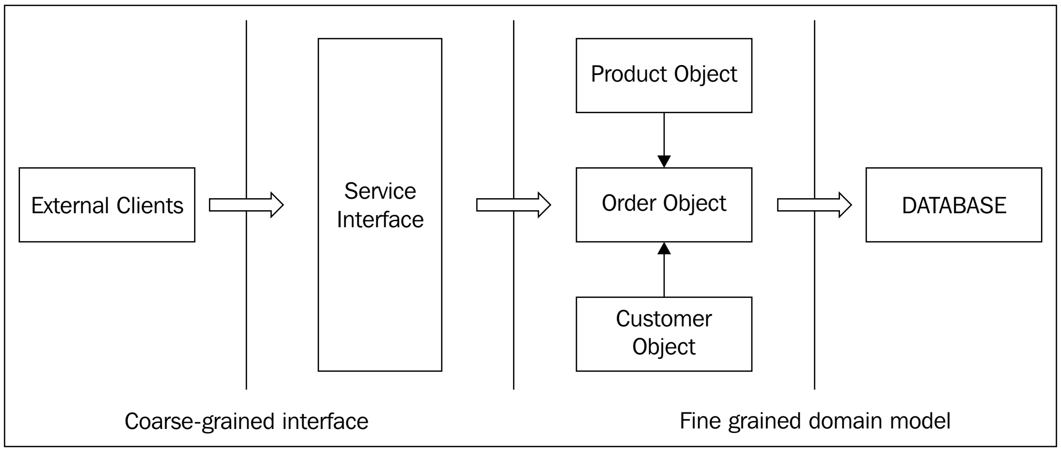
In traditional object-oriented systems, external clients (outside applications) need to understand the application's domain model in order to interact with it. The domain model was quite complex, and the external clients had to understand the fine-grained API, or use complex middleware technologies such as COM/COM+ to integrate applications.

With SOA, a service interface was built and the methods were exposed as coarse-grained units, performing individual database transactions to fetch/put data. And web services allowed easy portability and integration across platforms.



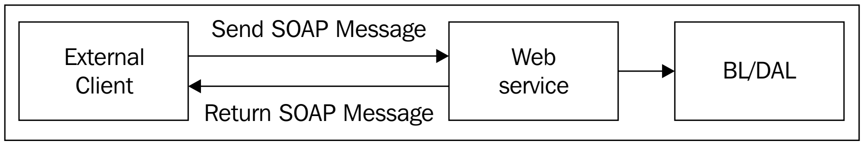
So we can abstract the conceptual fine-grained domain model by building an interface layer over it, which has relatively coarse-grained methods when compared to those in the model. These methods in turn interact with the domain model and provide an easy-to-use interface for external clients. Such coarse-grained methods can easily be implemented using XML web services.

XML Web Services and SOAP Messages

XML web services are software components used to communicate and share data between distributed systems. XML web services are simple to build and use, and can talk to disparate systems across networks.

We can use XML web services to create an SOA based framework. Using web services, we can create message-based systems, and implement web methods that are complete in themselves. This means that our web methods should not depend on or call other web methods, and should be independent entities in themselves, so that they can be called as a "message" by external clients. If the web methods depend on each other, then the system will become tightly-coupled, and break the basic principle of SOA, which is to keep components loosely coupled.

External clients should be able to talk to our system using messages, as shown here:



The above diagram depicts an XML web service using the Simple Object Access Protocol (SOAP) for exchanging data across the systems. SOAP is a lightweight XML-based protocol. SOAP can be used to serialize objects and data across HTTP using the XML format.

Behind the service interface, we can use our normal domain model. The main idea behind using web services is to make each part or module independent of the other parts or modules, in order to achieve a higher degree of loose coupling. We will use coarse-grained web services to achieve fewer calls, and wrap fine-grained business logic, exposing it as coarse-grained messages to the outside world.

Sample Project

Let's study a sample project using XML web services with VS 2008, and create a service interface for our Order Management System. We will perform the following:

1. Use the 5-tier solution we created in Chapter 4.
2. Create a service interface around it using XML web services in an SOA-like fashion.
3. Allow the GUI to talk to this interface instead of using the business layer directly.

[We can also use the sample code created in Chapter 5 using MVC. MVC and SOA complement each other, and one should not get confused with the question, "should I use MVC or SOA?" MVC has a purpose different from that of SOA, and can be used in any SOA implementation.]