exchanged. Communication diagrams are discussed as part of interaction diagrams in Chapter 10.

Interaction overview diagrams
Interaction overview diagrams are simplified versions of activity diagrams. Instead of emphasizing the activity at each step, interaction overview diagrams emphasize which element or elements are involved in performing that activity. The UML specification describes interaction diagrams as emphasizing who has the *focus of control* throughout the execution of a