

Professional UML with Visual Studio® .NET

Unmasking Visio® for Enterprise Architects

Andrew Filev
Tony Loton
Kevin McNeish
Ben Schoellmann
John Slater
Chaur G. Wu



Professional UML with Visual Studio® .NET

Unmasking Visio® for Enterprise Architects

Andrew Filev
Tony Loton
Kevin McNeish
Ben Schoellmann
John Slater
Chaur G. Wu



**Professional UML with Visual Studio® .NET:
Unmasking Visio® for Enterprise Architects**

Published by

Wiley Publishing, Inc.

10475 Crosspoint Boulevard

Indianapolis, IN 46256

www.wiley.com

Copyright © 2003 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

Library of Congress Card Number: 2003107079

ISBN: 0-7645-4376-8

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/QU/QW/QT/IN

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8700. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4447, E-mail: permcoordinator@wiley.com.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: WHILE THE PUBLISHER AND AUTHOR HAVE USED THEIR BEST EFFORTS IN PREPARING THIS BOOK, THEY MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK AND SPECIFICALLY DISCLAIM ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES REPRESENTATIVES OR WRITTEN SALES MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR YOUR SITUATION. YOU SHOULD CONSULT WITH A PROFESSIONAL WHERE APPROPRIATE. NEITHER THE PUBLISHER NOR AUTHOR SHALL BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGES, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (800) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Trademarks: Wiley, the Wiley Publishing logo, Wrox, the Wrox logo, the Wrox Programmer to Programmer logo and related trade dress are trademarks or registered trademarks of Wiley in the United States and other countries, and may not be used without written permission. Visual Studio is a trademark of Microsoft Corporation. Visio is a trademark of Visio Corporation. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Trademark Acknowledgments

Wrox has endeavored to provide trademark information about all the companies and products mentioned in this book by the appropriate use of capitals. However, Wrox cannot guarantee the accuracy of this information.

Credits

Authors

Andrew Filev
Tony Loton
Kevin McNeish
Ben Schoellmann
John Slater
Chaur G. Wu

Technical Reviewers

Kourosh Ardestani
Paul Churchill
Mitch Denny
Mark Horner
Andrew Krowczyk
Christian Nagel
Ben Schoellmann
David Schultz
Bill Sempf
Erick Sqarbi
Helmut Watson

Managing Editor

Louay Fatoohi

Commissioning Editors

Douglas Paterson
Gerard Maguire

Technical Editors

Gerard Maguire
Douglas Paterson

Project Manager

Charlotte Smith

Production Coordinator

Sarah Hall

Cover

Natalie O'Donnell

Indexer

Andrew Criddle

Proofreader

Chris Smith

About the Authors

Andrew Filev

Andrew Filev is President of dotSITE Software. This company specializes in cost-effective development on the .NET platform. Andrew's team has been developing commercial solutions using .NET since the first public announcement of this new Microsoft strategy. Andrew set up one of the first .NET portals, and has held a number of seminars and lectures dedicated to .NET in state and private companies.

Andrew has implemented numerous solutions in various high-tech fields – Web Services, ERP applications, medical systems, development frameworks, among others. He can be reached at andrew@dotsitesoftware.com or www.dotsitesoftware.com.

Special thanks for my friend Igor for his help with the book, also for Doug, Charlotte, and Gerard for their great editorial work. Thanks for dotSITE team without whom I would never take a part in working on the architecture of such interesting projects.

Tony Loton

Tony Loton works through his company LOTONtech Limited (<http://www.lotontech.com>) as an independent consultant, course instructor, and technical writer. The current area of interest at LOTONtech is the enhancement of UML visual modeling tools – specifically Rational Rose and Visio for Enterprise Architects – to facilitate .NET application design. Further details can be found at <http://www.lotontech.com/visualmodeling>.

Tony graduated in 1991 with a BSc. Hons. degree in Computer Science and Management and he currently holds an appointment as associate lecturer with the Open University in the UK.

My contribution is dedicated to my wife and children, for respecting my need to “get on with it”.

Kevin McNeish

Kevin is President of Oak Leaf Enterprises, a company that specializes in object-oriented developer tools, training, and software. He started his programming career twenty years ago working with Assembly Language, then moved to C, Visual FoxPro, and currently uses C# as his primary .NET development tool. He authored the book *.NET for Visual FoxPro Developers* and teaches both .NET and UML training classes in North America and Europe.

He has also written UML articles for *CoDe*, *FoxPro Advisor*, and *FoxTalk* magazines. Kevin, a Microsoft MVP, is the creator of a .NET business application framework called "The Mere Mortals Framework for .NET". He also mentors software companies in a variety of vertical markets to design and build component-based applications that scale from the desktop to the Internet. He can be reached at oakleaf@oakleafsd.com or www.oakleafsd.com.

As always, thanks to my wife Nicole and my sons Jordan, Timothy, and Alexander for their love and support while writing this book!

Ben Schoellmann

Benjamin Schoellmann credits his move to sunny Houston, Texas, with providing the inspiration necessary to pursue a development and writing career. Currently he is involved with evangelizing .NET technologies among his coworkers at Synhrgy HR Technologies. Among his favored activities are golfing, tinkering with his network, talking incessantly, and integrating hardware and software solutions, primarily home automation, to enhance his leisurely pursuit of Slack. He maintains several content-free WEB domains, including Benjammin.com. He is obsessive about keeping pace with emerging technologies, and is very quick to credit his developer friends with all his success in the IT field.

And while he's quick to blame the dog for just about everything, his friends know better.

I'd like to thank all my friends and family for putting up with me during this process... and the rest of the time as well.

John Slater

John Slater is a project manager at Management Reports International in Cleveland, OH. At MRI he is currently developing applications for the property management industry. Right now, he is working on several projects using .NET development tools and .NET Enterprise servers.

In his free time John enjoys outdoor activities and playing with his children Rachel and Nathan. He can be reached at jr_slater@hotmail.com.

Chaur G. Wu

Chaur Wu currently works for Trend Micro Inc. as a senior software engineer. He started software programming before was old enough to qualify for a driving license. The first program he wrote was a bingo game – in assembly code on a 8051 single chip. To capitalize on the program, he ran a small casino in the lab – he developed primitive game boxes that connected his pals and allowed them to place bets.

He's also been involved in much larger projects. For example, he developed a program in C++ to simulate the movement and geographical coverage of GPS satellites. As a research assistant in his graduate study, he implemented a wavelet-based video compression algorithm for a traffic surveillance system sponsored by Boston City Department of Transportation. He also helped solve a blurred image problem using inverse filters and other image processing algorithms for a client who designs fiber optics components in San Jose, CA.

His technical interests include distributed software systems in Java, COM, and .NET, generative programming, software design, and neural networks. Outside of work, his favorite vacation combines a one-night gambling trip to Reno followed by a day of skiing at some resort near Lake Tahoe.

You can e-mail Chaur at cha_urwu@hotmail.com.

I would like to dedicate my efforts in this book to my two-year-old daughter, Sarah.– CGW

Table of Contents

Introduction	1
What Does This Book Cover?	2
Who Is This Book For?	3
What You Need to Use This Book	3
Conventions	3
Customer Support	4
How to Download the Sample Code for the Book	4
Errata	4
E-Mail Support	5
p2p.wrox.com	5
Chapter 1: Review of UML	9
What is the Unified Modeling Language?	9
Why use UML?	10
A Brief History of UML	11
End-to-End UML Modeling	12
UML Essential Notation and Core Concepts	13
Activity Diagram	13
Use Case Diagram	15
Sequence and Collaboration Diagram	16
Statechart Diagram	18
Static Structure Diagram	20
Component Diagram	21
Deployment Diagram	23
Fitting the Pieces into the UML Jigsaw	24
UML Modeling Tools	26
Process Essentials	27
(Rational) Unified Process	27
RUP .NET Developer's Configuration	29
RUP .NET Plug-in	29
Microsoft Solutions Framework	30
A Framework not a Process	30
Summary	31
Modeling Summary	31
Process Summary	32

Chapter 2: A Tour of Visio **35**

Visio Background **35**

Beginning Visio – A Simple Diagram **36**

The Visio Environment 36

Shapes 39

Connectors 42

Common Visio Software Diagrams **47**

Creating COM and OLE Diagrams 49

Creating Data Flow Diagrams 52

Creating Enterprise Applications 55

Windows Interface Diagrams 56

Windows Form Shape 57

Creating Database Model Diagrams 66

Creating a Simple Data Model 66

Summary **71**

Chapter 3: Diagramming Business Objects **73**

What is a Business Object? **74**

Object Modeling Compared to Data Modeling 74

Modeling Attributes and Behavior 75

Building Monolithic Applications 75

Building Component-Based Applications 76

Benefits of using Business Objects **76**

Flexibility – Write Once, Reuse Everywhere 76

Data-Access Flexibility – Write Once, Change Once 77

Normalizing Application Logic – Write Once, Period! 77

Where's the Code? – Write It and Find It 77

Designing Complex Software 77

Designing a Component-based Application **78**

Business and Data-Access Base Classes **78**

Creating Namespace Packages 79

Creating an Abstract Data-Access Class 80

Creating a Class (Static Structure) Diagram 81

Adding Operations to the Class 82

The UML Class Properties Dialog 83

The UML Operation Properties Dialog 84

Specifying Operation Parameters 86

Marking an Operation as Abstract 89

Adding .NET Base Classes to the Model 90

Adding the SaveDataSet Operation 91

Creating Concrete Subclasses 93

Creating a Business Object Base Class 95

Choosing a .NET Type for Your Business Object Class 95

Creating the BusinessObject Class 96

Specifying a Composition Relationship	97
Adding Overloaded BusinessObject Operations	99
Adding the GetDataSet Operations	99
Adding the SaveDataSet Operations	100
Adding Attributes to the BusinessObject Class	101
Use Cases for a Simple Library System	103
Modeling the Check Out Media Use Case	105
Deriving Classes from Use Cases	105
Thinking about Data	106
Creating a Sequence Diagram	107
Changing the Drawing Page Orientation	108
Adding Use Case Text to the Sequence Diagram	108
Adding the Actor and UI Placeholder	110
Adding Messages Between Objects	114
Creating Business Object Classes	116
Adding the Borrower Object to the Sequence Diagram	118
Adding a Message Call to the Borrower Object	119
Resizing the Activation Shapes	121
Retrieving Checked-Out Media	122
Calculating Fines	124
Displaying the Borrower Information	125
Checking Out Media	127
Tweaking the Sequence Diagram	129
Summary	130
Chapter 4: Generating Code from the Visio Model	133
Overview of Code Generation	133
Code Generation in Visio	134
The Employee Class	135
Adding Fields	136
Previewing Code	136
Adding Properties	138
Adding a Method	141
Implementing an Interface	144
Generating Code	148
Regenerating Code	152
Checking Errors	152
Code Generation in Different Languages	153
Code Templates	154
Using Templates	155
Working with Templates	155
Templates for Specific Entities	158
Creating New Templates	160
XML Comments and Code Templates	162

Table of Contents

Enhancing the Model	167
Mapping Associations	167
Creating a Collection	169
Creating an Indexer	169
Creating Events and Delegates	170
Overriding Methods	171
Summary	171
<hr/> Chapter 5: Reverse Engineering	<hr/> 175
Why Reverse Engineer?	175
Reverse Engineering from Source Code	176
Reverse Engineering QuickStart	176
Key Features and Limitations of Reverse Engineering	178
Reverse Engineering Granularity	178
Semantic Errors	179
Static Structure Diagrams	180
Round-trip Engineering	181
Projects that Don't Compile	181
Source Code Required	181
Reverse Engineering Example	182
Reverse Engineering the ParcelTracker Application	182
Reverse-Engineered Model Structure	183
Code-to-UML Mapping Examples	185
Generalization (or Inheritance)	185
Associations and Attributes	187
Operations and Properties	189
Method Bodies	191
Primitive and Value Types	192
Reverse Engineering, No Source Code Required	193
Running the RE.NET Lite Reverse Engineer	194
Creating a .NET Framework Base Model	197
RE.NET Lite Internals	198
RE.NET Lite Limitations	203
Summary	203
<hr/> Chapter 6: Documenting the Project	<hr/> 207
The Typical Software Development Lifecycle	208
Requirements Development	209
Architecture or High-Level Design	210
Detailed Design	210
Coding/Implementation	210
Testing/Quality Assurance	210
Rollout	211
Support/Maintenance	211

Role of UML and Visio in the Project	211
UML as Documentation	211
Requirements Development Documentation	212
Use Cases	214
Using Use Cases for Documentation	216
Architecture Documentation	218
Using Class Diagrams	218
Using Activity Diagrams	219
Using Component Diagrams	221
Detailed Design Documentation	222
Using Detailed Class Diagrams	222
Using Sequence Diagrams	224
Coding and Implementation Documentation	225
Requirements Documentation for Coding	226
Architecture Documents for Coding	226
Testing and Quality Assurance Documentation	226
Using Visio Reports throughout the Project	227
Static Structure Diagram Report	232
Deployment Diagram Report	235
Component Report	236
Summary	237
<hr/> Chapter 7: Distributed System Design	<hr/> 239
Object-Based Distributed Systems in .NET	240
Distributed Systems and Local Systems	240
.NET Infrastructure for Distributed Systems	243
.NET Remoting	243
ASP.NET Versus .NET Remoting	246
Preparation Work in Visio	247
Custom UML Stereotypes for .NET Distributed Systems	248
Package and Deploy the Bank Application	252
System Requirements	253
.NET Remoting Type and Activation Mode	255
Identify the Elements that Should Go into One Component.	257
Adding a Detailed Component Diagram	257
Adding a Bird's-Eye-View Component Diagram	260
Component Packaging	261
Map Components to Physical Deployment Nodes	263
Summary	267

Chapter 8: Database Modeling with Visio for Enterprise Architects 269

Design Process Overview 270

- Database Modeling 270
- Object Role Modeling (ORM) 271
- What is ORM? 271
 - Elementary Facts 272
 - ORM Notation 272
 - The Conceptual Schema Design Procedure 273
- Visio Data Projects 276
 - Creating the Data Project 277
- Step 2 – Drawing the Fact Types 278
 - The Business Rules Window 278
 - The Fact Editor 279
 - Visio Reporting 283
- Step 3 of the CSDP 284
- Constraints 285
- Steps 4 to 7 of the CSDP 290
 - Traveler, TicketPrice, and FrequentFlierMiles 290
 - Add Constraint Form 291
 - The Add Constraint Editor 292
- Creating the Conceptual, Logical, and Physical Database 296
 - Creating the Object Types 296
 - Creating the Predicate Types 297
- Building the Logical Model 302
 - Generating the Database 308
 - Organization of the Data Projects 311
 - Refreshing the Model from the Database 312

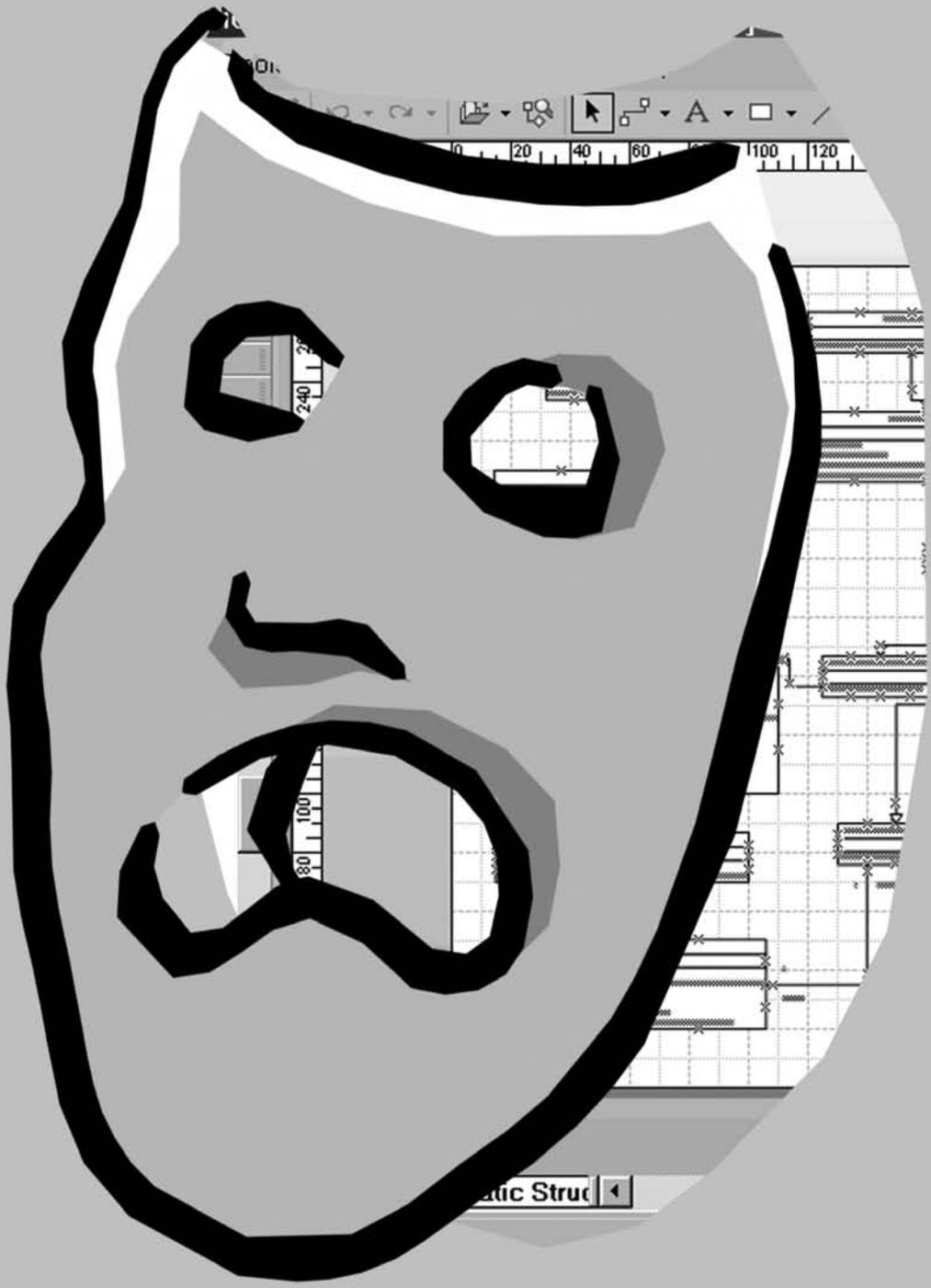
Reverse Engineering the Database 314

- Reverse Engineering an ER diagram 315
 - The Reverse Engineer Wizard 315
 - The Database Properties Window for ER diagrams 320
- Reverse Engineering an ORM Diagram 324
 - The Database Properties Window for ORM Diagrams 324

Summary 327

Index 329

- A Guide to the Index 329



Introduction

To many, Visio for Enterprise Architects appears to be a mysterious diagramming tool. In conjunction with Visual Studio .NET Enterprise Architect its potential seems clear – going from design to code, and back from code to design offers the developer tremendous benefits for rapidly developing applications. Why do we say 'mysterious'? This is because Visio's range of features can daunt the user, but most importantly, many aspects of its use directly relevant to software developers are frustratingly lacking in explanation.

This book aims to address this problem – here we focus exclusively on Visio's features for developing .NET applications, encompassing:

- ❑ UML diagrams
- ❑ Generating code from UML diagrams
- ❑ Reverse engineering source code into UML diagrams
- ❑ Database modeling

Along the way, we'll see some more general applications of Visio to the software development lifecycle, and also learn about Visio's idiosyncrasies, which almost every user of Visio will have encountered, and wondered "Is it just me?"

In other words, this book will allow you to finally unmask Visio for Enterprise Architects.

What Does This Book Cover?

Chapter 1 starts us off by reviewing the key UML concepts, the main diagram types, and the role of those diagrams within the software development process. If you're quite new to UML this will serve as a practical introduction that will help you make sense of the rest of the book.

In Chapter 2 we have our first dip into Visio, and have a look around the general Visio environment. Before we hit the main feature of the book, the UML diagrams, we look at other aspects of Visio that aid software development, and make an attempt to familiarize ourselves with Visio, its pages, shapes, and connectors.

In Chapter 3 we cover using Visio for object modeling – defining data access base classes for your .NET applications, defining a business object base class, deriving business classes from use cases, working with abstract and concrete classes, and using sequence diagrams to model the flow of messages between objects. Along the way we'll meet many of Visio's UML diagramming features, setting us up for the next chapter.

Visio for Enterprise Architects can generate skeleton source code from an existing UML diagram in C#, Visual Basic .NET, or C++. Moreover, Visio provides further options that give the developer greater control over the implementation of this source code. In Chapter 4 we look at how to generate code from a UML model in Visio, the various options available for generating code, including the use of code templates to specify the structure of the source code generated by Visio. We look at a variety of UML to code mappings, typical of the situations you will encounter in more complex models.

The Visual Studio .NET Enterprise Architect and Visio for Enterprise Architects combination provides a facility for reverse engineering existing C#, VB.NET, or C++.NET source code into a Visio UML static structure model. In Chapter 5 we'll look at this reverse engineering feature and cover why reverse engineering is useful how to reverse engineer .NET source code from within the Visual Studio .NET IDE, explore the structure of a typical reverse-engineered Visio UML model, and look at the code to UML mappings for important constructs such as generalization (inheritance) and association. We finish the chapter by using reflection to reverse engineer .NET assemblies to provide .NET Framework base class models for our UML diagrams.

In Chapter 6, we take a step back from the world of diagramming, generating code, and generating more diagrams from code, and look at the role of Visio and UML in the entire software development lifecycle. In effect, we'll be discussing how we document our work at different stages of a typical development project using Visio and UML – at the end of this chapter you'll take away some deeper insight into using Visio and UML in the course of working on your own projects.

Chapter 7 sees us move on to another area of using Visio to assist with general design issues. Designing a distributed system is an iterative process from **requirements analysis** to **modular breakdown** and to **packaging** and **deployment strategies**. However, designing a distributed system is different from designing a non-distributed one. In this chapter we look at a .NET Remoting example, a Bank application. We begin with an overview of .NET Remoting, and we see how to decide which classes in our application should be .NET Remoting types, how to decide the activation mode of each .NET remoting type, and how this can be diagrammed in Visio, what code elements should be grouped in a component, how to prepare a component diagram, and how to prepare a deployment diagram.

Chapter 8 moves us on to yet another aspect of Visio directly relevant to the enterprise developer – data modeling. We take a detailed walk through database modeling and Object Role Modeling (ORM), looking at Visio's ORM Source Diagrams and Entity Relationship Source Diagrams. We then see how to generate a database schema from these models, and further tweak the design with reverse engineering of the database into ORM and ER models, and updating the database with our modifications to yield round-trip database engineering.

Who Is This Book For?

This book is for the .NET developer who:

- ❑ Is comfortable with the basic concepts of UML
- ❑ Wants to learn how to use Visio for Enterprise Architects effectively
- ❑ Wants to see how UML and Visio can benefit their projects in general

What You Need to Use This Book

This book is based around the following combination:

- ❑ Visual Studio .NET Enterprise Architect Edition
- ❑ Visio for Enterprise Architects

Thus, having access to each is a prerequisite for using this book.

Conventions

We've used a number of different styles of text and layout in this book to help differentiate between different kinds of information. Here are examples of the styles we used and an explanation of what they mean.

Code has several fonts. If it's a word that we're talking about in the text – for example, when discussing a `for (. . .)` loop, it's in this font. If it's a block of code that can be typed as a program and run, then it's also in a gray box:

```
public Employee this[int index]
```

Sometimes we'll see code in a mixture of styles, like this:

```
public Employee this[int index]
{
    get
    {
        foreach (Employee em in employees)
        {
            if (em.ID == index)
```

```
        return em;
    }
    return null;
}
```

In cases like this, the code with a white background is code we are already familiar with; the line highlighted in gray is a new addition to the code since we last looked at it.

Advice, hints, and background information come in this type of font.

Important pieces of information come in boxes like this.

Bullets appear indented, with each new bullet marked as follows:

- ❑ **Important Words** are in a bold type font.
- ❑ Words that appear on the screen, or in menus like the **Open** or **Close**, are in a similar font to the one you would see on a Windows desktop.
- ❑ Keys that you press on the keyboard, like *Ctrl* and *Enter*, are in italics.

Customer Support

We always value hearing from our readers, and we want to know what you think about this book: what you liked, what you didn't like, and what you think we can do better next time. You can send us your comments, either by returning the reply card in the back of the book, or by e-mail to feedback@wrox.com. Please be sure to mention the book title in your message.

How to Download the Sample Code for the Book

When you visit the Wrox web site, www.wrox.com, locate the title through our Find a Book facility or by using one of the title lists. Click **Download Code** on the book's detail page, or on the **Download** item in the **Code** column for title lists.

The files that are available for download from our site have been archived using WinZip. When you've saved the archive to a folder on your hard drive, you need to extract the files using a decompression program such as WinZip or PKUnzip. When you extract the files, the code will be extracted into separate folders for each chapter of this book, so ensure your extraction utility is set to use folder names.

Errata

We've made every effort to make sure that there are no errors in the text or in the code. However, no one is perfect and mistakes do occur. If you find an error in one of our books, such as a spelling mistake or a faulty piece of code, we would be very grateful to hear about it. By sending in errata you may save another reader hours of frustration, and, of course, you will be helping us to provide even higher quality information. Simply e-mail the information to support@wrox.com – your information will be checked and, if correct, posted to the errata page for that title, and used in reprints of the book.

To find errata on the web site, go to www.wrox.com, and simply locate the title through our Advanced Search or title list. Click the Book Errata link below the cover graphic on the book's detail page.

E-Mail Support

If you wish to query a problem in the book with an expert who knows the book in detail, then e-mail support@wrox.com with the title of the book and the last four numbers of the ISBN in the subject field of the e-mail. A typical e-mail should include the following things:

- ❑ The **title of the book**, the **last four digits of the ISBN (7957)**, and the **page number** of the problem.
- ❑ Your **name, contact information**, and the **problem** in the body of the message.

We need the above details to save your time and ours – we *never* send unsolicited junk mail. When you send an e-mail message, it will go through the following chain of support:

- ❑ Customer Support – Your message is delivered to our customer support staff, who are the first people to read it. They have files on most frequently asked questions and will answer anything general about the book or the web site immediately.
- ❑ Editorial – Deeper queries are forwarded to the technical editor responsible for that book. They have experience with the programming language or particular product, and are able to answer detailed technical questions on the subject.
- ❑ The Authors – Finally, in the unlikely event that the editor cannot answer your problem, they will forward the request to the author. Wrox authors are glad to help support their books. They will e-mail the customer and the editor with their response, and again all readers should benefit.

The Wrox support process can only offer support for issues that are directly pertinent to the content of our published title. Support for questions that fall outside the scope of normal book support is provided via the community lists of our <http://p2p.wrox.com/> forum.

p2p.wrox.com

For author and peer discussion, join the P2P mailing lists. Our unique system provides **programmer to programmer™** contact on mailing lists, forums, and newsgroups, all in addition to our one-to-one e-mail support system. If you post a query to P2P, you can be confident that the many Wrox authors and other industry experts who are present on our mailing lists are examining it. At p2p.wrox.com, you will find a number of different lists that will help you not only while you read this book, but also as you develop your own applications. Particularly appropriate to this book are the `vs_dotnet` and `uml` lists.

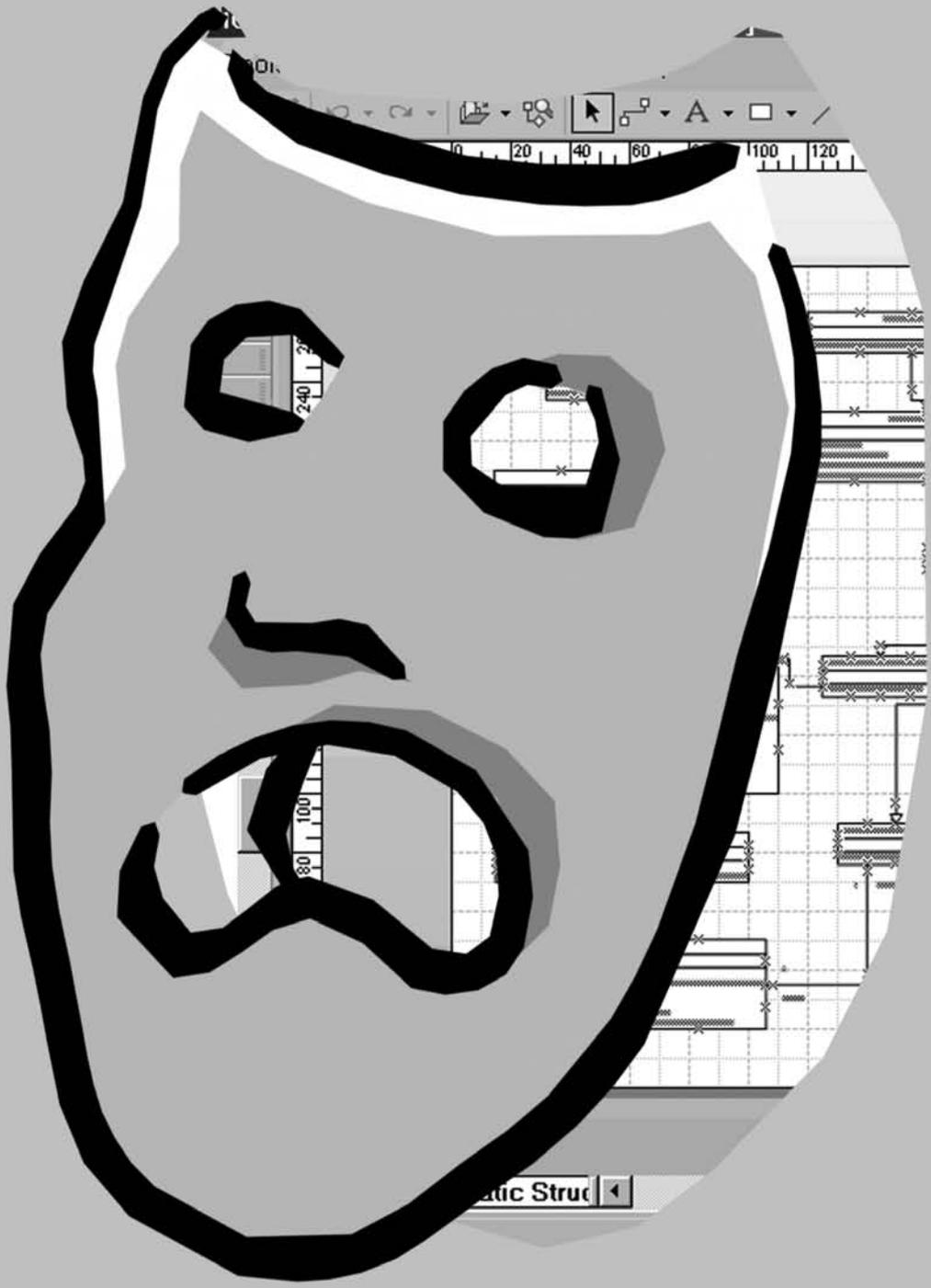
To subscribe to a mailing list, just follow these steps:

1. Go to <http://p2p.wrox.com/>.
2. Choose the appropriate category from the left menu bar.
3. Click on the mailing list you wish to join.
4. Follow the instructions to subscribe, and fill in your e-mail address and password.

5. Reply to the confirmation e-mail you receive.
6. Use the subscription manager to join more lists and set your e-mail preferences.

Why This System Offers the Best Support

You can choose to join the mailing lists, or you can receive them as a weekly digest. If you don't have the time (or the facility) to receive the mailing lists, then you can search our online archives. Junk and spam mails are deleted, and your own e-mail address is protected by the Lyris system. Queries about joining or leaving lists, and any other general queries about lists, should be sent to listsupport@p2p.wrox.com.



1

Review of UML

The purpose of this chapter is to set the scene by reviewing the key UML concepts, the main diagram types, and the role of those diagrams within the software development process. If you're quite new to UML this will serve as a practical introduction that will help you make sense of the rest of the book, before you move on to further reading. If you're experienced with UML the chapter will serve as handy revision and you might just find some nuggets of information that have so far eluded you.

Either way we'll all be moving on from roughly the same starting point: with the same appreciation of UML notation, with an understanding of relevant software development processes, and with a common bias towards .NET and the Visio for Enterprise Architects tool.

The final point is quite important, and the *raison d'être* for this book. In recent years the body of UML literature has focused mainly on Java development and the use of modeling tools such as Rational Rose. In this book we're applying a .NET development perspective at the same time as demonstrating the so far under-documented Visio modeling tool that comes bundled with the Visual Studio .NET Enterprise Architect.

With all this in mind we can now press on with the introduction to – or revision of, depending on your background – the Unified Modeling Language.

What is the Unified Modeling Language?

When discussing UML, we need to establish one important point right up front.

The Unified Modeling Language is a notation; that is a set of diagrams and diagram elements that may be arranged to describe the design of a software system. UML is not a process, nor is it a method comprising a notation and a process.

In theory you can apply aspects of the notation according to the steps prescribed by any process that you care to choose – traditional **waterfall**, **extreme programming**, **RAD** – but there are processes that have been developed specifically to complement the UML notation. You'll read more about the complementary process(es) later in this chapter.

Why use UML?

Hidden inside that specific question there's a more generic question, which is "**Why use a formal analysis and design notation, UML or otherwise?**" Let's start to answer that question by drawing an analogy.

Suppose you wanted to make a bridge across a small stream. You could just place a plank of wood across from one side to the other, and you could do so on your own. Even if it failed to hold your weight, the only downside would be wet feet.

Now suppose you wanted to make a bridge across a narrow river. You'd need to do some forward planning to estimate what materials you'd need – wood, brick, or metal – and how much of each. You'd need some help, and your helpers would want to know what kind of bridge you're building.

Finally, suppose you wanted to build a bridge across a very wide river. You'd need to do the same kind of forward planning as well as communicating your ideas to a much bigger team. This would be a commercial proposition with payback from fare-paying passengers, so you'd need to liaise with the relevant authorities and comply with health-and-safety requirements. You'd also be required to leave behind sufficient documentation to allow future generations to maintain the structure long into the future.

In a software context, this means that formal design becomes increasingly important as a function of the size and complexity of the project; in particular, as a function of the number of people involved. Based on that analogy, and wider project experience, we could conclude that a formal design notation is important in:

- Establishing a blueprint from the application
- Estimating and planning the time and materials
- Communicating between teams, and within a team
- Documenting the project

Of course, we've probably all encountered projects in which little or no formal design has been done up-front (corresponding with the first three bullet points in that list); in fact more projects than we care to mention! Even in those situations, UML notation has been found to be invaluable in documenting the end result (the last bullet point in that list). Though not recommended, if that's the extent of your commitment to UML you'll be most interested in the Reverse Engineering discussion in Chapter 5.

Now that we've answered the generic question, let's return to the specific question of **why use UML?**

Well it's become something of an **industry standard**, which means that there's a good chance of finding other people who understand it. That's very important in terms of the **communication** and **documentation** bullet points in our list. Also if you or anyone else in the team does not understand it, there's a good chance of finding relevant training courses, or books like this one.

That's very pragmatic reasoning and perhaps more convincing than a more academic (or even commercial) argument such as:

"The application of UML has a proven track record in improving the quality of software systems."

A Brief History of UML

Taking the phrase **Unified Modeling Language** as our starting point, we've discussed in the previous section the "language" (namely, notation) aspect. In the next section, we'll investigate the "modeling" aspect, which leaves us here with the word "unified". What, or who, preceded the UML and how did it all become **unified**? This will become clear as we step through a brief history of UML.

In the beginning although there was a plethora of object-oriented "methods", there were three principal methods:

- ❑ The **Booch** method devised by Grady Booch
- ❑ **Object Modeling Technique** (OMT) devised by Jim Rumbaugh
- ❑ **Object Oriented Software Engineering** (also known as Objectory) devised by Ivar Jacobson

These three methods have many ideas in common, yet different notation for expressing those ideas. Some of you may remember that in an OMT class diagram the classes were represented as rectangular boxes whereas in the Booch method they were represented as stylized cloud shapes. Also, each method placed emphasis on different aspects of object-oriented software development. For example Jacobson introduced the idea of use cases, not addressed by the other methods.

In simple terms, a use case is a unit of functionality provided by the system to an actor (such as a user). For example, in a word-processing application one of the use cases might be "Run spell checker".

The unification of these three methods combined the best bits of each method with a common notation (UML) for the common concepts – the end result being an industry-standard notation for analysis and design. If you speak with anyone who claims to be doing **object modeling**, chances are they'll be using UML.

So how did this unification play out in time? The key dates are:

- ❑ OOPSLA '94 – Jim Rumbaugh leaves General Electric to join Grady Booch at Rational Software, so as to merge their methods and achieve standardization across the industry.
- ❑ OOPSLA '95 – Booch and Rumbaugh publish version 0.8 of the **Unified Method**. Rational Software buys Objectory and Ivar Jacobson joins the company.
- ❑ January 1997 – Booch, Rumbaugh, and Jacobson (**the three amigos**) release – through Rational – a proposal for the UML version 1.0.
- ❑ September 1997 – UML version 1.1 is adopted by the Object Management Group (OMG).

The Object Management Group, previously best known for the CORBA standard, is a non-profit organization – comprising many member companies – that encourages, standardizes, and supports the adoption of object technologies across the industry. You can find out more about the OMG at <http://www.omg.org>.

If we've given the impression that the Unified Modeling Language is the exclusive work of only three contributors, the three amigos, then let's set the record straight. Some of the concepts are based in the early work of other individuals – for example, David Harel's work on Statechart diagrams – and some further enhancements have come from other member organizations of the OMG; for example, the Object Constraint Language (OCL) devised by IBM.

OCL was devised so that additional rules could be added to a UML model in a language that less ambiguous than English. For example, the statement "Person.Employer=Person.Manager.Employer" may be less ambiguous than "a person and their manager must both work for the same company."

More information on OCL can be found at <http://www-3.ibm.com/software/ad/library/standards/ocl.html>.

At the time of writing, the UML specification is at version 1.4 and in mid-2001 the OMG members started work on a major upgrade to UML 2.0. Modeling tools – including Visio for Enterprise Architects – will always be one or two steps behind in their support for the specification, but that's not usually a big problem because the core concepts discussed in the next section are now quite mature and stable.

At the time of writing, the version of Visio for Enterprise Architects used in the construction of this chapter provides support for UML at least up to version 1.2 – this can be determined from the *About error checking in the UML model* section of the Microsoft Visio Help:

"Semantic error checking occurs automatically, noting errors in the design of UML model elements, based on the well-formedness rules in the UML 1.2 specification."

End-to-End UML Modeling

Having looked at why UML is useful, and where it came from, we'll now look at the notation itself. To cover the complete notation in a single chapter would be impossible, so for a deeper coverage I'll refer you to some other works.

- ❑ **Instant UML** by Pierre-Alain Muller (Wrox Press, ISBN 1-86100-087-1).
- ❑ **The Unified Modeling Language User Guide** by Grady Booch, James Rumbaugh, and Ivar Jacobson (Addison Wesley, ISBN 0-201-57168-4).
- ❑ **UML Distilled** by Martin Fowler with Kendall Scott (Addison Wesley, ISBN 0-201-65783-X).

What we'll do here is cover the essential notation and core concepts that will allow us to progress through the rest of the book with a common understanding.

We'll also aim to address one of the problems of many UML courses and books. The problem being, that all too often the various diagrams are presented in isolation without a clear indication of how they relate to one another. To make matters worse, different examples are often used to demonstrate the different diagrams, not one of those examples being for a system that you might actually want to build. Think here of a **statechart diagram** that describes a motor car gearbox, or a **sequence diagram** that describes the operation of a hotel elevator.

So in the following section we'll have a single example, an Order Processing system, which you should be able to relate to even if you don't intend to build such a thing, and at the end, we'll pull it all together.

UML Essential Notation and Core Concepts

Now we'll step through the UML diagrams in turn, all the way from an **activity diagram** through to a **deployment diagram** in this order:

- ❑ Activity Diagram
- ❑ Use Case Diagram
- ❑ Sequence and Collaboration Diagram
- ❑ Statechart Diagram
- ❑ Static Structure Diagram
- ❑ Component Diagram
- ❑ Deployment Diagram

Each diagram is labeled in light gray with some of the names given to the UML elements that are shown, which – for the record – reflects the **UML metamodel**.

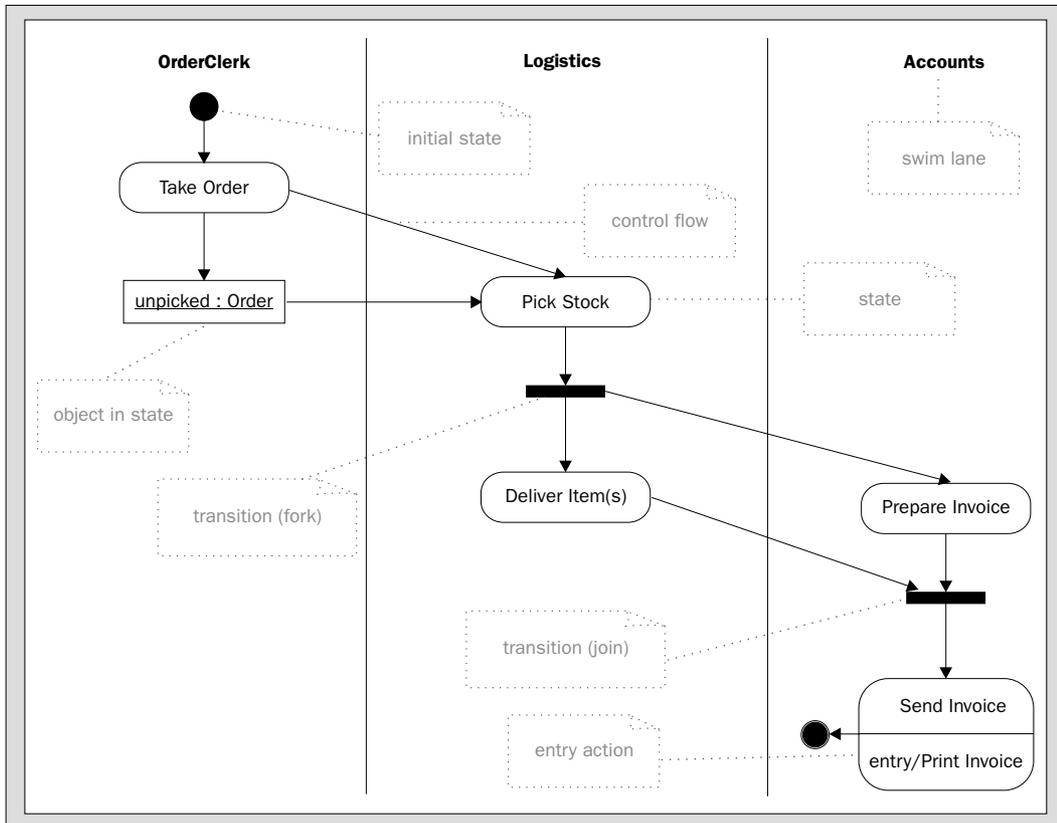
The UML Metamodel is itself a UML model, which defines the rules for constructing other UML models. Whereas in one of your own models you might state "Bank is associated with one or more Accounts", the metamodel would state a more generic relationship of "a Class may be associated with any Other Class".

On the whole, the model elements have been labeled using Visio EA terminology so as to reduce the potential for confusion when you come to use the tool. Historically – and in other modeling tools – you may have encountered alternative UML terminology. The alternative terms have been tabulated towards the end of this chapter.

As you'll see later in this chapter, the software development process that you follow might well be described as **use-case driven**, which implies the **use case diagram** as an obvious starting point. But those use cases will doubtless fit into some kind of overall **business process**, perhaps modeled up-front by a business analyst. So we'll take a business process as our starting point and use this as a vehicle for demonstrating the most suitable diagram for that purpose; the **activity diagram**.

Activity Diagram

The activity diagram is the closest you'll get in UML to a **flow chart**, and the closest you'll get to a **business process diagram**. Here is a sample activity diagram with the important UML elements labeled, followed by a description of those elements.



- ❑ **Initial state** is where the diagram begins.
- ❑ **Control flow** shows a transfer of control from one activity to another.
- ❑ **State** represents a period of time during which a piece of work is carried out by person or team.
- ❑ **Transition (fork)** shows the point as which two or more parallel activities will commence.
- ❑ **Transition (join)** shows the point as which two or more parallel activities must synchronize and converge.
- ❑ **Swim lane** allows all of the activities carried out by a particular person or team arranged into a column.
- ❑ **Entry action** shows what must happen when the activity begins.
- ❑ **Object in state** shows an object that is produced or consumed in the course of an activity, with the production or consumption (object flow) being represented by the dashed line.