Remoting is a system for Microsoft .NET platform that enables interprocess communication in a homogenous environment. With the help of remoting, we can communicate between two processes running on different machines (or different application domains) in a network (but having the same OS). Because remoting uses binary serialization, it is much faster than using web services. However, we cannot use remoting in a heterogeneous environment (say, where an ASP.NET web application is talking to a J2EE application) using a .NET specific API, as other systems, such as J2EE, cannot understand it. Also, the use of remoting when we want to talk to processes outside of the LAN (say, when communicating using WAN or Internet) can be troublesome, as it cannot bypass a firewall until specific access is given to the service. Using web services is preferred in such cases.

In this section, we will understand how to use WCF instead of ASMX services. We will not be going into detail as this chapter is more focused on the use of WCF for SOA, and is not intended to serve as a primer to WCF.

A WCF service is composed of three main components:

- **A WCF service class:** This contains the actual implementation of the methods. This class is similar in purpose to the class defined in the `ASMX.CS` code file earlier. This class will hold the code logic for the WCF Service.
- **Host Environment:** Any parent process that can host the service, such as a console application, or an IIS or Windows application.
- **Service Endpoints:** These are the communication endpoints of the service. Each service can have a collection of endpoints. Each end point is a combination of address, binding, and contract. End points can be created through code as well as through the configuration class. These end points act as the "gateways" for communicating with the outside world.

WCF treats data exchange between the applications more like a message exchange, rather than treating it is as a simple procedure call (which is how default ASMX web services are treated using web methods).

Sample Project using WCF

Let us understand WCF through a sample project. We will modify the project we started above to use WCF services instead of ASMX services. Open the same solution in which we created the previous `OMSService` and add a new WCF service file named `OMSWCFService.svc` to the `NTier.Services` project.

You will notice that VS automatically creates `IOMSWCFService.cs` and adds the necessary service registration information in the `web.config` file.

```
namespace NTier.Services
{
    [ServiceContract]
    public interface IOMSWCFService
    {
        [OperationContract]
        void DoWork();
    }
}
```

This file contains the definition of the interface that will be used as a service contract.

If you want, you can define the implementation class immediately, without defining the interface. In this case you would need to use the `ServiceContract` and `OperationContract` attributes directly in the implementation class.

Now, we need to define the methods that we want to expose in our implementation class. Let us put the method signature of the `GetAllProducts()` method here:

```
using NTier.Business;
using NTier.Common.DTO;
namespace NTier.Services
{
    [ServiceContract]
    public interface IOMSWCFService
    {
        [OperationContract]
        ProductDTO[] GetAllProducts(int);
    }//end interface
}//end class
```

The implementation class will look like this:

```
namespace NTier.Services
{
    // NOTE: If you change the class name "OMSWCFService" here, you
must also update the reference to "OMSWCFService" in Web.config.
    public class OMSWCFService : IOMSWCFService
    {
        /// <summary>
        /// Get all products
```