As we can see, using object data source controls complements the domain model and helps us avoid writing the UI layer code, as we can directly bind custom entities to data-bound controls in UI.

However, there are a few issues in using object data source controls:

- They are not very flexible. Sometimes we need to display data in UI in a complex way (which can be user friendly, but code un-friendly). In such cases, it is best to use and manipulate custom collections using manual coding; you cannot even extend the control to customize it.
- There is a slight performance hit when using object data source controls instead of using manual coding. This hit comes through the use of reflection by the control to access a class's attributes and methods.

[Reflection is a technique which VS uses to get metadata information about other entities, such as a class file or an assembly. Using reflection, the object data source control will first "read" the class to get all of the attributes and then use this metadata to connect to and perform operations on the class. On account of this additional "reading" step, the application suffers a performance hit.]

Therefore, for a flexible approach, it is best to use custom code. But for small projects where we don't foresee any major complexity and performance issues, object data source controls are a good option to save on development time while supporting a flexible n-layer option.

Summary

All of the samples we covered in this chapter were of the 1-tier n-layer style. We learned how to create a 1-tier 2 layer architecture using logical code separation. Then we focused on the need for a 3-layered solution and examined ER-diagrams, domain models and UML, all of which are important tools that aid in the understanding of commercial projects required to build a 3-layered structure.

Then we focused on a 3-layered application structure for OMS, and looked at how both custom code and object data source controls can be used in this architecture.

In the coming chapters, we will discuss the need to physically separate this 3-layered code into "tiers", and see how we need a different model to achieve this goal and create scalable n-tier applications.