A Case Study: 
A Distributed Web Application

Internet technology has been changing people’s lifestyles and business practice models. Now, the Internet itself is experiencing a paradigm shift. It is quickly evolving from Web sites that deliver simple user interface pages to a next generation of programmable Web-based applications.

In previous chapters, we discussed major features of the .NET technology, including ASP.NET, ADO.NET, and Web Services. Not only do these features make development of Web-based applications much easier and simpler, they also allow for more powerful, distributed information-processing systems to be built based on the Internet. In such a system, ADO.NET interfaces the data sources, ASP.NET facilitates browser-based information presentation, and Web Services deliver functional services over the Web. Furthermore, existing applications may also be integrated in this framework.

In the case study presented in this chapter, we build a Web-based distributed application for a group of travel agencies that have formed a loosely coupled consortium called Travel Agency Union, or TAU, in order to share their business opportunities. Information systems of the participant agencies are extended using .NET technology, creating an integrated system over which information can be exchanged. We call this integrated system TAUNET. TAUNET unites the participating agencies to form a large entity that can help agencies better serve their customers.
11.1 The TAU Business Model

Each TAU member is a travel agency that runs its business independently, providing its customers with products and services, such as flight reservations, hotel reservations, rental car reservations, and group tours and cruises. Each agency processes its business transactions using the following functions.

- **Inquiry.** Allows the agency and/or its customers to retrieve products and services information
- **Reservations.** Allows the agency and/or its customers to reserve and manipulate the transaction with desired products and services
- **Bill/Payment.** Bills the customers and acquires the payment from the customers as well as receives the invoices and makes the payment to the vendors

By joining TAU, member agencies can expand their functionality into the distributed environment of the Internet. The enhanced functions include

- **Inquiry.** A member agency may allow other member agencies to retrieve its products and services. In addition, a customer may retrieve the products and services via a single entry point, the Web site of his or her own agency.
- **Reservations.** Via the Web site of a single agency, a customer may manipulate his or her transaction with products and services of multiple TAU agencies, including reserve products and/or services, as well as update and cancel reservations.
- **Bill/Payment.** For a transaction involving multiple TAU agencies, the agency that directly interacts with the customer calculates the total cost and bills the customer. Other agencies that are involved in this transaction maintain their own accounts receivable information for the products and services they provide.

TAU.NET is a business network that provides users with a single entry point to accessing all the resources shared by its members. The business concept of TAU.NET is shown in Figure 11-1.
11.2 The Goals of TAU.NET System Design

11.2.1 Cell Systems

Before joining TAU, each member agency runs its business using a proprietary system of its own. We call these local agency systems the Cell Systems. The Cell Systems are heterogeneous in nature; they may be different in terms of system design, data structure, data storage, and user interface.
Each desired TAU functionality must be implemented as an extension of the Cell Systems. That is, there should be no central server in the TAU system. The Cell Systems, with the support of the TAU extensions, interact over the Internet to fulfill the goal of the distributed system. The TAU extensions are discussed in Section 11.2.4.

11.2.2 Information Exchange

Due to the heterogeneity of the Cell Systems, the distributed functions have to be supported by information exchange between their data storages. A standard data format is needed in order for the Cell Systems to share information. XML serves as an ideal vehicle to carry such information over the Internet.

11.2.3 Remote Services

In a distributed environment without a central server, each Cell System is extended to provide remote services over the Internet. These services allow the Cell System to communicate with its partner Cell Systems in order to exchange information that supports the distributed functions. Web Services are the natural choice for this purpose.

11.2.4 TAU.NET Node

A TAU.NET node is the extension for the Cell System of a participant agency, through which the Cell System joins the network of TAU. The Cell Systems communicate with each other via their TAU.NET nodes.

Communication between the TAU.NET nodes is based on Web Services. On the one hand, the TAU.NET acts as a server supplying Web methods with a standard interface. On the other hand, the TAU.NET node serves as a client supplying proxies whereby the remote Web methods can be called. In other words, the TAU.NET nodes constitute a middle tier that hooks the Cell Systems of the participant agencies so as to form an integral distributed system.

Figure 11-2 illustrates the concepts of TAU.NET Node and TAU.NET members.
Figure 11-2 Concept of TAU.NET.
11.3 Architecture

11.3.1 Overview

As discussed in the previous section, a TAU.NET node is implemented as an extension for the Cell Systems under .NET Framework. It supports the communication between the Cell Systems of the participant agencies.

A TAU.NET member system is composed of a TAU.NET Node and a Cell System. The TAU.NET Node consists of two sets of components, TAU.NET Node Interface and TAU.NET Node Adaptor.

As shown in Figure 11-3, the Cell System contains three major modules, corresponding to the business functions that were discussed in Section 11.1. These are

- Inquiry module
- Reservation module
- Bill/Payment module

![Architecture of TAU.NET Node.](image)
The modules in the Cell System are linked to the corresponding components in the TAU.NET Node so that their functions can be extended to the distributed environment of TAU.NET. The TAU.NET Node Adaptor components include the Process component and Remote Request component; they talk directly to the Cell System modules. On the other hand, the TAU Node Interface components include the Server objects and Client Proxy object for the Web Services; they talk to the remote TAU.NET Nodes via the Internet.

When an agency initiates a distributed inquiry transaction, the Inquiry module of its Cell System puts a request to the Inquiry Remote Request component in its TAU.NET Node Adaptor. This component translates the user request into the appropriate request of remote Web Services. Then the Proxy object in the TAU Node Interface is instantiated so that the remote Web Services are invoked.

In order to invoke a remote Web Service, a SOAP request message is sent to the server site, where it is received by the Inquiry Server component. Calling the services or methods of the Inquiry Process component in the TAU.NET Node Adapter on the server site, the Inquiry Server component generates the query result and sends it back to the client’s TAU.NET Node. In the TAU.NET Node Adapter on the server site, the Inquiry Process component interacts with the Inquiry module of the Cell System in order to fulfill the requested query.

On the client site, the Inquiry Proxy object receives the query result and passes it to the Inquiry module of the Cell System via the Inquiry Remote Request component.

Distributed Reservation and Bill/Payment transactions can be processed in a similar way. However, databases in the Cell Systems of both client site and server site need to be updated.

Components in TAU.NET Node Adaptor are implemented using ADO.NET, which supports consistent and scalable access to various data sources. Therefore, they need to be customized for different kinds of data sources so that all the discrepancies of the data sources are concealed from the TAU.NET Node Interface. That is, the requests and results are all represented as ADO.NET DataSet objects through which the TAU.NET Node Interface objects interact with the local data.

Furthermore, ADO.NET provides abundant functionality whereby the DataSet objects can be processed in a disconnected fashion. That is, DataSet objects are filled from the database; they are then manipulated in the TAU.NET Node Adaptor, and the database is finally updated if necessary. The Fill and Update operations are carried out in batch mode, but data manipulation can be performed without a database connection. That minimizes the required resource for the interaction between the TAU.NET Node Adaptor and Cell System.
In addition, business rules are also implemented in the components of TAU.NET Node Adapter. The data in DataSet objects is manipulated based on these rules.

The communication between TAU.NET Nodes is based on SOAP, and messages in XML format deliver the requests and results over the HTTP channels. Data of any type can be transferred. More specifically, the query results can be directly delivered as a DataSet object. Consequently, the components in TAU.NET Node Interface are very simple to implement; it simply provides a transparent channel for the TAU.NET Node Adaptors to their exchange DataSets.

11.3.2 Schema for Data Exchange

As we have decided, the DataSet class is used for data exchange. The DataSet can be represented in XSD Schema, which is an XML standard.

11.3.2.1 SCHEMA FOR INQUIRIES ABOUT TAU PRODUCTS AND SERVICES

Generally speaking, there are several fields in the schema that are crucial to a travel product or service:

- Unique Product/Service ID
- Category
- Destination or Location
- Rate and Unit
- Effective and Expiration date

In addition, there may be multiple line items as details and references to original sources.

The schema can be represented as DataSet, XML, and its outline in VS.NET.

See Figure 11-4 for DataSet ERD, and Figure 11-5 for XML outline and XML Schema file TauServiceDataSet.xsd.

The schema file is in XML format as follows.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema id="TauServiceDataSet"
    targetNamespace=
        "http://tempuri.org/TauServiceDataSet.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/TauServiceDataSet.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="TauServiceDataSet"
        msdata:IsDataSet="true">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="TAU_SERVICE">
                    <xsd:complexType>
                        <xsd:sequence>
```
<xsd:element name="SERVICEID"
    type="xsd:string" />
<xsd:element name="BOOKINFOID"
    type="xsd:string"
    minOccurs="0" />
<xsd:element name="CATEGORYCODE"
    type="xsd:string"
    minOccurs="0" />
<xsd:element name="RATE"
    type="xsd:decimal"
    minOccurs="0" />
<xsd:element name="UNIT"
    type="xsd:string"
    minOccurs="0" />
<xsd:element name="EFFECTIVEDATE"
    type="xsd:dateTime"
    minOccurs="0" />
<xsd:element name="THROUGHDATE"
    type="xsd:dateTime"
    minOccurs="0" />
<xsd:element name="LOCATIONCODE"
    type="xsd:string"
    minOccurs="0" />
<xsd:element name="TAU_SERVICE_DETAIL">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="SERVICEID"
                type="xsd:string" />
            <xsd:element name="LineItemOrder"
                type="xsd:int" />
            <xsd:element name="ItemDesc"
                type="xsd:string"
                minOccurs="0" />
            <xsd:element name="ItemUrl"
                type="xsd:string"
                minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:complexType>
</xsd:element>
There is no central server in TAU.NET, nor is there any central controller in the entire Internet. As far as name resolving is concerned, TAU.NET may use Internet standards. That is, member systems store each other’s reference information, and messages are passed between them to synchronize this reference information. To achieve this, the members should share the basic information about their service scope, such as category and destination or location.

The schema should have basic information fields, such as member ID, member name, and service URL. One member may have one or more categories of service to provide and one or more locations or destinations to serve. Figures 11-6 and 11-7 show the schema in two views.
Figure 11-5  Schema outline for TauServiceDataSet in VS.NET.

Figure 11-6  Diagram of a member service information schema.
In XML format, the schema file of TauMemberServiceDataSet has text as follows.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema id="TauMemberServiceDataSet"
    targetNamespace="http://tempuri.org/TauMemberServiceDataSet.xsd"
    elementFormDefault="qualified"
    xmlns="http://tempuri.org/TauMemberServiceDataSet.xsd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="TauMemberServiceDataSet"
        msdata:IsDataSet="true">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="TauMember">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="MemberID" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="MemberName" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="WebServiceUrl" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="WebServiceLocation" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="WebServiceCategory" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="MemberServiceDescription" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="ServiceServerName" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="ServiceServerCategory" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="ServiceServerDescription" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="ServiceServerLocation" type="xsd:string"
                                minOccurs="0" />
                            <xsd:element name="ServiceServerVersion" type="xsd:string"
                                minOccurs="0" />
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Figure 11-7  Document outline of a member service information schema.
11.3.2.3 OTHER SCHEMA FOR DATA EXCHANGE

The reservation transaction information schema is shown in Figure 11-8. The service may have booking information in diverse forms based on the category or vendor. BOOKDATA stores the booking data based on the schema referred by BOOKINFOID.

The schemas for data exchange to serve Billing and Payment transactions are beyond the scope of this case study.

11.3.2.4 CLASSES FOR THE SCHEMAS

When VS.NET is used to construct the schema, the corresponding classes are generated by the tool, creating programs TauServiceDataSet.cs, TauMemberService.cs, and TauBookDataSet.cs. Alternatively, the classes of a schema may be created using command xsd.exe. The following commands generate the classes in TAU namespace:

```markdown
xsd TauServiceDataSet.xsd /dataset /n:TAU
xsd TauMemberService.xsd /dataset /n:TAU
xsd TauBookDataSet.xsd /dataset /n:TAU
```
11.3.3 TAU.NET Node Interface

11.3.3.1 SERVER OBJECTS FOR WEB SERVICES

Having defined the schema for data exchange, we can work on setting the protocol of the Web methods for data exchange. Each member should implement all those Web methods in order to communicate with each other.

First, let’s consider the Web Service for inquiries about products and services. Two criteria, category and destination or location, may be commonly used to inquire about travel services; consequently, two Web methods should be provided, as follows:

```csharp
[WebMethod]
public TauServiceDataSet
    GetServiceDataSetByCategory(
        String categoryCode);

[WebMethod]
public TauServiceDataSet
    GetServiceDataSetByLocation(
        String locationCode);
```

The `categoryCode` string may be a code for information about lodging, airline tickets, or car rentals. The `locationCode` string is about the destination or location the travel agency serves.

Second, let’s consider the Web Service for inquiries about member services. This method is used to get an overview of available services provided by this member. Using this information, other members can build up or refresh their own repositories for member services in TAU.NET.
Finally, let's consider the Web Service for making reservations. Two methods are required here. One is used to get the reservation information for a specific travel service:

```csharp
[WebMethod]
public TAU.TauMemberServiceDataSet
GetMemberServiceInfo ();
```

The other one is for making a reservation on a particular service specified by a given DataSet:

```csharp
[WebMethod]
public string
MakeReservation (TauBookDataSet bookData);
```

All these methods can be gathered in one Web Service file, **TauService.asmx**. Figures 11-9 and 11-10 show the Web Service Description screens on Internet Explorer.
11.3.3.2 PROXY OBJECTS FOR WEB SERVICES

The .NET utility program, wsdl.exe, can generate the Client Proxy for the TAU.NET Web Service.

The following command creates the code TauService.cs, which defines a Client Proxy class, TauService in the TAU.Proxy namespace. The Client Proxy class serves as an access point at the client site to the remote Web Services.

wsdl http://localhost/TAU/TauService.asmx /n:TAU.Proxy

Because all the members implement a Web Service with a standard interface, the same service on various member systems may be invoked by using a general-purpose Client Proxy object. In this case, the URL of the target service needs to be specified as a property of the Client Proxy object. For example, if...
we want to request the Web method `GetServiceDataSetByLocation` in `www.TauMember1.com`, we first need to create a new object:

```csharp
TAU.Proxy.TauService objService;
objService = new TAU.Proxy.TauService();
```

Second, we set its `Url` property value:

```csharp
objService.Url = "http://www.TauMember1.com/TAU/TauService.asmx";
```

Third, we instantiate the `DataSet` to receive the return:

```csharp
TAU.Proxy.TauServiceDataSet objDataSetTemp;
objDataSetTemp = new TAU.Proxy.TauServiceDataSet();
```

Finally, we specify the Web method to fill the `DataSet`:

```csharp
objDataSetTemp =
objService.GetServiceDataSetByLocation( "BOSTON" );
```

### 11.4 TAU.NET Node Adaptor

Now let’s consider the TAU.NET Node Adaptor in which the data is manipulated and the business rule realized.

#### 11.4.1 TAU.NET Node Adaptor to Link Cell System Database

The following code segment is designed to implement the Web method in the Web Service `GetServiceDataSetByLocation`. We need to connect a database in the Cell System to retrieve the travel service records and detail line items according to the criteria on location.

```csharp
[WebMethod]
public TAU.TauServiceDataSet
GetServiceDataSetByLocation(String locationCode)
{
    TAU.TauServiceDataSet returnDataSet
        = new TAU.TauServiceDataSet();
    try
    {
        sqlConnection1.Open();
        sqlDATauServiceLocation.SelectCommand.Parameters
            @LocationCode].Value
            = locationCode;
        sqlDATauServiceDetailLocation.SelectCommand.Parameters
            @LocationCode].Value
            = locationCode;
```
In this code segment, `sqlConnection1`, an instance of `System.Data.SqlClient.SqlConnection`, is used to support connection to the SQL database.

`SqlDATauServiceLocation` is an instance of `SqlDataAdapter` that is built on the following SQL statement:

```
SELECT TAU_MEMBER.NAME AS Name,
       TAU_MEMBER.SERVICEURL AS WebServiceUrl,
       TAU_MEMBER_LOCATION.LOCATIONCODE AS Location
FROM TAU_MEMBER INNER JOIN TAU_MEMBER_LOCATION ON
    TAU_MEMBER.TAUMEMBERID = TAU_MEMBER_LOCATION.TAUMEMBERID
WHERE (TAU_MEMBER_LOCATION.LOCATIONCODE = @LocationCode)
```

`SqlDATauServiceDetailLocation` is an instance of `SqlDataAdapter` that is based on this SQL script:

```
SELECT TAU_SERVICE_DETAIL.SERVICEID AS SERVICEID,
       TAU_SERVICE_DETAIL.LineItemOrder AS LineItemOrder,
       TAU_SERVICE_DETAIL.ItemDesc AS ItemDesc,
       TAU_SERVICE_DETAIL.ItemUrl AS ItemUrl
FROM TAU_SERVICE_DETAIL INNER JOIN TAU_SERVICE ON
    TAU_SERVICE_DETAIL.SERVICEID = TAU_SERVICE.SERVICEID
WHERE (TAU_SERVICE.LOCATIONCODE = @LocationCode)
```

In this simple case, the implementation of the TAU.NET Node Adaptor components is embedded in the implementation of the Web method. Should applications become more complex, the TAU.NET Node Adaptor components should be implemented as separated classes and the Web method should access the data via its instances.
11.4.2 TAU.NET Node Adaptor to Serve a Web Page

In the previous section, we built a TAU.NET Node Adaptor to produce a DataSet containing records about travel service items by retrieving a relational database on a Cell System. Here, we implement another TAU.NET Node Adaptor as a client to request the Web method. This adapter is used from an ASP.NET page, which means the DataSet returning the Web Service would be served to a Web server page and Web browser eventually.

The ASP.NET page is designed as shown in Figure 11-11. We have a master DataGrid and a detail DataGrid. When the Search button is clicked, we load the DataSet for those DataGrid via the Web Service.

The following code is for the button click:

```csharp
private void buttonLoad_Click(object sender, System.EventArgs e)
{
    objTauServiceDataSet = new TAU.TauServiceDataSet();
    this.LoadDataSet(this.locationCode.Text);
    this.masterDataGrid.SelectedIndex = -1;
    this.masterDataGrid.DataBind();
    this.detailDataGrid.Visible = false;
    Application["objTauServiceDataSet"] =
        this.objTauServiceDataSet;
}
```

![Figure 11-11](image-url) Design of the ASP.NET page showing TAU.NET service item information.
In this code, `this.LoadDataSet(this.locationCode.Text)` is designed to load the DataSet via the Web Service.

```csharp
public void LoadDataSet(string locationCode)
{
    TAU.Proxy.TauService objService;
    objService = new TAU.Proxy.TauService();
    objService.Url = "http://localhost/TAU/TauService.asmx";
    TAU.Proxy.TauServiceDataSet objDataSetTemp;
    objDataSetTemp = new TAU.Proxy.TauServiceDataSet();
    try
    {
        objDataSetTemp =
        objService.GetServiceDataSetByLocation(
            locationCode );
    }
    catch (System.Exception eFillDataSet)
    {
        // Add exception handling code here.
        throw eFillDataSet;
    }
    try
    {
        // Merge the records that were just pulled from the data store into the main dataset
        objTauServiceDataSet.Merge(objDataSetTemp);
    }
    catch(System.Exception eLoadMerge)
    {
        // Add exception handling code here
        throw eLoadMerge;
    }
}
```

Once the selection changes on the master DataGrid, we show the corresponding detail by following code logic:

```csharp
private void masterDataGrid_SelectedIndexChanged( object sender, System.EventArgs e)
{
    this.ShowDetailGrid();
}
private void ShowDetailGrid()
{
    if ((this.masterDataGrid.SelectedIndex != -1))
```
{  
    System.Data.DataView parentRows;
    System.Data.DataView childRows;
    System.Data.DataRowView currentParentRow;
    this.objTauServiceDataSet =
    (TAU.TauServiceDataSet)
    (Application["objTauServiceDataSet"]) ;
    parentRows = new DataView();
    parentRows.Table =
    this.objTauServiceDataSet.Tables["TAU_SERVICE"] ;
    currentParentRow = parentRows
    [this.masterDataGrid.SelectedIndex];
    childRows = currentParentRow.CreateChildView(
    "TAU_SERVICE_TAU_SERVICE_DETAIL");
    this.detailDataGrid.DataSource = childRows;
    this.detailDataGrid.DataBind();
    this.detailDataGrid.Visible = true;
} else
{
    this.detailDataGrid.Visible = false;
}

Finally, we get the page shown in Figure 11-12.

Figure 11-12  ASP.NET Web page showing the result retrieved from the Web Service.
Summary

In this chapter, we looked at a Web-based application that uses .NET Framework technology, including ADO.NET, Web Services, and ASP.NET. This application has challenging requirements; it would be a nightmare for developers using the old technology.

As can be seen, our approach is really clean and simple. The distributed functions are implemented by a lightweight extension; no substantial changes are required to each Cell System. .NET Framework and the Internet provide the infrastructure for information exchange. All these characteristics facilitate a new paradigm for development of distributed information processing.

In our discussion, we have just dealt with very basic aspects of systems like this. There are a number of open issues yet to be addressed, including mechanisms for handling distributed transactions, an interesting topic for research and development.