Introduction

This book is about application architecture, design, and development in .NET using object-oriented concepts. The focus is on business-focused objects called *business objects*, and how to implement them to work in various distributed environments, including web and client/server configurations. The book makes use of a great many .NET technologies, object-oriented design and programming concepts, and distributed architectures.

The first half of the book walks through the process of creating a framework to support object-oriented application development in .NET. This will include a lot of architectural concepts and ideas. It will also involve some in-depth use of advanced .NET techniques to create the framework.

The second half of the book makes use of the framework to build a sample application with several different interfaces. If you wish, it's perfectly possible to skip the first half of the book and simply make use of the framework to build object-oriented applications.

One of my primary goals in creating the CSLA .NET framework was to simplify .NET development. Developers using the framework in this book don't need to worry about the details of underlying technologies such as remoting, serialization, or reflection. All of these are embedded in the framework, so that a developer using it can focus almost entirely on business logic and application design, rather than getting caught up in "plumbing" issues.

From .NET 1.0 to 2.0

This book is a major update to the previous edition: *Expert C# Business Objects*. This updated book takes advantage of new features of .NET 2.0 and applies lessons learned by using .NET 1.0 and 1.1 over the past few years.

This book is nearly identical to the *Expert VB 2005 Business Objects* book—the only difference between the two books is the syntax of the programming languages.

Both the VB and C# books are the most recent expression of concepts I've been working on for nearly a decade. My goal all along has been to enable the productive use of object-oriented design in distributed n-tier applications. Over the years, both the technologies and my understanding and expression of the concepts have evolved greatly.

The VB 5 and 6 books that started this whole process discussed how to use VB, COM, DCOM, MTS, and COM+ to create applications using object-oriented techniques. (Or at least they were as object-oriented as was possible in VB 5/6 and COM.) They also covered the concept of *distributed objects*, whereby a given object is "spread" over multiple machines in a physical n-tier environment. In COM, this isn't a trivial thing to implement, and so these books included a fair amount of discussion relating to object state and state serialization techniques.

The end result was an architecture that I called *CSLA*, for component-based, scalable, logical architecture. Over the years, I've received hundreds of emails from people who have used CSLA as a basis for their own architectures as they've built applications ranging from small, single-user programs to full-blown enterprise applications that power major parts of their businesses.

In .NET, the idea of *distributed objects* has given way to the more appropriate idea of *mobile objects*, where objects actually move between computers in an n-tier environment. At a high level, the architecture is comparable, but mobile objects provide a far more powerful way to implement object-oriented designs in distributed environments.

I've also received a handful of emails from people for whom CSLA .NET *wasn't* successful, but this isn't surprising. To use CSLA .NET effectively, you must become versed in object-oriented and component-based design, understand the concept of distributed objects, and develop a host of other skills. The mobile object architecture has many benefits, but it's not the simplest or the easiest to understand.

Designing CSLA .NET

One of the characteristics of .NET is that it often provides several ways to solve the same problem. Some of the approaches available will be better than others, but the best one for a given problem may not be immediately obvious. Before writing the .NET 1.0 books, I spent a lot of time trying various approaches to distributing objects. Although a variety have proven to work, in the end I've arrived at the one that best matches my original goals.

Before I discuss those goals, I think it's important to talk about one other issue that I wrestled with when writing this book. Given the large number of people using the concepts and code from the previous edition of the book, I wanted to preserve backward compatibility whenever possible. At the same time, this new edition of the books is an opportunity to not only use .NET 2.0 features, but also to apply lessons learned by using .NET over the past several years.

Applying those lessons means that using the new concepts and code requires changes to existing business objects and user interface code. I don't take backward compatibility lightly, yet it is important to advance the concepts to keep up with both changes in technology and my views on both object-oriented and distributed computing.

When possible, I have minimized the impact on existing code, so the transition shouldn't be overly complex for most applications.

I have a specific set of goals for the architecture and the book. These goals are important, because they're key to understanding why I made many of the choices I did in terms of which .NET technologies to use, and how to use them. The goals are as follows:

- To support a fully object-oriented programming model
- To allow the developer to use the architecture without jumping through hoops
- To enable high scalability
- To enable high performance
- To provide all the capabilities and features of the original CSLA, namely:
  - N-level undo on a per-object basis (edit, cancel, apply)
  - Management of validation rules
  - Management of authorization rules
  - Support for many types of UI based on the same objects
  - Support for data binding in Windows and Web Forms
  - Integration with distributed transaction technologies such as Enterprise Services and `System.Transactions`
- To simplify .NET by handling complex issues like serialization, remoting, and reflection
- To use the tools provided by Microsoft, notably IntelliSense and the Autocomplete in Visual Studio .NET

Of these, saving the developer from jumping through hoops—that is, allowing him or her to do "normal" programming—has probably had the largest impact. To meet all these goals without a framework, the developer would have to write a lot of extra code to track business rules, implement n-level undo, and support serialization of object data. All this code is important, but adds nothing to the business value of the application.

Fortunately, .NET offers some powerful technologies that help to reduce or eliminate much of this "plumbing" code. If those technologies are then wrapped in a framework, a business developer shouldn't have to deal with them at all. In several cases, this goal of simplicity drove my architectural decisions. The end result is that the developer can, for the most part, simply write a normal C# class, and have it automatically enjoy all the benefits of n-level undo, business rule tracking, and so forth.

It has taken a great deal of time and effort, but I've certainly enjoyed putting this architecture and this book together, and I hope that you will find it valuable during the development of your own applications.

What's Covered in This Book?

This book covers the thought process behind the CSLA .NET 2.0 architecture, describes the construction of the framework that supports the architecture, and demonstrates how to create Windows Forms, Web Forms, and Web Services applications based on business objects written using the framework.

Chapter 1 is an introduction to some of the concepts surrounding distributed architectures, including logical and physical architectures, business objects, and distributed objects. Perhaps more importantly, this chapter sets the stage, showing the thought process that results in the remainder of the book.

Chapter 2 takes the architecture described at the end of Chapter 1 and uses it as the starting point for a code framework that enables the goals described earlier. By the end, you'll have seen the design process for the objects that will be implemented in Chapters 4 and 5; but before that, there's some other business to attend to.

Chapters 3 through 5 are all about the construction of the CSLA .NET framework itself. If you're interested in the code behind n-level undo, mobile object support, validation rules, authorization rules, and object persistence, then these are the chapters for you. In addition, they make use of some of the more advanced and interesting parts of the .NET Framework, including remoting, serialization, reflection, .NET security, Enterprise Services, `System.Transactions`, strongly named assemblies, dynamically loaded assemblies, application configuration files, and more.

The rest of the book then focuses on creating an application that makes use of the architecture and framework. Even if you're not particularly interested in learning all the lower-level .NET concepts from Chapters 3 through 5, you can take the framework and build applications based on it by reading Chapters 6 through 12.

In Chapter 6, I discuss the requirements of a sample application and create its database. The sample application uses SQL Server and creates not only tables but also stored procedures in order to enable retrieval and updating of data.

Chapter 7 discusses how to use each of the primary base classes in the CSLA .NET framework to create your own business objects. The basic code structure for editable and read-only objects, as well as collections and name/value lists, is discussed.

Chapter 8 creates the business objects for the application. This chapter really illustrates how you can use the framework to create a powerful set of business objects rapidly and easily for an application. The end result is a set of objects that not only model business entities, but also support n-level undo, data binding, and various physical configurations that can optimize performance, scalability, security, and fault tolerance, as discussed in Chapter 1.

Chapter 9 demonstrates how to create a Windows Forms interface to the business objects. Chapter 10 covers the creation of a Web Forms or ASP.NET interface with comparable functionality.

In Chapter 11, Web Services are used to provide a programmatic interface to the business objects that any web service client can call.

Finally, Chapter 12 shows how to set up application servers using .NET Remoting, Enterprise Services, and Web Services. These application servers support the CSLA .NET framework and can be used interchangeably from the Windows Forms, Web Forms, and Web Services applications created in Chapters 8 through 11.

By the end, you'll have a framework that supports object-oriented application design in a practical, pragmatic manner. The framework implements a logical model that you can deploy in various physical configurations to optimally support Windows, web, and Web Services clients.

Framework License

9. Patents. If you sue anyone over patents that you think may apply to the Software for a person's use of the Software, your license to the Software ends automatically.

   The patent rights, if any, licensed hereunder, only apply to the Software, not to any derivative works you make.

10. Termination. Your rights under this License end automatically if you breach it in any way.

    Rockford Lhotka reserves the right to release the Software under different license terms or to stop distributing the Software at any time. Such an election will not serve to withdraw this Agreement, and this Agreement will continue in full force and effect unless terminated as stated above.

11. Governing Law. This Agreement shall be construed and enforced in accordance with the laws of the state of Minnesota, USA.

12. No Assignment. Neither this Agreement nor any interest in this Agreement may be assigned by Licensee without the prior express written approval of Developer.

13. Final Agreement. This Agreement terminates and supersedes all prior understandings or agreements on the subject matter hereof. This Agreement may be modified only by a further writing that is duly executed by both parties.

14. Severability. If any term of this Agreement is held by a court of competent jurisdiction to be invalid or unenforceable, then this Agreement, including all of the remaining terms, will remain in full force and effect as if such invalid or unenforceable term had never been included.

15. Headings. Headings used in this Agreement are provided for convenience only, and shall not be used to construe meaning or intent.

What You Need to Use This Book

The code in this book has been verified to work against Microsoft Visual Studio 2005 Professional, and therefore against version 2.0 of the .NET Framework. The database is a SQL Express database, and SQL Express is included with Visual Studio 2005 Professional. The Enterprise version of VS 2005 and the full version of SQL Server are useful, but not necessary.

In order to run the tools and products listed previously, you'll need at least one PC with Windows 2000, Windows Server 2003, or Windows XP Professional Edition installed. To test CSLA .NET's support for multiple physical tiers, of course, you'll need an additional PC (or you can use Virtual PC or a similar tool) for each tier that you wish to add.

Conventions

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

Code has several fonts. If I'm talking about code in the body of the text, I use a fixed-width font like this: `foreach`. If it's a block of code that you can type as a program and run, on the other hand, then it will appear as follows:

```
if (Thread.CurrentPrincipal.Identity.IsAuthenticated)
{
  pnlUser.Text = Thread.CurrentPrincipal.Identity.Name;
  EnableMenus();
}
```

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

```
dgProjects.DataSource = ProjectList.GetProjectList();
DataBind();

// Set security
System.Security.Principal.IPrincipal user;
user = Threading.Thread.CurrentPrincipal;
```

When this happens, the code with a normal font is code you're already familiar with, or code that doesn't require immediate action. Lines in bold font indicate either new additions to the code since you last looked at it, or something that I particularly want to draw your attention to.

Tip Advice, hints, and background information appear in this style.

Note Important pieces of information are included as notes, like this.

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

- *Important words* are in italics.

How to Download Sample Code for This Book

Visit the Apress website at www.apress.com, and locate the title through the Search facility. Open the book's detail page and click the Source Code link. Alternatively, on the left-hand side of the Apress homepage, click the Source Code link, and select the book from the text box that appears.

Download files are archived in a zipped format, and need to be extracted with a decompression program such as WinZip or PKUnzip. The code is typically arranged with a suitable folder structure, so make sure that your decompression software is set to use folder names before extracting the files.

Author and Community Support

The books and CSLA .NET framework are also supported by both the author and a large user community.

The author maintains a website with answers to frequently asked questions, updates to the framework, an online discussion forum, and additional resources. Members of the community have created additional support websites and tools to assist in the understanding and use of CSLA .NET and related concepts.

For information and links to all these resources, visit www.lhotka.net/cslanet.