Introduction

You probably picked up this book because of the buzzwords *Ajax, REST,* and *patterns*. You will probably read this introduction and skim through the pages. But I want to stop you from skimming through the pages, at least for a moment. I want you to read this introduction and then decide whether you want to buy the book.

Here are the things you need to know about Ajax:

- Ajax is an acronym, and the ramifications of the acronym are immense.
- Ajax is not just about a fat client, JavaScript, XML, or asynchronous behavior, but about developing the next generation of web applications.
- We are at the beginning of building the next generation of web applications.

You are still reading, and that means I still have your interest, which is a good thing. So now let me tell you what this book is about:

- Using Ajax is the act of creating a web application that implies using REST, that implies using HTTP, and that implies using the Internet. The patterns of this book illustrate how JavaScript can be used to control the `XMLHttpRequest` object to make `XMLHttpRequest` calls that process XML or HTML.
- This book for the server side focuses on using Java and C# .NET. The patterns can be used with Python or Ruby on Rails. I focus on Java and C# because at the time of this writing I feel that most developers are using them. In the next edition of this book, I want to extend the materials to include Python and Ruby on Rails examples, because I happen to be an avid Python programmer.
- The patterns in this book can be used in other contexts, such as Flex (Flash Ajax). For example, the Permutations pattern can be used to generate Flex content.

Good, you're still reading and haven't closed the book. That means you are still interested and probably willing to spend a few more moments. Here is what I suggest: finish reading the Introduction because it includes a road map of the patterns. Skim Chapter 1 to get an idea of what Ajax does and is. Then skim the patterns and focus on reading the "Motivation" and "Architecture" sections. And if after that you are still interested, please buy this book because the remaining sections fill in the details of what the patterns are trying to achieve. If you would like to experiment with the patterns, I suggest you surf to the site `http://www.devspace.com/ajaxpatterns`, which will either have the live patterns or redirect you to where the live patterns are.

What's My Vision of Ajax?

Philosophizing about the vision of Ajax raises the question of what Ajax really is. Some say Ajax is a client-side–only technology. Some say it is an extension of a server framework. Yet others say, "Heck, it's new technology blah and now technology bleh can be ignored." However, ignoring REST is like saying if it's a liquid, it can be drunk by humans. Sure, humans can drink anything that is a liquid, but the bigger question is, will you survive? Sometimes you will, and sometimes you won't! Drinking without questioning is playing Russian roulette. The same can be said of writing Ajax that ignores REST, ignores XML, ignores JSON, and ignores JavaScript. Ajax is Ajax because of these new technologies that can be combined in new and interesting ways.

My vision of Ajax goes beyond the technologies and represents a new way of building applications. In the early days when building web applications, the server was responsible for generating the content, navigating the content, and controlling the content. The web application became a technology that depended on the sophistication of the server framework to determine whether the application would be interactive. Ajax breaks those bonds!

Ajax breaks those bonds because it decouples the client from the server. An Ajax application still needs a server, but an Ajax application can decide when, where, and how that content will be delivered. A web application that relies on the server is a tightly coupled web application that can exist only while the server exists. Any content required by the client is controlled by the server. With Ajax, content can be focused because pieces of content can be assembled, as illustrated by the Content Chunking pattern.

Where I become very concerned with respect to Ajax is when individuals like to sell a server framework that relies on said framework to implement Ajax. If Ajax is about decoupling the client from the server, why must a server framework be used to implement Ajax? The logic simply does not make any sense. I can understand the argument that a framework has extensions to enable Ajax-like architectural designs. But I cannot accept the argument that a sever framework is necessary to make an Ajax application happen.

The focus of this book is the advantages of Ajax using specific patterns that, among other techniques, extensively use de-coupling to create architectures that can be maintained and extended. I personally believe that productivity is a good thing, but in specific situations what may be more important is the ability to figure out what you did and why you did it.

Book and Pattern Road Map

The book is pattern based, with the exception of Chapters 1 and 2. Following is the road map for those first two chapters:

- **Chapter 1**: This chapter is used as an introduction to the book and the topic of Ajax. The focus of the chapter is to provide the context of Ajax and to compare an Ajax application to other methodologies (for example, traditional client-server).
- **Chapter 2**: This chapter introduces the `XMLHttpRequest` object. When you are writing Ajax applications, the `XMLHttpRequest` object is a core technology used to communicate with an HTTP server. Best practices when using the `XMLHttpRequest` object are also illustrated.

Chapter 3 and thereafter present patterns. A hierarchy of the patterns is illustrated in Figure 1.

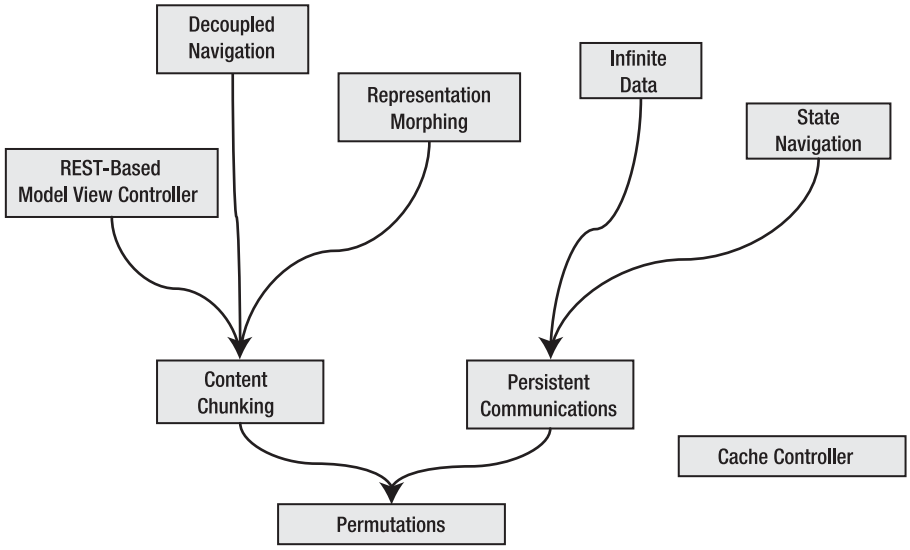

Figure 1. *Hierarchy of patterns explained in the book*

The road map for the patterns of Figure 1 is as follows:

- **Chapter 3—Content Chunking pattern**: Makes it possible to incrementally build an HTML page, allowing the logic of an individual HTML page to be distributed, and allowing the user to decide the time and logic of the content that is loaded.
- **Chapter 4—Cache Controller pattern**: Provides the caller a mechanism to temporarily store resources in a consistent manner, resulting in an improved application experience for the caller.
- **Chapter 5—Permutations pattern**: Used by the server to separate the resource (URL) from the representation (for example, HTML or XML). This separation makes it possible for an end user to focus on the resource and not have to worry about the content. For example, if a client's bank account is at the URL `http://mydomain.com/accounts/user`, the same URL can be used regardless of device (phone, PC, and so on).
- **Chapter 6—Decoupled Navigation pattern**: Defines a methodology indicating how code and navigation on the client side can be decoupled into smaller modular chunks, making the client-side content simpler to create, update, and maintain.
- **Chapter 7—Representation Morphing pattern**: Combines the state with a given representation, and provides a mechanism whereby the representation can morph from one representation to another without losing state.
- **Chapter 8—Persistent Communications pattern**: Provides a mechanism whereby a server and a client can communicate on a continuing basis, allowing the server to send data to the client, and vice versa, without prior knowledge.

- **Chapter 9—State Navigation pattern**: Provides an infrastructure in which HTML content can be navigated, and the state is preserved when navigating from one piece of content to another.
- **Chapter 10—Infinite Data pattern**: Manages and displays data that is seemingly infinite, in a timely manner.
- **Chapter 11—REST-Based Model View Controller pattern**: Accesses content that is external to the web application and transforms the content so that it appears as if the web application generated it.