3

ER Diagrams, Domain Model, and N-Layer Architecture

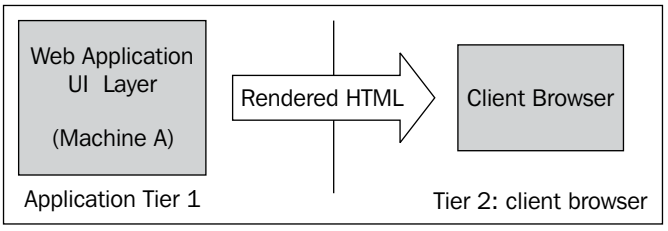
In the last chapter, we saw the basic layering of the monolithic 1-tier 1-layer architectural style in action, with the UI layer having code-behind classes as the sub-layer. This 1-tier 1-layer architecture is the default style in ASP.NET and Visual Studio 2005/2008. To overcome the limitations of this style, we can further break the application code into n-layers, where the number "n" actually depends on the project requirements.

In this chapter we will:

- Learn the 2-layer style
- Understand ER diagrams
- Understand what a domain model is, and what UML relationships are
- Learn the 3-layer style
- Learn about object data source controls

Let's revisit the 1-tier ASP.NET application configuration described in the last chapter.

Note that the application as a whole including database and client browser is three tier.



As mentioned in the last chapter, we can call this 1-tier architecture a 3-tier architecture if we include the browser and database (if used). For the rest of this chapter we will ignore the database and browser as separate tiers so that we can focus on how to divide the main ASP.NET application layers logically, using the n-layer pattern to its best use.

We will first try to separate the data access and logical code into their own separate layers and see how we can introduce flexibility and re-usability into our solution. We will understand this with a sample project. Before we go ahead into the technical details and code, we will first learn about two important terms: ER Diagram and Domain Model, and how they help us in getting a good understanding of the application we need to develop.

Entity-Relationship Diagram

Entity-Relationship diagrams, or ER diagrams in short, are graphical representations depicting relationships between different entities in a system. We humans understand and remember pictures or images more easily than textual information. When we first start to understand a project we need to see how different entities in the project relate to each other. ER diagrams help us achieve that goal by graphically describing the relationships.

An entity can be thought of as an object in a system that can be identified uniquely. An entity can have attributes; an attribute is simply a property we can associate with an entity. For example, a Car entity can have the following attributes: EngineCapacity, NumberofGears, SeatingCapacity, Mileage, and so on. So attributes are basically fields holding data to indentify an entity. Attributes cannot exist without an entity.

Let us understand ER diagrams in detail with a simple e-commerce example: a very basic Order Management System. We will be building a simple web based system to track customer's orders, and manage customers and products. To simplify the learning curve, we will use this example regularly in the coming chapters, to see how different architectural styles shape the solution differently, and how we can move towards a more scalable n-tier system.

To start with, let us list the basic entities for our simplified Order Management System (OMS):

- **Customer**: A person who can place Orders to buy Products.
- **Order**: An order placed by a Customer. There can be multiple Products bought by a Customer in one Order.