

# Application design of a layered architecture in ASP.NET 2.0

- and a comparison between J2EE and .NET

---

4-week project  
IT-University of Copenhagen (ITU)  
Handed in May 29<sup>th</sup> 2006

**Supervisor:**

Peter Sestoft

**Authors:**

Nils Kasper Emil Oldby  
Mikkel Byrsø Dan

<b>1</b>	<b>INTRODUCTION.....</b>	<b>3</b>
1.1	MOTIVATION .....	3
1.2	PROBLEM DEFINITION AND SCOPE.....	3
1.3	HOW THIS REPORT IS ORGANIZED .....	4
<b>2</b>	<b>APPLICATION DESIGN.....</b>	<b>5</b>
2.1	GENERAL DESIGN PRINCIPLES.....	5
2.2	LAYERED ARCHITECTURE.....	5
2.2.1	<i>Logical and physical tiers</i> .....	5
2.2.2	<i>Vertical division into layers</i> .....	6
2.2.3	<i>One-, two-, three and n-tier applications</i> .....	6
2.2.4	<i>General advantages of layered applications</i> .....	8
2.3	DESIGN PATTERNS .....	8
2.4	THE MODEL-VIEW-CONTROLLER (MVC) IN A WEB CONTEXT .....	9
<b>3</b>	<b>APPLICATION DESIGN IN .NET .....</b>	<b>11</b>
3.1	LAYERED ARCHITECTURE IN .NET .....	11
3.1.1	<i>Two-tier architecture</i> .....	11
3.1.2	<i>Three-tier architecture</i> .....	12
3.1.3	<i>Logical n-tier applications</i> .....	13
3.2	THE PRESENTATION LAYER.....	14
3.2.1	<i>MVC in ASP.NET 2.0</i> .....	14
3.2.2	<i>WebForms</i> .....	14
3.2.3	<i>Data binding</i> .....	14
3.2.4	<i>Master Pages and Themes</i> .....	16
3.2.5	<i>Sitemap</i> .....	16
3.3	THE BUSINESS LOGIC LAYER.....	17
3.3.1	<i>Enterprise Services</i> .....	17
3.4	THE DATA ACCESS LAYER.....	19
3.4.1	<i>ADO.NET 2.0</i> .....	19
3.4.2	<i>Data Access Application Block</i> .....	21
3.4.3	<i>Handling concurrency</i> .....	21
3.5	NEW FEATURES IN ASP.NET 2.0.....	21
<b>4</b>	<b>SAMPLE APPLICATION GOALS .....</b>	<b>22</b>
4.1	THE ARCHITECTURE .....	22
4.2	THE DATABASE .....	22
<b>5</b>	<b>IMPLEMENTATION OF SAMPLE APPLICATION.....</b>	<b>23</b>
5.1	ARCHITECTURE .....	23
5.2	SYSTEM OVERVIEW .....	24
5.3	BUSINESS LOGIC- & DATA ACCESS LAYER .....	25
5.3.1	<i>Communication through the tiers</i> .....	26
5.3.2	<i>The implemented data providers</i> .....	27
5.4	PRESENTATION LAYER.....	30
5.4.1	<i>ASP.NET 2.0 website</i> .....	30
5.4.2	<i>WinClient</i> .....	31
5.5	FURTHER IMPLEMENTATION ISSUES.....	32

<b>6</b>	<b>COMPARISON OF J2EE AND .NET.....</b>	<b>33</b>
6.1	OVERVIEW OF THE PLATFORMS .....	33
6.2	LANGUAGE AND IDE.....	33
6.3	APPLICATION SERVERS AND OPERATING SYSTEMS .....	34
6.4	RUNTIME ENGINES.....	34
6.5	CLASS LIBRARIES .....	34
6.6	TECHNOLOGIES IN THE DIFFERENT LAYERS .....	34
6.6.1	<i>Presentation layer</i> .....	34
6.6.2	<i>Business logic layer</i> .....	37
6.6.3	<i>Data access layer</i> .....	38
6.7	SUMMARY OF J2EE VERSUS .NET.....	39
<b>7</b>	<b>EVALUATION OF ASP.NET 2.0.....</b>	<b>41</b>
<b>8</b>	<b>CONCLUSION.....</b>	<b>42</b>
<b>9</b>	<b>PERSPECTIVES.....</b>	<b>43</b>
<b>10</b>	<b>BIBLIOGRAPHY .....</b>	<b>44</b>
<b>11</b>	<b>APPENDICES .....</b>	<b>48</b>

## 1 Introduction

This report has been written in a four-week project carried out as a part of the “C# and .NET project cluster May 2006” at the IT-University of Copenhagen. Further information about the project cluster is available at the project cluster’s homepage[33]. The project was supervised by Peter Sestoft, who we thank for taking the time to supervise and for the valuable comments and feedback in the process of developing the sample application and writing this report.

### 1.1 Motivation

Our motivation for doing this project follows the main goal of the project cluster, namely to get a chance to familiarize ourselves with the C# programming language and the .NET class libraries, including the version 2.0 features released with Visual Studio 2005.

Furthermore we are interested in the design of object-oriented distributed applications and have some knowledge of the architecture and implementation of such applications in the J2EE framework. In general we are eager to learn about systems and architectures that are easy maintainable, scalable, flexible and that can exchange information with other systems and clients on other platforms in an internet context. Therefore this project cluster seems like the perfect combination for us, since it will allow us to look into the design of the type of applications just mentioned using the C# programming language and ASP.NET 2.0.

### 1.2 Problem definition and scope

The purpose of this project is to discuss architecture and application design of object-oriented distributed applications in general and in relation to the .NET platform. In addition we will try to elaborate on the differences between J2EE and .NET in relation to these subjects.

We will try to challenge the typical 2-tier application design that is encouraged in .NET and Visual Studio, where it’s normally to make use of direct data binding from the user interface to an underlying data base. This is specifically possible because of the drag ‘n’ drop facilities in Visual Studio which makes development fast and easy but might have negative effects in relation to maintenance etc.

The project will discuss n-tier architecture from a theoretical perspective and examine how and what differences there are between n-tier application design using J2EE and .NET. The .NET design should not only be examined on a theoretical level but also be implemented. The implementation of a small sample application should illustrate blueprints of a .NET-based layered application architecture.

In the project an effort will also be placed on learning and using new features in ASP.NET 2.0 and Visual Studio 2005. Due to the fact that this project is written in a C# project cluster, the sample application will only be developed using C# / ASP.NET 2.0 and not using J2EE.

### **1.3 How this report is organized**

This report could be viewed as consisting of three logical parts. The parts being, application design in general and in .NET, a sample application designed using .NET and finally a comparison between J2EE and .NET.

Each main chapter of the report will always include a short introduction. The purpose of these introductions is to briefly introduce the coming information and to keep the reader aware of where in the context the information fits.

#### **Citations**

The citation style used in this report is “bibliography style plain”, which means that we use square bracket parenthesis and a number when referring to a resource e.g. [1]. Our bibliography is as usual placed in the end of the report. It is arranged by the type of source e.g. books or online articles. Within each type the references are sorted alphabetically.

#### **Computer code**

Computer code is marked using a monospace font. Computer code looks like the following:

```
localhost.Service ws = new localhost.Service();
```

## 2 Application Design

The purpose of this section is to examine general architectural principles that exist in application design. Further it should be determined what advantages dividing an application into layers and tiers have. Finally different layered architectures will be described and their advantages examined. The overall purpose of this section is to get a general understanding of layered application design.

### 2.1 General design principles

In general, when designing applications it is important to focus on the system design in order to ease maintainability, interoperability, scalability, further development etc. One thing that is especially important is low coupling, hence it is important to “reduce the impact of change” [5, p. 285]. Briefly, coupling is a measure of how strong one element is connected to other elements, if it has knowledge of or depends on them. Elements in this context can be of all levels of granularity from classes in an object-oriented language to components to large sub-systems or software modules. If there is a high coupling or dependency between elements, then a change in the depended-upon element will cause a change in the dependant and this is often not desirable.

Another important matter is high cohesion, “how to keep objects focused, understandable, and manageable, and as a side-effect, support low coupling” [5, p. 315]. The term cohesion is a measure of how strongly and focused the responsibilities of an element are. Thus, an element with highly related responsibilities that does not do a lot of different tasks has high cohesion. High cohesion makes the system easier to comprehend, easier to reuse and maintain which in general is desirable.

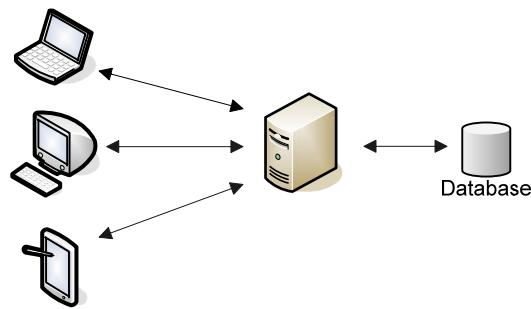
### 2.2 Layered architecture

When the different elements in a system are organized systematically we talk about the system architecture. The architecture is the large-scale division of a system into layers or tiers, each with responsibility for a major part of the system and with as little direct influence on other layers as possible. This definition was derived from [11].

#### 2.2.1 Logical and physical tiers

A tier can both refer to a logical and a physical entity. The term tier was originally introduced as meaning a logical layer [5, p. 207], but now has both a logical and a physical meaning because of the widespread client/server model and distributed applications.

The physical meaning of the tier term refers to the fact that different parts of a system can be placed on different hardware, e.g. client/server. In this case you can for instance place the user interface on the client and the business logic and data on the server. This is called a two-tiered architecture. In this situation you work with different communication protocols to exchange information between the physically divided tiers. A division into physical tiers is illustrated in Figure 1.



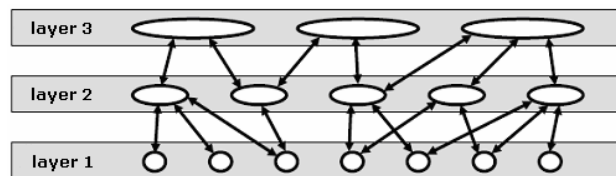
**Figure 1 - Physical tiers**

The logical meaning of the layer/tier term refers to a conceptually distinct aspect of a system e.g. separating the user interface from the business logic. The user interface can be implemented with different technologies and have different appearance, while the business logic remains the same. These two aspects are thus logically distinct and can be said to represent two distinct layers or tiers. Only tiers right beside each other communicate directly and they perform different tasks in the system as the example with the user interface and business logic.

### 2.2.2 Vertical division into layers

Separating the system into vertical layers means that the system is divided into layers or slices on top of each other. The higher layers call upon services from lower layers and not vice versa. It is also often ensured that all communication between layers is placed in distinct interaction classes and that each layer has a well-known interface so that it is always clear where and how the layers communicate.

In a vertical layer split-up different layers often provide the same functionality, just at a different abstraction level. This is opposite of the logical layer/tier, where the different layers provided different functionality. One example of a vertical division is the TCP protocol, which is the base for both the FTP- and HTTP protocol. In high-level layers you can have high-level functions e.g. requesting a data transfer process while the low-level layers can perform very specialized and fine-grained functions detailing how the data transfer should be accomplished. One layer thus accomplishes a function by calling several other more fine-grained functions in the layer below. This is illustrated in Figure 2.



**Figure 2 - Vertical division into layers [29]**

### 2.2.3 One-, two-, three and n-tier applications

There are a number of ways a system can be split into physical and logical tiers. How many physical and logical tiers an application should be divided into depends on the use and

requirements of the system. In the following we will describe the typical difference between one, two, three and n-tier applications. The following section is based on [16].

### One-tier applications

One-tier applications are normally simple isolated programs that don't have network access. Therefore it's not necessary to take network communication and the risk of network failure into consideration. Neither do these programs need to think of synchronization of data, since all the data exists within the same application. When separating tiers physically the application will be slower due to the inevitable fact that communication over network will result in a loss of performance, therefore one-tier applications have high performance.

### Two-tier applications

A two-tier application, in contrast to the one-tier application just described, does not combine all functions into a single process but separate different functions. A good example of this kind of application is a chat application. This kind of application contains two separated tiers, a client and a server. The client has the responsibility of capturing user input and displaying the actual messages. The server is responsible of the communication between the people that uses the chat client.

### Three-tier applications

A three-tier application, adds another tier to the previous mentioned chat application, this could be in the form of a database. One could think of a three-tier application as a dynamic web application, which has a user interface, business logic and a database each placed in different tiers, this is illustrated in Figure 3. As mentioned in the previous section a two-tier architecture separates the user interface from the business logic, in the same way the three-tier architecture separates the database from the business logic.

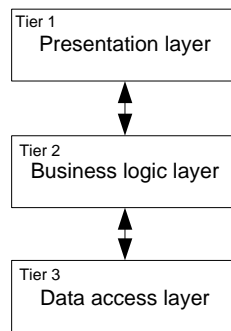


Figure 3 - Three-tier application

### Logical n-tier applications

As was mentioned in 2.2.1, logical and physical tiers are not the same. A logical n-tier application is an application where all logical parts are separated into discrete classes. In a typical business application, this generally involves a presentation layer, business logic layer and a data access layer. This was illustrated in the previous section, see Figure 3. This

separation also makes the application easier to maintain. The advantages of this architecture are that all business rules are centralized which make them easy to create, use and re-use. The data access is also centralized, which has the same advantage as the centralization of the business rules. Centralizing the data access routines are also good when it comes to maintenance since changes only has to be implemented at one location. The effect of using a middle tier can be seen in Figure 4.

There are really not that many disadvantages of this kind of architecture, however, it takes a bit longer to get up and running since several separate components has to be developed, which also might make the code a bit more complicated to grasp for less experienced programmers. This section was based on [19] and [20].

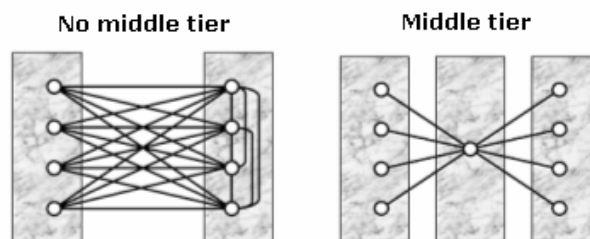


Figure 4 - Middle tier [29]

### 2.2.4 General advantages of layered applications

There are several advantages of designing applications that are split up into different tiers or layers. Most importantly this form of architecture helps to enforce the principles of high cohesion within system components and low coupling between different components. A system built on these principles will be more robust, easier to adjust, easier to maintain and understand and it allows different software components to be developed in parallel.

The key point is that a system should be split into different smaller parts that are as cohesive and self-governing as possible. Each part has distinct responsibilities and interacts with other parts of the system to accomplish its tasks and responsibilities. This also ensures that the systems can cooperate across different platforms and communication protocols and makes it easier to reuse existing solutions to problems often encountered.

All in all these advantages are desirable because they work in the direction of low coupling and high cohesion. The hard part is, however, to implement this in practice and to know when and how it should be implemented. What objects should have responsibility for the different tasks and how do they interact. This is where the design patterns play a vital role.

## 2.3 Design patterns

A very essential strength of object-oriented software development is the ability to use software classes as more or less independent building blocks that can be reused when appropriate. A pattern is basically a structured description or design of a problem and its solution that can be applied in different situations. If one group of software developers has found a good solution to a problem, there is no reason for another group to reinvent this solution when they encounter a similar problem. Many resources can be saved if it is possible to reuse a design pattern that is known to solve the problem.

It is important to emphasize that patterns are not classes – that is, patterns are not software code, but conceptual designs on how to solve a problem. Patterns can be low-level, concerning a quite detailed solution to a specific problem, or high-level, containing guidelines on how to structure large parts of the system.

Patterns are also a mean of communication between developers. Instead of referring to specific implementation details developers might speak in terms of design patterns.

Throughout this report we will introduce different design patterns that are used in J2EE and .NET. In our sample application we also make use of the Factory Method [9], Façade [9] and Data Transfer Object [8] pattern. In the following we will start by introducing the MVC-pattern in a web context which is closely related to the earlier described logical division between the user interface and the business logic.

## 2.4 The Model-View-Controller (MVC) in a web context

The following section is based on [18]. The MVC pattern is widely discussed and exists in different contexts. The MVC in a web context is segmented into a Model, a View and a controller. The Model is the gateway to the business logic. It's the model that is responsible for the functionality of the application. The View is the presentation layer. It's the View that provides the user with the available actions and the information needed to use them. The Controller is the bridge between the Model and the View, interpreting requests from the user and commanding the Model and/or View to change as appropriate. The general flow through the MVC is shown in Figure 5.

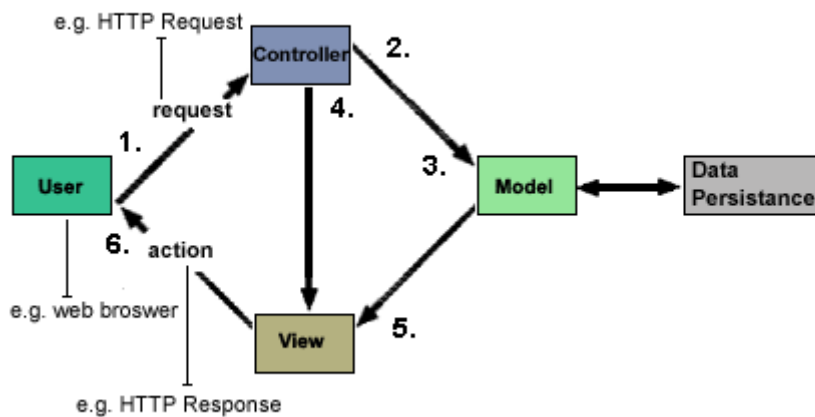


Figure 5 – MVC [18]

In the web context what activates the MVC is when events are generated in the presentation layer. A typically scenario is when a page is requested. Then the MVC handles it in the following way:

- The Event that was generated is handled by the Controller.
- The Controller reads and interprets the event. What actually happens is that the Controller maps the generated Event with the appropriate handler in the Model.
- The Model now performs the action needed. When the Model is finished the result is returned to the Controller.

- Then the Controller determines which View that should be shown. Thereafter everything is forwarded to this responsible View.
- When the responsible View loads it may retrieve data from the Model through a data interface, it may however not execute actions on the Model. The data is now rendered by the View, showing it to the user.

The separation of components created by the MVC pattern helps eliminate UI dependencies, as each component is responsible for a separate set of logically grouped tasks. The separation also reduces code duplication and enhances the maintainability of the application. Further it also makes handling data easier, whether adding new data sources or changing data presentation, because business logic is kept separate from data. Introducing new clients will also be easier since it is not necessary to change the business logic with the addition of each new type of client [14]. Another strength due to the separation of the application into responsibility based parts, is that it's more easy to have different development teams working on what they know best without interfering with each other.

### 3 Application design in .NET

The purpose of this chapter is to examine how n-tier applications can be designed in .NET. A description of the different possible layered architectures and their advantages and disadvantages will be presented. Next an examination of the features and technologies that might be helpful when implementing the sample application will be described in the layer they belong to. Briefly, an overview of COM+ in .NET will be provided in the business logic layer.

#### 3.1 Layered architecture in .NET

The architectural part in the coming sections is mainly based on [19] and [20].

##### 3.1.1 Two-tier architecture

It's normal to use a two-tiered architecture when programming in ASP.NET using Visual Studio. The reason is that when programming using the drag 'n' drop functionality available a direct data binding is often established between the ADO.NET data and the control the data is bound to.

Direct data binding is not a problem; however it's important to understand when it's appropriate to use it and not to use it. In general direct data binding (two-tier architecture) should not be used when developing to an application where changes and maintenance are important, hence development to an enterprise environment should not in general involve direct data binding.

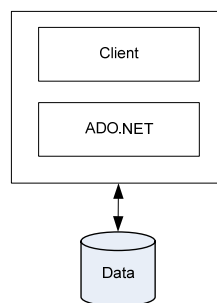


Figure 6 - Two-tier architecture [19]

##### Advantages

However, direct data binding between ADO.NET and the controls available do have some advantages. Development is extremely fast. Features such as sorting, paging and editing/updating/adding/deleting are functionality that is directly available in the control. It is also easier to get an overview of the code, since all code is gathered in one place.

### Disadvantages

The simplicity brought to the application as a result of the direct data binding was mentioned as an advantage in the previous section. However this simplicity does come at a high price, there are several disadvantages.

The direct data binding forces all the business rules to be placed in the front-end of the application. In an application where business rules changes, this demands a lot of maintenance since all clients has to be programmatically updated manually to adopt the new business rules. Hence maintainability is a potential problem, which is generally the case when code duplication exists.

If the data sources changes, e.g. from XML to a database or vice versa, more updating is necessary to be able to load the data into the application. The same dilemma will occur if the database fieldname is changed, then all involved ADO.NET data bindings has to be altered. As described in this section, the disadvantages are mainly due to the lack of a binding gateway, which result in more work when it comes to maintainability since changes needs to be implemented everywhere.

### 3.1.2 Three-tier architecture

Building a three-tier architecture in ASP.NET could be accomplished in several ways, e.g. through .NET Remoting or through XML Web Services. In the following only the web service version is examined, however, the basic advantages and disadvantages are the same.

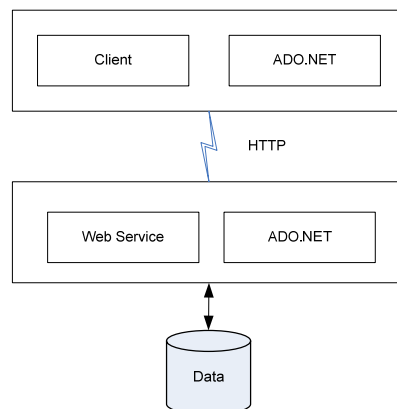


Figure 7 - Three-tier architecture using XML Web Service [19]

#### Using an XML Web Service

In .NET there are two ways to simply extend the previous described two-tier architecture into a three-tiered one. A possibility is to include a web service to separate the database access into a component. The application would be the exact same as the mentioned in the previous section 3.1.1 however, it would be possible to send and receive data to the application using HTTP. The advantages and disadvantages of this type of three-tier architecture are different from the two-tier architecture just described.

### Advantages

The application is now distributed in the sense that users can access it from outside as long as they are connected to the internet. The client maintenance should be minimized since all the ADO.NET connection configuration etc. is placed at the web service. The web service works as a kind of communication gateway which means that all SQL is placed in one location, removing it from the client side and enforcing that the database layer is updated at one place only.

### Disadvantages

The business rules are still placed in the front end, as described in the two-tier architecture in section 3.1.1. The extra layer and the HTTP communication make the application as whole work a bit slower. Without internet the application does not function.

### 3.1.3 Logical n-tier applications

In section 2.2.3 the logical n-tier architecture was examined. To implement this using .NET one would most often create the front-end user interface. This would be done using either WinForms or WebForms (ASP.NET). The business rules would be implemented as a separate Class Library project as would the data layer component. The communication between the client and the data would be done by using web service. It should be mentioned, as can be seen from Figure 8 the business rules are placed on the client side, which minimizes the network traffic. However, this might not be such a good idea if the business rules changes frequently, since an update would be necessary. But .NET is able to copy over new DLLs without registration; therefore this is not a major problem. There are possibilities like having only presentation at the client and the business rules at the server. We will examine this in our sample application.

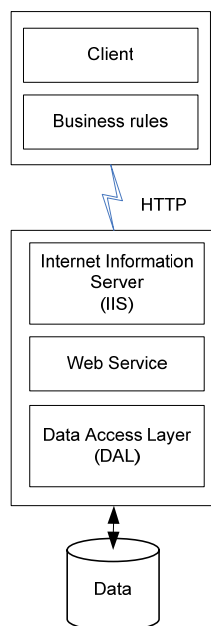


Figure 8 - N-tier application using an XML Web Service [19]

## 3.2 The presentation layer

This purpose of this section is to examine which possibilities .NET offers to get the data from the data source to the user interface. Further it should be examined what kind of controls there exists to show the data and when to use what control. Due to the wide scope of the project we have decided not to include WinForms but only WebForms in this chapter.

### 3.2.1 MVC in ASP.NET 2.0

ASP.NET does not implement the MVC in the sense that was described in section 2.4. In ASP.NET MVC is implemented through the use of the “Page Controller pattern” [8]. This pattern, applies the controller at the level of individual pages and not on an application level. If an application level controller is needed, this can be created through the `HttpHandler` since it intercepts the request before the ASP.NET page does. ASP.NET divides each page into two parts: a View that contains the various controls, and the code-behind, which contains event handlers for every event that the ASP.NET Controller can raise as it processes the page. Developers don't need to interact with the actual control mechanism; they simply write event handlers to wire up the model and view components. What actually happens is that the runtime intercepts the page requests and invokes the requested actions on the model, and determines the correct view to use for the resulting page. All of this is out of site for the programmer. This part was based on [18].

### 3.2.2 WebForms

WebForms are what ASP.NET web sites are developed around. ASP.NET WebForms have the extension “.aspx” and are the standard way to provide web functionality to clients. A request to an “.aspx” resource will execute the file (the .aspx resource and the code behind file). The code-behind files helps to separate the logic behind a page from the visual part of the page. [6]. This means that the html markup is separated physically from the C# code.

### 3.2.3 Data binding

The following section is mainly based on [22]. In section 2.2, where the application architecture was discussed it was argued that building a n-tier application involves several layers. Most often a presentation-, business logic- and a data access layer. Communication only takes place between neighbouring layers. Therefore, using the **SqlDataSource** is not always an option, especially not when building n-tiered application, where the user interface communicates with the business logic layer and not directly with the data source.

The `SqlDataSource` control is a very easy to use and fast to configure. The control refers SQL statements or stored procedures directly from the user interface layer, which is not good design. However, if it's necessary to display database content on a single isolated page, it might be overkill to go through the business logic layer (since this layer, in this case would not serve its purpose). In these situations it might be okay to have the presentation layer (“.aspx” pages) call the data access layer directly, hence using the `SqlDataSource`. Although this is a violation of the guidance provided by the layering pattern, according to [13].

In the case when one is building an n-tier web application it's necessary to have a control that is able to bind to components from the business logic layer. This is exactly what the **ObjectDataSource** does. The `ObjectDataSource` control enables you to bind controls

used in the user interface to render data (e.g. the GridView) to a middle-tier component. The ObjectDataSource is a web site control. In this situation the data is available through the ObjectDataSource. Now a decision about which control to use to show the data has to be taken. There exist about five controls that show the data, each is shortly described in the following. The description is based on [24].

- GridView
- DetailsView
- DataList
- FormView
- Repeater

The **GridView control** displays data as a table and provides the capability to sort columns, page through data, and edit or delete a single record. Due to all the including functionality the control is heavy to use.

The **DetailsView control** renders a single record at a time as a table and provides the capability to page through multiple records, as well as to insert, update, and delete records. A selected record in e.g. a GridView control could determines the record displayed by the DetailsView control.

The **FormView control** renders a single record at a time from a data source and provides the capability to page through multiple records, as well as to insert, update, and delete records, similar to the DetailsView control. However, the difference between the FormView and the DetailsView controls is that the DetailsView control uses a table-based layout where each field of the data record is displayed as a row in the control. In contrast, the FormView control does not specify a pre-defined layout for displaying a record. Instead, you create templates that contain controls to display individual fields from the record. The template contains the formatting, controls, and binding expressions used to lay out the form.

While the **Repeater control** renders a read-only list from a set of records the **DataList control** renders data as table and enables you to display data records in different layouts. The DataList control differs from the Repeater control in that the DataList control explicitly places items in an html table, where as the Repeater control does not.

As has been described in this section different kind of data source controls exists. We described the ObjectDataSource and SqlDataSource. These controls work together with the data-bound controls just described. This is done by binding them to the data source control. When a data-bound control is bound to a data source control, little or no additional code is required for data operations, because the data-bound control can automatically take advantage of the data services provided by the data source control [21]. The overview of what has just been described is shown in Figure 9.

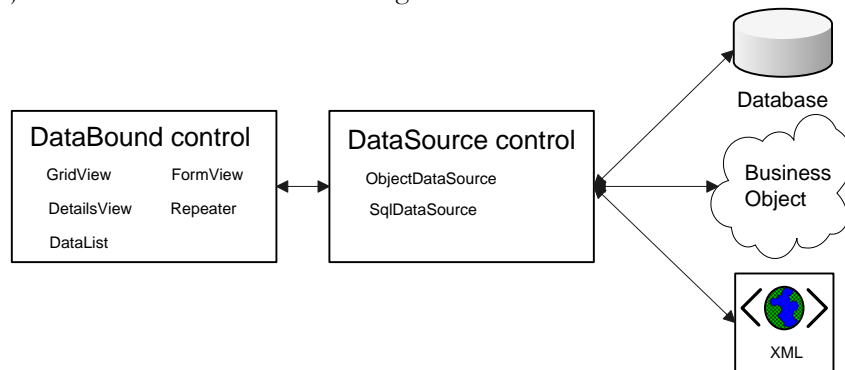


Figure 9 – DataBinding (freely transformed from [21])

When the data is shown in the DataBound control, it is actually presented to the user. Therefore the next section will examine what features ASP.NET 2.0 offers to make the data and the data controls look and feel in a consistent way, throughout a set of web pages.

### 3.2.4 Master Pages and Themes

ASP.NET Master Pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behaviour that you want for all of the pages (or a group of pages) in your application. You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page. However, a page does not necessarily have to use the content of the Master Page, each page are able to override the default content of the Master Page showing other content. How Master Pages works is shown in Figure 10. [23]

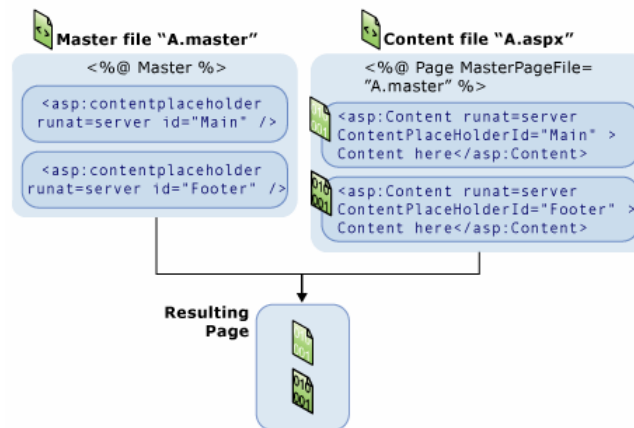


Figure 10 - Master Pages[23]

Themes enable you to customize server controls with a consistent look. Themes are used together with Master Pages. “A theme is the union of all visual styles for all customizable elements in the pages - a sort of super CSS file”[7]. A theme makes it possible to provide different looking web applications with the exact same source code. A Theme can be programmatically switched to e.g. fulfill a users choice of look and feel on the page.

### 3.2.5 Sitemap

ASP.NET 2.0 offers a new feature to ease development and maintenance of navigation on web sites. The new control is the **SiteMapDataSource** control. Through this control it's possible to bind a number of menus e.g. the TreeView control. The control uses a site map provider to read the site map. These nodes are then passed to the TreeView (or another menu) which arranges the HTML rendering. The only problem is that the XmlSiteMapProvider is the only site map provider included in ASP.NET 2.0. The result is that all site maps must be stored in XML files and not in databases or any other persistent media. However it is possible to develop a new provider that reads the information from a database. This was based on [3].

### 3.3 The business logic layer

In this section we will examine how useful features that are often needed in enterprise applications works in the .NET framework. Features such as thread and transaction management, queued components and security will be examined.

#### 3.3.1 Enterprise Services

Support for Component Services has been extended into the .NET Framework. In this section it's shortly examined how some useful components are used in .NET. This section is based on [13].

It should be mentioned that the functionality of COM+ is exposed in the .NET Framework via the assembly named System.EnterpriseServices, which is part of the Framework class library. Before using it, it has to be referenced from inside the projects. This will make the ServicedComponent class available to all class library projects.

#### Just-In-Time Activation (JITA)

This feature has the effect that an instance of an object only survives for the span of a single method call, even if a reference to the instance is hold. The main goal of this feature is to optimize the efficiency of stateless components. How this is used can be seen in the code example in the next section.

#### Object Pooling

An object pool is a number of active instances of a type that COM+ maintains in memory and then dynamically allocates as clients request instances of the type. A pool of objects increase the scalability of an application by avoiding expensive object instantiation and destruction which is more effective than using an instance per client. For this to work, the object must be stateless. Pooling could be used in combination with JITA. This is indicated below:

```
[ObjectPooling(1, 1000)] //Minimum and maximum pool size
[JustInTimeActivation(true)]
public class MyPool : ServicedComponent
{
}
```

**Listing 1 - ObjectPooling**

If the instance is to be returned to the pool when the method call is completed, `Context.Util.DeactivationOnReturn = true;` should be added before the return statement of the method.

#### Transactions

COM+ can also manage transactions. Transactions can span methods, components and databases. The problem with COM+ transactions is that they rely on the Distributed Transaction Coordinator (DTC) which consumes a lot of resources. It has some advantages, but if the transactional work does not occur across relational database systems but in a single database, other alternatives might be better. For a single database one could use T-SQL Transactions or ADO.NET connection-based transactions. Transactions in .NET 2.0 are made available in the System.Transaction namespace of the Framework class library. The following method does its work in a transaction:

```
public bool UpdateQuantity(int id, int q)
{
    using (TransactionScope tx = new
        TransactionScope(TransactionScopeOption.Required))
    {
        tx.Complete();
    }
    return true;
}
```

**Listing 2 - Transaction**

#### Queued Components

Configuring a component as queued component gives asynchronous method invocation without it being necessary to fix the underlying details of preparing MSMQ messages and placing them in queues. The details of creating a message, putting it in a queue and processing it on the receiver are managed by the hosting environment, hence the Microsoft Transaction Server. When a component is configured as queued, a call to the message prompts COM+ to prepare a MSMQ message and place it in a private queue. Another COM+ process acts as a listener to this queue, pulls the message out when it arrives and invokes the methods described in the message. It is beyond the scope of this project to examine queued components any further.

#### Role-Based Security

COM+ enables roles at the component, interface or method level. Role creation is done with an assembly-level attribute. The example in Listing 3 would force that a caller to this class are in the Professor role.

```
[assembly: SecurityRole("Professor")]
```

This would result in the corresponding COM+ roles that are created. Now the SecurityRole attribute can be used e.g.:

```
[SecurityRole("Professor")]
public class Student
{
}
```

**Listing 3 – Security**

This section has briefly provided information about how complex features could be used in .NET at the business logic layer level. Since the business logic layer communicates with the data access logic layer, next section will describe this layer.

### 3.4 The data access layer

When building an application it has to be considered how the database communication is handled. The mapping could primarily be done in two different ways, either through an Object/Relational (O/R) mapping tool such as NHibernate or manually. In the following section a discussion about the data layer in .NET follows. This section focuses on ADO.NET, not on using an O/R mapping tool.

#### 3.4.1 ADO.NET 2.0

“ADO.NET is the primary relational data access model for Microsoft .NET-based applications. It may be used to access data sources for which there is a specific .NET Provider, or, via a .NET Bridge Provider, for which there is a specific OLE DB Provider, ODBC Driver, or JDBC Driver. ADO.NET is sometimes considered an evolution of ADO technology, but it is important to note that some major changes were made between the two.”[34]. The rest of this section is primarily based on [15].

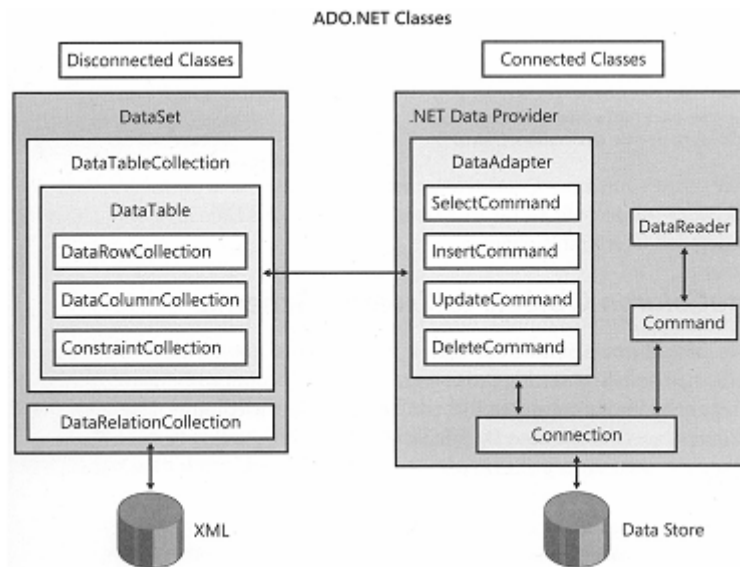


Figure 11 - ADO.NET classes [10]

#### Minimized Open Connections

ADO.NET minimizes the time with open database connections since a connection is only established to retrieve and update records. The records received are copied into a DataSet where after the connection is closed. The DataSets are designed to store data in disconnected and distributed data environments [15]. The ADO.NET classes can be seen above in Figure 11.

#### **.NET Data Providers**

The .NET data providers enable an application to connect to a data source, execute commands and receive results. The .NET Data Provider consists of four components. These components with explanation are as follows:

- Connection
- Command
- DataReader
- DataAdapter

The **Connection** object enables you to establish the connection to the data source. The **Command** object enables you to execute SQL commands (Read, Insert, Update, and Delete) and retrieve results from a database. The **DataReaders** provide forward-only, read-only, connected access to data sources, no natively sorting functionality and they don't support data manipulation however they are very fast. In remoting there is a risk that the client application holds the database connection open for lengthy periods. This is the case since once a DataReader has been opened, the connection it is using cannot be used by any other object and the connection to the data source is maintained until the close method is called. The **DataAdapter** object enables a database and a DataSet to communicate with each other; hence you use the adapter to transfer data between a DataSet and a data source.

#### **DataSets**

This section about DataSets is based on [1]. A DataSet is an in memory database specialized to hold any kind for tabular data. DataSet allow relations to be defined between pairs of tables. DataSets are designed to handle data from different sources, such as the file system, memory, a database etc. DataSets have native support for XML serialization and optimistic concurrency. DataSets are also very easy to use when working with either Web Forms or Windows Forms. The previous described DataAdapter is used to bridge the DataSet with the data source. By calling methods on a data adapter, you can execute database commands using the contents of the DataSet.

As was mentioned before, DataSets contains tabular data, they do however not ad any specific additional behaviour. In situations where objects need behaviour datasets might not be suitable. The fact that DataSets are not business entities but merely holds data in tabular form makes the programming more relational than object oriented.

#### **Strongly Typed DataSets**

The difference between a typed DataSet and a normal DataSet is that the typed version provides strongly typed methods, events and properties. The typed DataSet derives from a DataSet and therefore inherits all methods, events and properties of a DataSet. Beside that the strongly typed DataSet gives type safety of values at compile time. It also provides IntelliSense in Visual Studio and making the work simple due to the fact that tables and columns are accessed by name. It makes the database programming more object-oriented-like.

Creating a strongly typed dataset in Visual Studio .NET is easy, basically the only thing that has to be done is to drag a database table from the Server Explorer onto a DataSet designer (XSD file). This XSD file then stores the XML that defines the schema for the strongly typed DataSet. This section was based on [2].

### 3.4.2 Data Access Application Block

The patterns & practices Enterprise Library is a library of application blocks designed to assist developers with common enterprise development challenges. Microsoft has developed a Data Access Application Block which simplifies development tasks that implement common data access functionality. Applications can use this application block in a variety of situations, such as reading data for display, passing data through application layers, and submitting changed data back to the database system. [32]

### 3.4.3 Handling concurrency

In principle concurrency can be handled either pessimistic or optimistic. If a user reads data with the intention of updating it, under pessimistic concurrency a lock is established on the data source (row) during the whole operation. The data source (row) is first available after the operation. When using optimistic concurrency, a user does not lock the data source (row) when reading it, however it has to be determined before the update takes place whether the data source (row) has been changed since it was read. In the following we explain how optimistic concurrency can be implemented. This section is based on [5].

#### Distributed time stamps

One solution is to add a timestamp column to each data row. If the timestamp is returned with each query, it's fairly simple to verify whether the time stamp value in the database is the same as the original time stamp, before updating. Optimistic concurrency violation would occur if the timestamps are not the same.

#### Maintain original values

Another solution is to maintain a copy of the original values. Before updating it's only necessary to verify that the original values match with the ones stored in the database. It should also be mentioned that maintaining the original values gives the possibility to check each value instead of the whole row, which is the case when using distributed time stamps.

#### DataSets

DataSets has native support for optimistic concurrency. This is because the DataSets maintain a copy of the original values.

## 3.5 New features in ASP.NET 2.0

Throughout this section we have highlighted some of the new features of ASP.NET 2.0 that we intend to use in the sample application. We have talked about Master Pages, Data Source Controls, Data Controls and Data-Driven Site Navigation. We have also indirectly talked about Codebehind 2.0 and some of the other new controls. However there are still quite a few new features that due to the scope of this project will not be examined. These features can be seen in Appendix 1.

## 4 Sample application goals

In order to fulfill the goals of the project, namely to get familiarized with C#, ASP.NET 2.0, Visual Studio and to investigate how to design and implement a layered architecture, we will develop a small sample application. The sample application is based on a general business case, namely a company that sells products through the internet. Therefore the company has an existing database containing customers, orders, products etc.

The sample application should allow to select a customer and to view all the orders for that given customer. For each order it should be possible to see the orderlines and the actual product. It should be possible to view the above mentioned information as a WebForm and WinForm.

Since this project focuses on the architecture and that the time available for the project is limited, the sample application chosen is very simple and no energy will be placed on the graphical layout.

### 4.1 The architecture

As was mentioned in our problem definition the purpose of this project is partly to challenge the typical 2-tier application design that is encouraged in .NET. Therefore our implementation of the sample application should use the previous explained n-tier architecture in .NET. In our case this should be a logical three-tier architecture with a presentation layer, business logic layer and data access layer.

Our application is simple; however, it's not difficult to imagine scenarios where this type of application needs "Enterprise Services" as described previous. Therefore the architecture should also enable these to be easily configured.

This architecture should provide us with a middle tier; the business logic layer that makes the presentation layer independent of the data access layer. We should furthermore ensure loose coupling between the business logic layer and the data access layer in order to be able to change aspects of the system without too much affect on the other.

The presentation layer should use the right controls and navigation functionality provided by ASP.NET 2.0.

The implementation of the sample application should illustrate blueprints of a .NET-based layered application architecture.

### 4.2 The Database

The database used to illustrate this company is the Northwind sample database from Microsoft. The database design is available in Appendix 2. Since we previously discussed strongly typed DataSets, we have provided the "database diagram" as a DataSet (.xsd file), which could be used for programming the sample application using strongly typed DataSets.

## 5 Implementation of sample application

Now that we have examined how .NET-based layered applications could be designed and have established goals for the sample application we will present our implementation. As described in the problem definition, our intention was to challenge the typical two-tier architecture of ASP.NET. However, as it was described in section 3.1.1 there are scenarios where two-tier architecture is preferred due to the scope and requirements of the application. In relation to this it should be said that our architecture might seem a bit over-engineered for this simple application. But it provides the flexibility to plug in new features and establishes a loose coupling between the different layers. Inspiration for our implementation has been found in [30] and [31].

In the following we will start by introducing the overall architecture of the application and how the different layers is interconnected and communicates. After that a more detailed description of the system in its whole and the contents of each tier follow.

### 5.1 Architecture

We have designed the sample application as a logical three-tier architecture with a presentation layer (UI), business logic layer (BLL) and data access layer (DAL). The UI is implemented both as a WinForm and as a WebForm (ASP.NET 2.0) application to show that the architecture is able to service different clients. Because the WinForm is placed on a different physical tier than the rest of the application, the BLL is exposed to the UIs through an XML Web Service. The sample application architecture is shown in Figure 12.

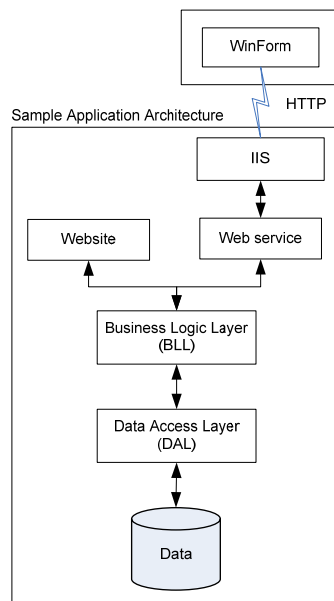
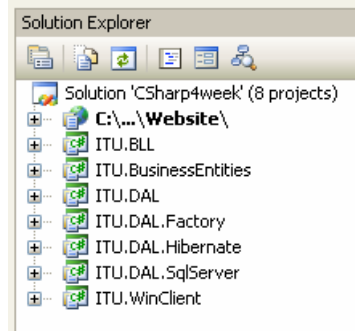


Figure 12 - Sample application architecture

The website can either use this web service or access the BLL component directly. Since the website is hosted in the same environment it is unnecessary to go through the web service so the BLL component is accessed directly. However, the web service makes it possible to host the website on another physical tier than the BLL and DAL. Beside for the logical separation

into tiers we compiled different parts into C# class libraries. The whole solution can be seen in Figure 13.



**Figure 13 - Sample application Solution Explorer**

For instance the BLL is implemented as a C# class library that is compiled into one assembly. It is the middle tier and acts as a Façade[9] to the rest of the system. It provides the methods to handle creation and retrieval of customers, orders etc. The methods in the BLL doesn't contain that much "business logic" but with this design it is possible to implement more intelligent methods that for instance perform different enterprise services such as distributed transactions. However, we haven't implemented them due to the time limit of this project. Enterprise Services was discussed and small implementation examples where provided in section 3.3.1.

The BLL retrieves persistent data only through the DAL. The DAL encapsulates all communication with the database and offers different data providers. We have implemented a Microsoft SQL Server database and a NHibernate data provider.

To pass information between the tiers we have created Business Entities, also known as a "Data Transfer Object" [8]. This enables the use of more coarse-grained operations. The Business Entities such as Customer is light-weight objects that simply represent data from the database. But since it is plain C# objects we can return a Collection of the objects and send them all the way to the UI that can print them in different DataBound controls, e.g. a GridView.

## 5.2 System overview

To give a better overview of how the application is structured and how the different tiers / layers communicate we have provided a class diagram of the main aspects of the application that can be seen in Figure 14. This is a simplified diagram that only deals with customers. There is for instance only an ICustomerDataProvider interface in the DAL and a CustomerManager class in the BLL. In the implementation we also have classes and interfaces to deal with Orders, OrderDetails and Products.

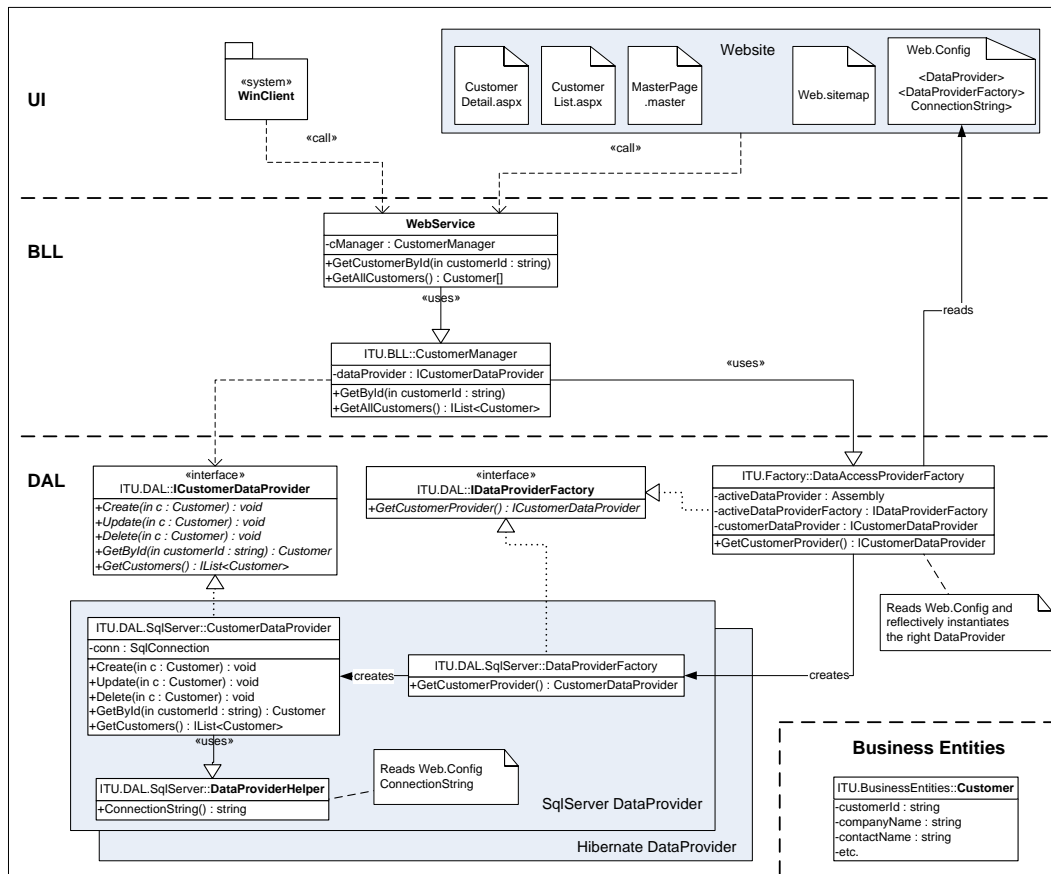


Figure 14 - Class diagram dealing with Customers

### 5.3 Business logic- & data access layer

To ensure loose coupling between the BLL and the DAL we use Factory Methods [9] that reads the `Web.Config` configuration file and reflectively instantiates objects that the BLL uses to communicate with the DAL. In this way by changing an entry in the `Web.Config` the application will use another data provider and therefore even another database. This is done without recompiling anything. The sample code in Listing 4 is from our `Web.Config` file. The `ITU.DAL.Hibernate` version of the data provider is out-commented; however, if the other data provider is out-commented instead, the `Hibernate` version would be used at runtime.

```

<!--
<add key="DataProvider" value="ITU.DAL.Hibernate" />
<add key="DataProviderFactory" value="ITU.DAL.Hibernate.DataProviderFactory" />
-->

<add key="DataProvider" value="ITU.DAL.SqlServer" />
<add key="DataProviderFactory" value="ITU.DAL.SqlServer.DataProviderFactory" />

```

Listing 4 - Web.Config

Furthermore we use the pattern Separated Interface [8] and are always coding against an interface and not the real implementation. In this way we can easily provide a new implementation of the DAL called a data provider that supports another kind of database and change the configuration file to use the new implementation.

The DAL defines interfaces to work on e.g. customers. The `ICustomerDataProvider` interface defines the typical CRUD functionality for working on that a data provider has to implement. As mentioned we have created two different data providers / implementations that work on `SqlServer` and `Hibernate` respectively.

The DAL also defines an `IDataProviderFactory` interface that data providers need to implement. The implementation of this interface is used within the data provider to create the right object of the type `ICustomerDataProvider`. You could imagine that each data provider have different objects that could be used, e.g. one that used optimistic concurrency and one that used another strategy. The `ICustomerDataProvider` interface that defines the CRUD methods is shown in Listing 5.

```
namespace ITU.DAL
{
    public interface ICustomerDataProvider
    {
        void Create(Customer c);
        void Update(Customer c);
        void Delete(Customer c);

        Customer GetById(string customerId);

        DataSet GetCustomersDataSet();
        void UpdateCustomersDataSet(DataSet ds);

        IList<Customer> GetCustomers();
    }
}
```

**Listing 5 - ICustomerDataProvider**

### 5.3.1 Communication through the tiers

In the following example we assume that the `Web.Config` file is set up to use our `SqlServer` data provider.

When either the client running the WinForm or the website requests the web service for a list of customers, the web service calls the `GetAllCustomers()` method on the `CustomerManager`. The `CustomerManager` then uses the `DataAccessProviderFactory` to get an object that implements `ICustomerDataProvider` (here `SqlServer.CustomerDataProvider`) on which the `GetAllCustomers()` method can be called, which return the actual list of customers as a `IList` of `ITU.BusinessEntities.Customer`. To do this the `DataAccessProviderFactory` first reads the `Web.Config` file and reflectively creates an object that implements `IDataProviderFactory` (here `SqlServer.DataProviderFactory`). This factory object creates and returns the right `CustomerDataProvider` object on which the `CustomerManager` can invoke the `GetAllCustomers()` method and return the list of customers to the web service which returns it to the client.

The following code shows what happens in the constructor of the `DataAccessProviderFactory`. The `providerName` and `providerFactoryName` is read from the `Web.Config` file whereafter the assembly is loaded and an instance of the right `DataProviderFactory` is created.

```
string providerName = ConfigurationSettings.AppSettings["DataProvider"];
string providerFactoryName =
ConfigurationSettings.AppSettings["DataProviderFactory"];
activeDataProvider = Assembly.Load(providerName);
activeDataProviderFactory =
(IDataProviderFactory)activeDataProvider.CreateInstance(providerFactoryName);
```

**Listing 6 - Loading the DataProviderFactory**

### 5.3.2 The implemented data providers

In the following we will shortly describe features of the two implemented data providers. As was mentioned earlier the different data providers implement the different interfaces in the DAL.

#### SqlServer data provider

In the SqlServer data provider all methods etc. needed in the application has been implemented against Microsoft SQL Server database. Our application has not been designed to work on DataSets but on objects, therefore the primary focus of this layer is to load Business Entities with data from the database. Since we do not use DataSets which has native support for concurrency, we have to implement this manually. However the sample application is built on top of Microsofts Northwind sample database and this database does not use Timestamp in the database columns. To be able to handle concurrency we could have modified the database or we could have used the other approach described in 3.4.3, hence keeping a copy of the original values. Then the concurrency would have been implemented by verifying that the values haven't been changed from the read operation. If the read operation looked like, `SELECT Column1, Column2 FROM Table1`. Then the update would look like:

```
UPDATE Table1
Set Column1 = @NewValueColumn1,
Set Column2 = @NewValueColumn2
WHERE Column1 = @OldValueColumn1
AND Column2 = @OldValueColumn2
```

**Listing 7 – Concurrency**

If a concurrency violation occurs an exception could be thrown and handled. We have not implemented this.

It was just mentioned that the application was not designed to work on DataSets, however we have provided a couple of methods that actually works with DataSets, to be able to “feel” the difference and learn how DataSets work. Our data access is primary done using SqlCommand and SqlDataReader. An example of how this is implemented can be seen in Listing 8.

```
public IList<Product> GetProducts(string[] productId)
{
    IList<Product> products = new List<Product>();
    try
    {
        string query = String.Format("SELECT ProductID, ProductName,
SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock,
```

```
UnitsOnOrder, ReorderLevel, Discontinued FROM products WHERE ProductID
IN ('{0}'), productId);

SqlCommand myCommand = new SqlCommand(query, myConnection);
myConnection.Open();
SqlDataReader myReader;
myReader = myCommand.ExecuteReader();
while (myReader.Read())
{
    //Add the product to the collection
}
finally
{
    myReader.Close();
    myConnection.Close();
}
return products;
}
```

**Listing 8 – GetProducts**

As can be seen from the code in Listing 8 the SQL statement is coded within quotations. There is no compile time check of the SQL statements which is dangerous due to the potential of runtime errors. If we would have used strongly typed DataSets, there would have been IntelliSense and compile time check of the SQL provided by the DataAdapter.

### Hibernate data provider

In the Hibernate data provider we use the ORM persistence framework NHibernate. NHibernate is a .NET port of the popular Hibernate framework for Java. In NHibernate one are persisting C# classes (in our case the business entities such as Customer) to a database via an XML mapping file. In this mapping file you specify how the fields of an object are going to be mapped to the relational database. NHibernate supports mapping of complex object structures which uses inheritance and collections of other objects as fields. The advantage of using NHibernate is that the framework handles all communication to and from the database and constructs the SQL automatically for the target database. An example of a mapping file from our sample application can be seen below in Listing 9.

```
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  <class name="ITU.BusinessEntities.Customer, ITU.BusinessEntities"
    table="Customers">
    <id name="CustomerID" column="CustomerId" type="String(5)">
      <generator class="assigned" />
    </id>
    <property name="ContactName" column="ContactName"
      type="String(30)" not-null="false" />
    <!-- More properties follows here -->
  </class>
</hibernate-mapping>
```

**Listing 9 - Customers.hbm.xml**

In the mapping file we are specifying that the Customer class in the assembly ITU.BusinessEntities is going to be mapped to the Customers table in the database. Furthermore we specify the primary key and some properties of the Customer object. This mapping file is read by the NHibernate framework when we create a new Configuration and ISessionFactory as in the example code in Listing 10.

```
internal class HibernateHelper
{
    private static readonly Configuration config;
    private static readonly ISessionFactory sessionFactory;

    static HibernateHelper()
    {
        config = new Configuration();
        config.AddAssembly("ITU.DAL.Hibernate");
        sessionFactory = config.BuildSessionFactory();
    }
    // Get methods for config and sessionFactory should be here
}
```

**Listing 10 – HibernateHelper.cs**

When we have an object of ISessionFactory we can create a new ISession on which we can open a session.

```
public abstract class CommonDataProvider
{
    protected ISession GetNewSession()
    {
        return HibernateHelper.GetSessionFactory().OpenSession();
    }
    protected void EndSession(ISession session)
    {
        session.Flush();
        session.Close();
    }
}
```

**Listing 11 – CommonDataProvider.cs**

The OpenSession() method opens a connection to the database that will be used to persist the data in an object. We use the ISession returned from GetNewSession() in our code to persist a Customer object by calling the Save() method as shown below.

```
public class CustomerDataProvider : CommonDataProvider, ICustomerDataProvider
{
    public void Create(Customer c)
    {
        ISession session = GetNewSession();
        session.Save(c);
        EndSession(session);
    }
    // More CRUD methods here
}
```

**Listing 12 – CustomerDataProvider.cs**

The above example is very simple but NHibernate provides lots of possibilities for advanced O-R mapping and querying against the data. Due to the scope of the project we will not go into further details about NHibernate. For more code samples we refer to the source code in Appendix 4.

## 5.4 Presentation layer

The implementation provides both a WebForm (ASP.NET) and a WinForm application. In this section our primary focus is to describe the implementation of the WebForms.

### 5.4.1 ASP.NET 2.0 website

Previously it was described that the layering pattern only allows the user interface to communicate with its neighbouring layer. Since we have a business logic layer that uses business entities to transport the data through the layers, the user interface make use of the ObjectDataSource as a DataSource Control. The ObjectDataSource and other controls were described in section 3.2.3. In Figure 15 it is shown that the ObjectDataSource is configured to receive data from the web service.

It's important to make sure that one does not try to use any data type that is not supported by the XML Web Service using the SOAP protocol. As mentioned we use business entities to pass the data through the tiers. No problems should be encountered since XML Web Services using the SOAP protocol support "Class and struct types with public fields or properties. The public properties and fields are serialized. Classes must have a default constructor that does not accept any parameters" [27].

From Figure 15 it can be seen that the GridView is configured to use the ObjectDataSource as its data source. The GridView is placed in the WebForm, hence the .aspx page. The Custom content placeholder does not default to the content of the Master Page, since we wish to display the data configured in the ObjectDataSource, and not any content from the Master Page. However, the other content placeholder does default to the content of the Master page. From Figure 15, it can be deduced that the left content placeholder implements a TreeView, which is connected to a SiteMapDataSource, which reads its content from a SiteMap file. The SiteMap file is an XML file where we have defined the navigation we wish to use in the application.

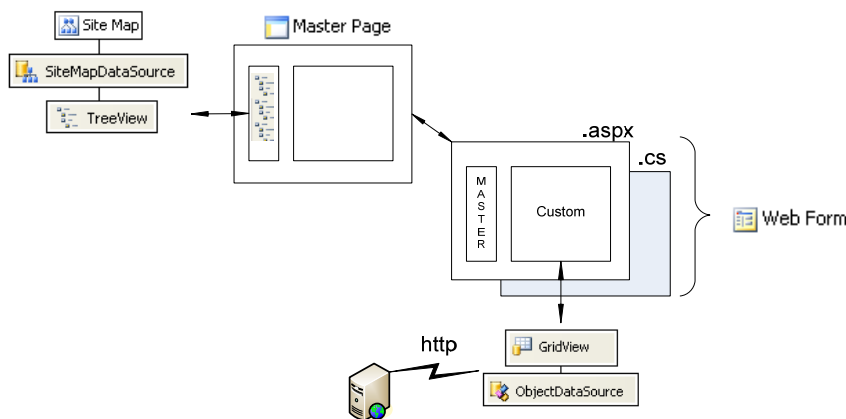


Figure 15 - Sample Application UI website solution (own creation)

We have implemented a couple of WebForms to be able to handle the sample application. We have a CustomerDetail.aspx where the details for a given customer is shown and possibly updated. Before the customer details are shown the customer is chosen from a list. CustomerList.aspx shows all the customers, and enables updating and a link to all the orders

for each customer. The CustomerOrders.aspx shows all the orders for a given customer, again each order consist of a number of orderlines, these are shown by the OrderDetails.aspx. Screenshots of the CustomerList WebForm is shown in Appendix 3.

### 5.4.2 WinClient

The WinForm does not operate and look in exactly the same way as the WebForm. The WinForm providing the same services, calling them from the same web service looks as in Appendix 3.

#### Using asynchronous calls from WinClient

A normal, or synchronous, page holds onto the thread for the duration of the request, preventing the thread from being used to process other requests. If a synchronous request becomes I/O bound—for example, if it calls out to a remote web service or queries a remote database and waits for the call to come back—then the thread assigned to the request is stuck doing nothing until the call returns [4]. As just described, IO-related tasks should be considered to be performed asynchronously. Since web service calls can take a long time to return, pages that execute them are ideal candidates for asynchronous processing. We have implemented both synchronous and asynchronous calls to the web service in the WinClient. The asynchronous calls are done in the following way:

```
/*
 * Async call to webservice, get all the customers in a collection
 * This will only work in ASP.NET 2.0
 */
private void btn_CollectionCustomerAsync_Click(object sender, EventArgs e)
{
    wS.GetAllCustomersCompleted += new
localhost.GetAllCustomersCompletedEventHandler(GetAllCustomersCompleted);
    wS.GetAllCustomersAsync();
}

void GetAllCustomersCompleted(object sender,
localhost.GetAllCustomersCompletedEventArgs e)
{
    localhost.Customer[] customers = e.Result;
    this.UpdateView(customers);
}
```

**Listing 13 - Asynchronous calls**

The above example is new in the .NET Framework 2.0 and is only supported by the .NET Framework version 2.0 web service proxy. The new service proxy includes a method named *NameOfMethodAsync* and an event named *NameOfMethodCompleted*. You can call *NameOfMethod* asynchronously by registering a handler for *NameOfMethodCompleted* events and calling *NameOfMethodAsync*. This was done in Listing 13. The above code is based on [4].

Another thing that was a bit different between the WinForm and the WebForm was the DataBound and DataSource controls. In the WinForm we use *BindingList<T>* when displaying the list of all customers. This enables the *DataGridView* (this control is not available in ASP.NET 2.0 websites) to a list of Customers where Customer is a class with properties like ID, Name, etc. The code in Listing 14 solves the problem:

```
private void UpdateView(localhost.Customer[] customers)
{
    BindingSource bs = new BindingSource();
    BindingList<localhost.Customer> cList = new BindingList<localhost.Customer>();

    foreach (localhost.Customer c in customers)
    {
        cList.Add(c);
    }

    bs.DataSource = cList;
    dataGridView1.DataSource = bs;
}
```

**Listing 14 – BindingList (code based on [28])**

“dataGridView1” will show all the Customers in the BindingList and display columns for each of the properties on Customer i.e. it will have columns for ID, Name, etc. This is shown as the upper-most left window in the WinForm screenshot in Appendix 3. The advantages of BindingList<T> are that as you add, remove, insert Customers into this BindingList the DataGridView will show up the changes and correspondingly add, remove and insert rows [28]. The same approach was used for each DataGridView in the WinForm.

### 5.5 Further implementation issues

In addition to the above description of our implementation it should be mentioned that our code certainly lack certain things. As an example we have not applied any error handling strategy such as defining customer exceptions for various situations. One of the main objectives of good programming is to create robust programs. To be able to do this a good error handling strategy is a must since it will minimize the probability that the program will crash. However, our code should not be viewed as “ready to ship” but be seen as a guideline for how an application could be designed and for us to get familiar with the C# and ASP.NET languages.

There are certain things that must be implemented before any of our code could ever be considered for production, this is mainly unit testing, exception handling and concurrency. In addition to exception handling it should also be mentioned that when working with web services exceptions thrown are sent back to the client in the form of a SOAP fault. Therefore it’s necessary to investigate further on handling and throwing exceptions in XML Web Services and displaying it presentable to the user.

## 6 Comparison of J2EE and .NET

At this point we have examined layered application architecture both in general and in .NET. Furthermore we are now familiar with some of the technologies in .NET in the different layers / tiers. It would be interesting to investigate how all this relate to the J2EE world. The purpose of the following sections is to compare the J2EE and .NET technologies primary in the scope of this project, namely application architecture. However, we will not only compare on what we previously has examined in the .NET environment but also to give a complete overview, a top-to-bottom comparison of the two platforms. This chapter is primarily based on [18] and [12].

### 6.1 Overview of the platforms

The complete overview of the similarities and differences between the two technologies and how they relate to one another are summarized in Figure 16. The figure is based on [18] but has been modified to be up-to-date with the most recent changes in the technologies.

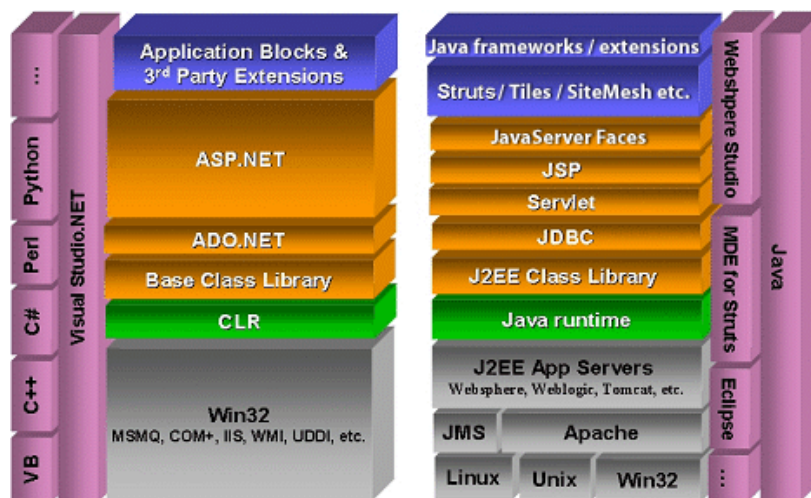


Figure 16 - NET and J2EE platform stack (freely transformed from [18])

In the remainder of this section we will focus on the similarities and differences in Figure 16. The purpose is not to give a complete elaboration, but to present a small summary about the two technologies.

### 6.2 Language and IDE

One of the major differences between the two platforms is the language support. As Figure 16 shows, J2EE only has support for the Java language, where .NET has support for multiple languages. For a complete list of language support see [25].

With regards to the integrated development environment (IDE) J2EE has a broad selection to choose from. Examples are the commercial IBM WebSphere Studio Application Developer, BEA WebLogic Workshop or Eclipse as the open-source alternative.

In .NET there is one primary choice of IDE, namely Visual Studio that supports development of all the different kind of application in the .NET framework.

### 6.3 Application servers and operating systems

Both J2EE and .NET rely on one or more servers to host their applications and to provide essential services such as transactions, security, logging, messaging etc. These servers, and the runtimes themselves, also rely on the base operating system.

J2EE is designed for platform independence and many vendors have built J2EE application servers that run on a variety of operating systems. As such J2EE is very flexible but the platform independence has associated costs such as loss in performance because of missing native integration with the OS.

Unlike J2EE the .NET application server is the Windows Operating System and the services it provides. .NET is closely integrated with IIS and COM+ to provide reliable application hosting services. Therefore .NET runs in a closely integrated environment but also need to run on that exact platform. However, various projects are working on porting the .NET framework to other platforms (e.g. the Mono project for Linux, Unix etc.).

### 6.4 Runtime engines

Both J2EE and .NET run on top of a runtime engine that interprets and runs the code and provides critical services such as memory management and garbage collection. Source files are compiled into bytecode or intermediate language that is interpreted into native machine language at runtime. However there is one big difference between the two. Java is designed for platform independency where the .NET Common Language Runtime (CLR) is designed for optimized performance on the Windows operating system.

### 6.5 Class libraries

Both J2EE (or just Java) and the .NET framework provides a big class library with lots of classes. In Java it is often just called the Java API while in .NET it is called the base class library (BCL). In Java the API is organized into packages, which are logical groupings of the classes while it is called namespaces in .NET.

### 6.6 Technologies in the different layers

As this project is concerned about application design of layered architectures we will in the following sections compare the technologies in the presentation layer, business logic layer and data access layer in J2EE and .NET.

#### 6.6.1 Presentation layer

Both J2EE and .NET offers various web application technologies. In .NET web applications are built using ASP.NET WebForms as described earlier and include different UI controls such as buttons, text fields, GridViews etc. Furthermore ASP.NET is an implementation of the MVC design pattern and the programming model is event-driven as opposed to the old ASP model. ASP.NET provides many valuable features also described earlier.

In J2EE the basic technologies to build web applications is Servlets and JavaServer Pages (JSP). These correspond roughly to the old ASP model, where html markup and scripting is mixed together in one file. Because of problems with this kind of development different Java web frameworks has emerged such as Struts, Tapestry and JavaServer Faces (JSF). All of them implement the MVC pattern to ease the development of web applications. The latter (JSF) is furthermore an UI based framework based on components comparable to ASP.NET's controls and offers many of the same features. JSF is a specification from Sun as is the rest of the J2EE framework. This enables different vendors to provide an implementation of JSF and makes it competitive for the vendors to provide something special and to extent the framework.

### **ASP.NET vs. JSF**

ASP.NET and JSF share many similarities but also differ on some points. Since we think that the presentation layer is quite appealing and ASP.NET and JSF is interesting technologies we will make a somewhat detailed comparison of their features and architecture.

### **Implementation of MVC**

There is a big difference between the two in their implementation of the MVC pattern. The JSF architecture is implemented using a Front Controller pattern [8] called the FacesServlet. This Servlet is responsible for all incoming requests and delegates control to the different classes that take care of validation, navigation, updating model values etc. This means that the Controller is application-wide and every request goes through the FacesServlet in opposite of ASP.NET which uses the Page Controller pattern as described in section 3.2.1.

### **Backing beans / code-behind files**

Both ASP.NET and JSF make it possible to separate UI markup such as html from the actual code in C# or Java. In ASP.NET this technique is called code-behind files and these classes define the event handlers and other code belonging to the page. From these event handler methods you can access the UI components programmatically, change properties etc.

JSF uses the notion of “backing beans” to separate code from markup. Backing beans are JavaBeans that contain action and ActionListener methods as well as properties that can be bound to UI components such as input fields and buttons. In this way you can programmatically access the components in the UI as with ASP.NET and respond to the user's actions like clicking buttons and updating input fields. This actually leads to the event-based programming model that both ASP.NET and JSF provides.

### **Event-based programming**

Both ASP.NET and JSF delivers the intuitive event-based programming which makes development of web applications more like the old Visual Basic UI development and the current Windows Forms environment. However, the way it is implemented differs.

In ASP.NET any events such as a Click event are handled by delegates that wrap methods (event handlers) on the page itself. So you usually end up with a single page that encapsulates all of the controls for the page, is responsible for displaying it, and handles all events associated with the page.

In JSF you associate events such as Click events with action or ActionListener methods in a backing bean via method binding expressions. These expressions is written in JSF-EL (Expression Language) which also is used in value binding expressions where the value of e.g. a input filed is bound to a property in the bean.

Even though they differ in the semantics they both provide a very convenient way to hook up UI components to methods that perform the requested action. This abstraction has made web development much easier.

### **Navigation**

Both ASP.NET and JSF have the possibility to externalize the navigation to an XML file. In ASP.NET you define a sitemap file as described in section 3.2.5 which can be used from different controls to handle navigation. In JSF you define navigation rules and outcomes in faces-config.xml which maps logical outcomes such as “view\_products” to a physical view (e.g. a JSP page). These logical outcomes can then be returned from action methods in a backing bean to show the correct view. There is no hard-coded url redirecting from the backing bean code. In ASP.NET however, you often see hard-coded navigation in the methods in the code-behind file with the use of `Server.Transfer("somefile.aspx")`. This can make a strong dependency between different pages, which is not always desirable if change in the application is to be implemented.

### **Templating**

The ability to make consistent looking web applications is very powerful with the new master page feature in ASP.NET as described in section 3.2.4. It makes it easy to reuse and include content on different pages. JSF does not have a template mechanism beside the `<jsp:include>` element. This is okay for very simple reuse of content, but JSF misses another mechanism. However, other open-source J2EE frameworks provide this functionality like the Jakarta Struts Tiles project or OpenSymphony’s SiteMesh framework.

### **Validation**

A part of any application that process input from the user validation is often required to ensure that input have been entered in the right format, the user has selected something in a dropdown menu etc. Both ASP.NET and JSF provides validation controls / components that can be associated with the different input controls. The key difference is that ASP.NET support the use of client-side validation where JSF only supports server-side. This requires that JSF has to do a roundtrip to the server every time the user clicks a button and values should be validated. This can result in performance and user experience disadvantages in our opinion.

### **Rendering**

ASP.NET components render themselves directly on the page. This is different from JSF where components can render themselves, but more often they delegate rendering to special Renderer objects made available in a RenderKit. A different RenderKit can be supplied for each different type of medium like html, wap etc. A default HTML RenderKit is provided by

every JSF implementation. But this means that the same JSF component can be rendered differently in a web browser or wireless device by simply changing to a different RenderKit. This is a powerful capability and leads to the general principle in JSF about extensibility of the framework

### **Extensibility / plugability**

Since JSF is a specification and not a product developed by one vendor, it has been designed to make it possible for vendors and the application developer himself to provide alternative and powerful implementations and plug in new features to the framework. This means that most of the framework can be extended and switched to support different things such as providing another ViewHandler or NavigationHandler. As a result you can e.g. use another view technology than JSP like the open-source alternative Facelets. Although the extensibility in JSF is a great feature it may not be as relevant regarding ASP.NET because there is only one vendor.

### **IDE support**

As a final point we wish to elaborate on the IDE / tool support in the two technologies. Both ASP.NET and JSF was designed with tool support in mind, but the IDEs differ much in features and ease of use. ASP.NET has a very nice IDE in Visual Studio which has great functionalities regarding drag 'n' drop and easy and visual preview of pages, wizards for configuring the controls etc. We think that it is a little harder to find such an integrated IDE for JSF, although IDEs like Oracle JDeveloper and Sun's Java Studio Creator is very competitive. But the choice of IDE is also tight coupled to the choice of J2EE application server since they often is highly integrated. You may loose some features if you develop in JDeveloper and deploys your web application to the JBoss Application Server instead of Oracle Application Server.

### **6.6.2 Business logic layer**

When it comes to the business logic layer J2EE and .NET both provides enterprise services such as security, transaction support, connection pooling, message queuing etc. Although these services are alike and almost delivers the same functionality their implementation is quite different. As described in section 3.3.1 the .NET framework offers these services primarily through Component Services (COM+).

J2EE on the other hand offer the Enterprise Java Bean (EJB) server-side component architecture. EJBs are the roughly equivalent of COM+ components or managed code components, and can provide services/cross-cutting concerns such as maintaining state, transactional support, method level security, message queue support and persistence support.

EJBs are deployed to a J2EE Application Server such as IBM WebSphere and are encapsulated in a component container. The container is responsible for receiving client requests and mediates the calls to the EJB in order to control the cross-cutting concerns. The EJBs properties such as transaction level, JNDI name and security options are configured through an XML deployment descriptor or as Java annotations on classes and methods (the latter only in EJB 3.0).

EJBs exist in three different forms:

- Session beans (stateless and stateful)
- Message-driven beans
- Entity beans

**Session beans** is used to model business logic and often provides some kind of business critic transactions or services such as retrieving all customers from a persistent storage. Session beans can be stateless which means they can't maintain state across method calls and their instances exists in object pools. They are often used to model a Session Facade [26] as a gateway to the server-side application and business logic. Stateful session beans maintain state across multiple method calls, tying the client to the session bean for the object's lifetime. This makes it possible to e.g. model a shopping cart in a web application which has to maintain state across requests.

**Message-driven beans (MDB)** are used in connection with JMS (Java Message Service) to process messages asynchronously. MDBs are JMS message consumers and clients do not access MDBs directly; they send a JMS message to a destination (JMS Server) where the MDB is listening. This enables reuse of the MDB, and also means that the developer does not have to worry about where to go to get the message. MDBs are like session beans used to implement business logic.

**Entity beans** are used to give an object-oriented view on relational data. In general you can say that they represent a domain object in persistent storage. In EJB 3.0 entity beans provide almost the same functionality as the object-relational mapping (ORM) tool Hibernate and are POJOs (Plain Old Java Objects). This means that they are normal Java objects that can be detached from the container, serialized and transferred all the way to the client. In that sense they replace the Data Transfer Object [8]. In the following section about the data access layer in J2EE and .NET we will shortly describe entity bean because they play a vital role in J2EE persistence.

### 6.6.3 Data access layer

Both J2EE and .NET contain data access libraries to access databases and other data sources. The core features of each library allow similar direct database access by opening a connection, submitting a command and receiving the results.

In .NET this task is accomplished using ADO.NET with SqlAdapters, DataReaders, DataSets etc. as described in section 3.4.1. ADO.NET is based on a disconnected model because of the DataSet and supports optimistic concurrency and the ability to read and write XML data. ADO.NET furthermore has the Strongly Typed DataSets as described in section 3.4.1.

In J2EE (or just J2SE actually) fundamental database access is accomplished with JDBC (Java Database Connectivity). JDBC is a set of interfaces that vendors have to implement in order to provide a JDBC driver for their database. This means that JDBC is not a product but a specification as the rest of J2EE. The JDBC library supports a very traditional approach to database access. The user submits a Statement (usually a query or stored procedure call) and may receive results in the form of a ResultSet with a live connection to the database. However, JDBC also supports disconnected result sets called CachedRowSets that supports serialization and optimistic concurrency.

JDBC does not directly support reading or writing XML data as ADO.NET. Although several vendors have created JDBC wrappers for XML-based database access, the core J2EE classes do not support it [12].

As mentioned in section 6.6.2 J2EE also provides **entity beans**. Entity beans represent an object in persistent storage. Each instance of an entity bean corresponds to a row of data from a database table, although it is not always that simple because of different mapping approaches between the relational and object-oriented world. There are two types of entity beans; container managed persistence (**CMP**) beans, which are managed by the container; and bean managed persistence (**BMP**) beans, which manage their own persistence (the programmer).

With CMP entity beans, the container manages the mapping between bean and database fields using a XML deployment descriptor or by reading Java annotations on the bean class. The container manages all the communications with the database and no database code is needed from the programmer of the bean.

With BMP entity beans, the developer is responsible for writing all the JDBC code for transferring data between a bean and the data store. BMPs require the developer to write much more data related code than CMP but in this way optimization and flexibility can be achieved.

As part of the entity beans specification is an object-oriented query language called EJB-QL. This is used to perform queries on the entity beans that at runtime are translated into the corresponding SQL calls targeted for the database in use. This helps enforce maintainability and makes it possible to switch the underlying database very easily. You can however also use native SQL queries on the entity beans to optimize performance or to use specialized features in your database system.

### 6.7 Summary of J2EE versus .NET

Throughout this section we have compared some of the features and technologies in J2EE and .NET. To give an overview we have listed the features and technologies in Table 1 on the next page (freely transformed and updated from [12]).

## 6 Comparison of J2EE and .NET

Feature or Service	Microsoft .NET	J2EE	Comments
Technology Type	Product	Standard/Specification	
Middleware Vendors	Microsoft and partners	50+ vendors	
Client Side GUI	Windows Forms	AWT/SWING	Part of J2SE
Web GUI	ASP.NET	Servlets /JSP and JavaServer Faces	
Web Scripting	ISAPI HttpHandler HttpModule	Servlet Filter	
Web Application Hosting	Internet Information Server	Multiple (depends on vendor)	J2EE examples include Apache Tomcat
Runtime engine	CLR (Common Language Runtime)	JRE (Java Runtime Engine)	
Server Side Business Logic Component	.NET Class or Serviced Component (COM+)	EJB Session Beans	
Server Side Data Components 1	Serviced Component with DB Logic	EJB with Bean Managed Persistence	
Server Side Data Components 2	ADO.NET Data Set	EJB with Container Managed Persistence	Only an approximate equivalence!
Directory Access	Active Directory Services Interface (ADSI) through LDAP	Java Naming and Directory Service (JNDI) through LDAP	LDAP compatibility makes switching between directory services very easy.
Remote Invocation	.NET Remoting	RMI-IIOP	
Data Access	ADO.NET	JDBC, SQL/J, JDO	
Messaging	Microsoft Message Queuing	JMS (Java Message Service)	Microsoft Message Queuing is a product. JMS is a specification
Transactional Support	COM+/Distributed Transaction Controller (DTC)	JTA (Java Transaction API)	

**Table 1 - J2EE versus .NET**

## 7 Evaluation of ASP.NET 2.0

ASP.NET 2.0 and .NET in general is a powerful and well-supported software development framework and it has been both interesting and educational to work with various aspects of this technology. One of the strengths of the .NET framework is that it is based on XML, which facilitates communication between different devices and software systems. It has thus been interesting to see how web services can be used to create communication paths between different systems. Another great advantage of the .NET framework is the tool support integrated in Visual Studio. With Visual Studio it is e.g. easy to create functional GUIs for both standard applications and for websites. Normally this simplicity is partially a result of the direct data binding that is established. In our case we have implemented a middle tier that all the control is bound to and thereby removed the direct data binding from the presentation controls to the database. This was actually quite easy due to the new ObjectDataSource control in ASP.NET 2.0.

The ADO.NET technology provides a powerful and efficient tool for linking the applications to a database and managing concurrency issues, even that this was not used in the implementation. In relation to the data access layer, we have also experimented with NHibernate that seemed quite nice to work with.

Much of the efficiency when working in Visual Studio comes from the fact that most of the application code is auto generated. Besides the efficiency this has the advantage that it is ensured that these code parts are constructed in a uniform fashion. It does, however, also have a disadvantage, namely that the developer only has direct insight into a smaller part of the code.

Overall, we find that the .NET framework is very well suited for developing business and enterprise applications efficiently, and we are convinced that the .NET technologies will continue to be a great alternative or primary choice in regards to J2EE. As a final remark we think that it is noteworthy to mention the good effect of the increased competition in the field of enterprise development frameworks. .NET really pushes the technology further and forces the other competitors like J2EE to develop even more competitive enterprise application frameworks.

## 8 Conclusion

As stated in the problem definition the purpose of this project has been to discuss architecture and application design of object-oriented distributed applications in general and in relation to .NET. Differences between J2EE and .NET in relation to these subjects were going to be examined. Further based on our studies a small sample application that illustrates blueprints of a .NET-based layered application architecture was going to be implemented. Our intention was also to learn about features in ASP.NET 2.0 and Visual Studio 2005.

The scope of this project has been wide. In order to answer the problem definition we have conducted a lot of literature studies and in order to write this report we have also conducted the following development activities: software architecture, database design, database implementation, network configuration and we have worked with a wide range of technologies/tools: XML, C#, Visual Studio, CVS, SQL Server, ASP.NET and ADO.NET. The size of this project has at times been both frustrating and overwhelming, but we have certainly learned a lot by completing the project. It has been very satisfying to explore layered application design in the .NET framework and especially ASP.NET 2.0. Further the comparison with J2EE has provided us with a great overview of the similarities and differences between the two frameworks. The comparison also provided us with great insight about what enterprise application frameworks are able to provide today.

Based on the fact that we have fulfilled the goals of the project and also learned about new features in ASP.NET 2.0, the C# programming language through our implementation of the sample application, the .NET environment and Visual Studio 2005 we conclude that the project has been successful.

## 9 Perspectives

Even though the wide scope of this project have allowed us to examine lots of different features in .NET and to some extent J2EE there are still many interesting areas left out. Regarding the data access layer some pretty exiting things is happening around C# 3.0 that includes its new Language Integrated Query (LINQ), which provides a standard way of querying non-object-oriented from within the .NET languages. In this project we have discussed layered architecture. In relation to the data access layer it could be interesting to examine what effect LINQ will have on the application design.

In relation to the presentation layer new exciting things are also happening. In the web context a new paradigm is evolving with the AJAX technology. It could be interesting to investigate AJAX in relation to ASP.NET and JavaServer Faces. Both have some initiatives going on in the area and especially Microsoft is promoting their version called ATLAS. The idea of using Asynchronous JavaScript and XML in web applications will certainly enrich the user experience on the web. Another UI related technology from Microsoft for developing rich clients is Windows Presentation Foundation that is launched with Windows Vista. This is going to replace the current Windows Forms applications and provides some really interesting features because of 3D acceleration, vector graphics and the possibility to declarative build your interfaces in XAML.

As a final remark it could be really interesting to examine how SOA is going to change application architecture and communication between processes. Also here Microsoft is launching a new platform together with Windows Vista called Windows Communication Foundation. This is a new API that unifies the former technologies in .NET like Remoting and XML Web Services to provide a foundation for the future SOA-based applications.

## 10 Bibliography

### Magazine articles:

- [1] Esposito, Dino  
DataSets vs. Collections  
MSDN Magazine, August 2005
- [2] John Papa  
Efficient Coding With Strongly Typed DataSets  
MSDN Magazine, December 2004
- [3] Prosize, Jeff  
An Overview Of The New Services, Controls, And Features In ASP.NET 2.0  
MSDN Magazine, 2006
- [4] Prosize, Jeff.  
Asynchronous Pages in ASP.NET 2.0  
MSDN Magazine, October 2005

### Books:

- [5] Crocker, Angela., Olsen, Andy. and Jezierski, Edward  
Designing Data Tier Components and Passing Data Through Tiers  
Microsoft patterns & practices aticle, August 2002  
URL <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnda/html/BOAGag.asp>  
Last accessed 2006-05-17
- [6] Darie, Cristian. and Watson, Karli  
Beginning ASP.NET 2.0 E-Commerce in C# 2005  
Apress, 2006
- [7] Esposito, Dino  
Introducing Microsoft ASP.NET 2.0  
Microsoft Press, 2005
- [8] Fowler, Martin  
Patterns of Enterprise Architecture  
Addison-Wesley Professional, 2002
- [9] Gamma, Erich., Helm, Richard., Johnson, Ralph., John Vlissides  
Design Patterns: Elements of Reusable Object-Oriented Software  
Addison-Wesley Professional Comuting series, January 15, 1995
- [10] Glenn Johnson  
Programming Mircrosoft ADO.NET 2.0 Applicationc Advanced Topics, 2005 edition  
Microsoft Press, 2006

- [11] Larman, Craig  
Applying UML and Patterns: An introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd ed.  
Pearson Education Inc., 2005
- [12] Laudati, Peter., et al.  
Application Interoperability: Microsoft .NET and J2EE  
Microsoft pattern & practices, December 2003  
URL <http://msdn.microsoft.com/library/en-us/dnpag/html/JDNI.asp>
- [13] Selly, Dominic., Troelsen, Andres. and Tom Barnaby  
Expert ASP.NET 2.0 Advanced Application Design  
Apress, 2006
- [14] Singh, Inderjeet., Stearns, Beth., Johnson, Mark. and the Enterprise Team  
Designing Enterprise Applications with the J2EE(TM) Platform, Second Edition  
Addison-Wesley, 2002
- [15] Webb, Jeff  
Developint XML Web Services and Server Componentes with Microsoft Visual Basic .NET and Visual C# .NET  
Microsoft Press, 2003

**Online articles:**

- [16] Chaffee, Alex  
One, two, three, or n tiers  
URL <http://www.javaworld.com/javaworld/jw-01-2000/jw-01-ssj-tiers.html>  
Last accessed 2006-05-18
- [17] Dobson, Rick  
Using DataReaders to Increase Speed and Reduce Memory  
URL <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnhcvs04/html/vs04g6.asp>  
Last accessed 2006-05-17
- [18] Mortensen, M. Keith., McGovern, Rob., Liptaak, Charles  
ASP.NET and Struts: Web Application Architectures  
MSDN Library article, December 2003  
URL <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/ASPNet-ASPNet-J2EE-Struts.asp>  
Last accessed 2006-05-24
- [19] Sheriff, Paul D.  
Building an N-Tier Application in .NET  
MSDN Library article, February 2002  
URL <http://msdn.microsoft.com/library/en-us/dndotnet/html/buildntierapp.asp>  
Last accessed 2006-05-17

- [20] Sheriff, Paul D.  
Designing a .NET Application  
MSDN Library article, April 2002  
URL <http://msdn.microsoft.com/library/en-us/dndotnet/html/designnetapp.asp>  
Last accessed 2006-05-17
  
- [21] Rørbæk, Jeppe  
ASP.NET 2.0 DataSourceControls og Databinding  
URL [download.microsoft.com/download/3/3/c/33c00d2e-05f9-4655-a35b-5e5356705a0e/TechTalk\\_DataSourceControls.ppt](download.microsoft.com/download/3/3/c/33c00d2e-05f9-4655-a35b-5e5356705a0e/TechTalk_DataSourceControls.ppt)  
Last accessed 2006-05-23
  
- [22] Walther, Stephen  
Working with the ASP.NET 2.0 ObjectDataSource Control  
URL <http://msdn.microsoft.com/asp.net/reference/data/default.aspx?pull=/library/en-us/dnvs05/html/asp2objectdatasource.asp>  
Last accessed 2006-05-23

**World Wide Web:**

- [23] ASP.NET Master Pages Overview  
URL <http://msdn2.microsoft.com/en-US/library/wtxbf3hh.aspx>  
Last accessed 2006-05-23
  
- [24] ASP.NET Data-Bound Web Server Controls Overview  
URL <http://msdn2.microsoft.com/en-US/library/ms228214.aspx>  
Last accessed 2006-05-23
  
- [25] Brian Ritchie  
Brian Ritchie's .NET Development Site  
URL <http://dotnetpowered.com/languages.aspx>  
Last accessed 2006-05-24
  
- [26] Core J2EE Pattern Catalog  
Session Facade  
URL <http://java.sun.com/blueprints/corej2eepatterns/Patterns/SessionFacade.html>  
Last accessed 2006-05-26
  
- [27] Data Types Supported by XML Web Services Created Using ASP.NET  
.NET Framework Developer's Guide  
URL <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcondatatypesupportedbywebservices.asp>  
Last accessed 2006-05-24
  
- [28] Dinesh Chandnani, .Net Client Team  
BindingSource and BindingList Of T - DataBinding made simple  
URL <http://blogs.msdn.com/dchandnani/archive/2005/03/12/394438.aspx>  
Last accessed 2006-05-24

- [29] Komponentbaseret Design og J2EE lecture slides  
URL <http://www.itu.dk/courses/SKBD/F2006/lecture2.pdf>  
Last accessed 2006-05-26
  
- [30] Le, Alan et al.  
Microsoft .NET Pet Shop 4: Migrating an ASP.NET 1.1 Application to 2.0  
Microsoft Corporation and Vertigo Software, Inc., February 2006  
URL <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/bdasamppet4.asp>  
Last accessed 2006-05-22
  
- [31] Managed/designs Northwind Starter Kit  
URL <http://download.manageddesigns.it/nsk.aspx>  
Last accessed 2006-05-22
  
- [32] Microsoft Corporation  
Data Access Application Block  
Microsoft patterns & practices, January 2006  
URL [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/entlibjan2006\\_dataaccessappblock.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/entlibjan2006_dataaccessappblock.asp)  
Last accessed 2006-05-22
  
- [33] Peter Sestoft  
C# Project Cluster webpage  
URL <http://www.itu.dk/people/sestoft/csharp/>  
Last accessed 2006-04-29
  
- [34] Wikipedia The Free Encyclopedia  
<http://en.wikipedia.org/wiki/ADO.NET>  
Last accessed 2006-05-22

## 11 Appendices

### **Appendix 1:**

Overview Of The New Services, Controls, And Features In ASP.NET 2.0 not examined in the report.

### **Appendix 2:**

Northwind as a strongly typed DataSet

### **Appendix 3:**

Screenshots of WebForm and WinForm

### **Appendix 4:**

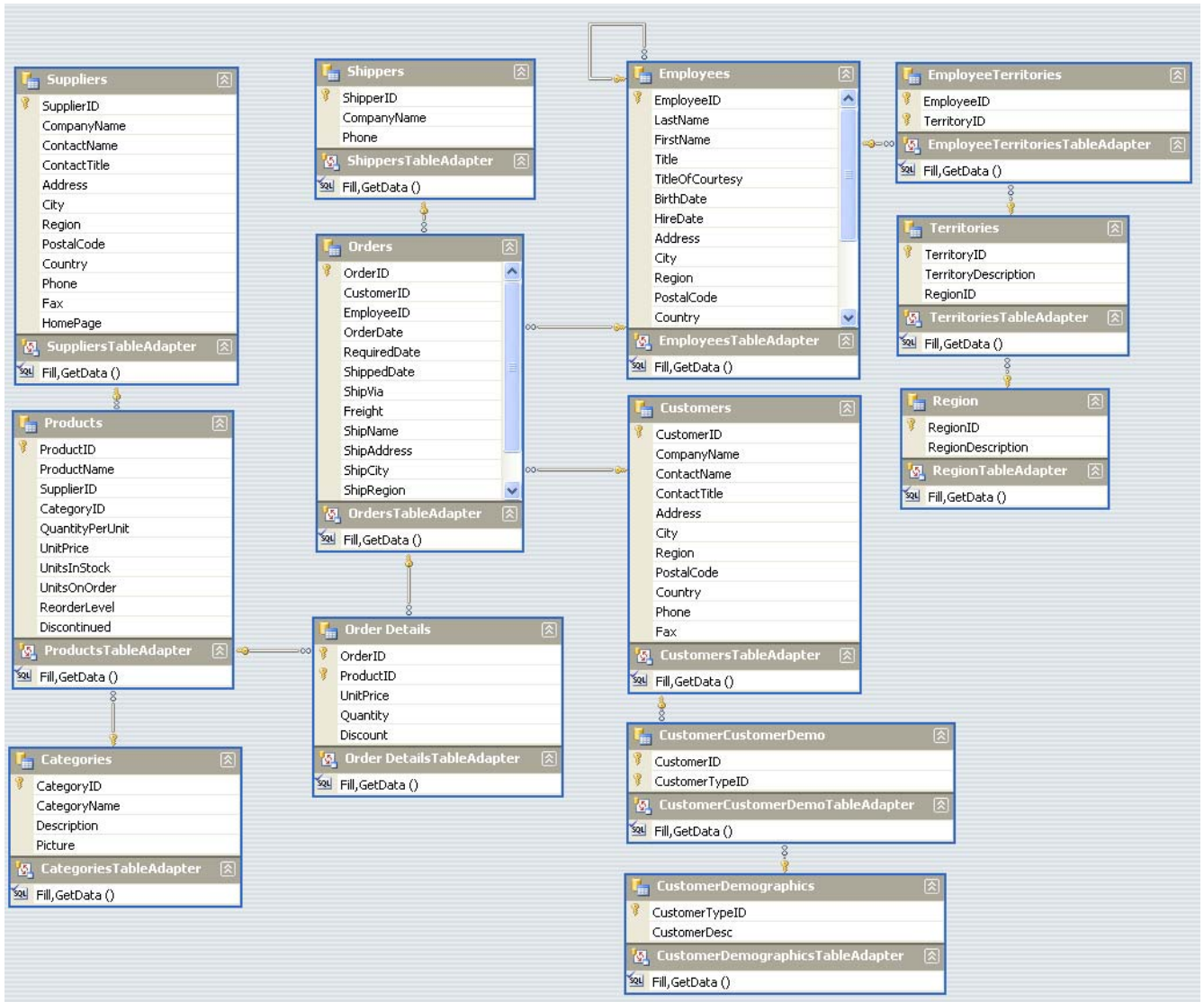
Source code

## **Appendix 1: Overview Of The New Services, Controls, And Features In ASP.NET 2.0 not examined in the report.**

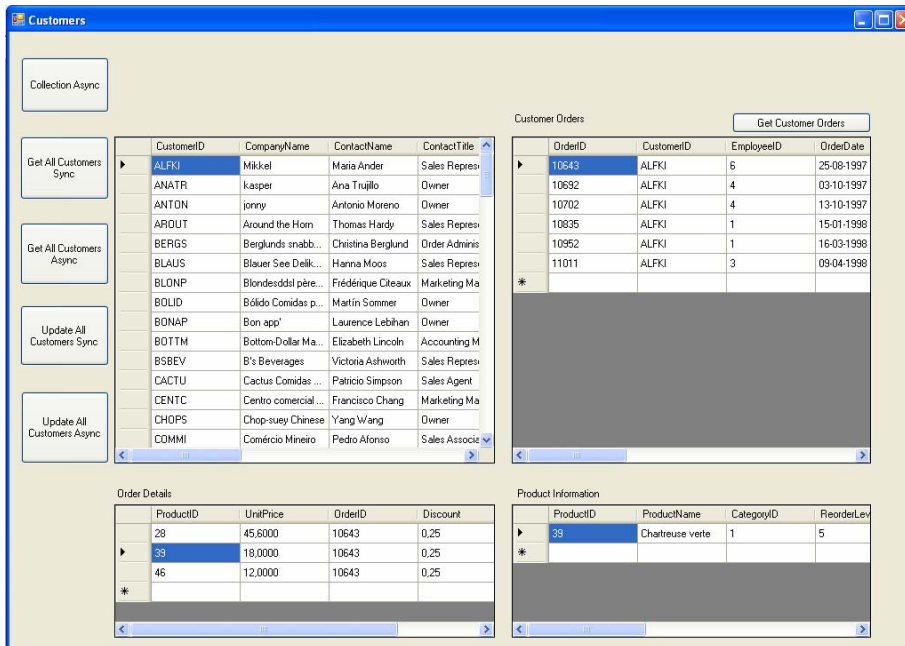
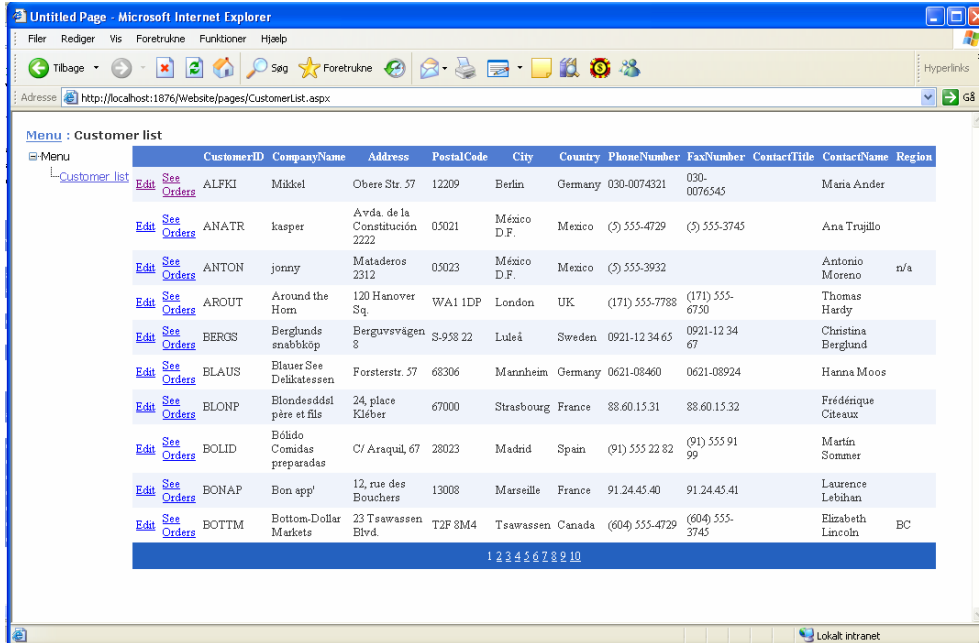
- New Dynamic Compilation Model
- Precompiling and Deploying Without Source
- Other New Controls
- Membership Service
- Login Controls
- Role Management
- Profiles
- URL Mapping
- SQL Cache Dependencies
- Validation Groups
- Cross-Page Postbacks
- Client Callback Manager (XML-HTTP)
- Asynchronous Pages
- Encrypted Configuration Sections

This is based on [3]

## Appendix 2: Northwind as a strongly typed DataSet



## Appendix 3: Screenshots of WebForm and WinForm



## **Appendix 4: Source code**

The full source code is included from the next page and forward.

**ITU.BusinessEntities**

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace ITU.BusinessEntities
6 {
7     [Serializable]
8     public class Customer
9     {
10         private string customerID = string.Empty;
11         private string companyName = string.Empty;
12         private string contactName = string.Empty;
13         private string contactTitle = string.Empty;
14         private string address = string.Empty;
15         private string city = string.Empty;
16         private string region = string.Empty;
17         private string postalCode = string.Empty;
18         private string country = string.Empty;
19         private string phone = string.Empty;
20         private string fax = string.Empty;
21
22
23         //private string originalcustomerID = string.Empty;
24         //private string originalCompanyName = string.Empty;
25         //private string originalContactName = string.Empty;
26         //private string originalContactTitle = string.Empty;
27         //private string originalAddress = string.Empty;
28         //private string originalCity = string.Empty;
29         //private string originalRegion = string.Empty;
30         //private string originalPostalCode = string.Empty;
31         //private string originalCountry = string.Empty;
32         //private string originalPhoneNumber = string.Empty;
33         //private string originalFaxNumber = string.Empty;
34
35         public Customer() { }
36
37         public Customer(string customerID, string companyName, string contactName,
38             string contactTitle, string address, string city, string region, string postalCode,
39             string country, string phone, string fax)
40         {
41             this.customerID = customerID;
42             this.companyName = companyName;
43             this.contactName = contactName;
44             this.contactTitle = ContactTitle;
45             this.address = address;
46             this.city = city;
47             this.region = region;
48             this.postalCode = postalCode;
49             this.country = country;
50             this.phone = phone;
51             this.fax = fax;
52         }
53
54         public string CustomerID
55         {
56             get { return customerID; }
57             set { customerID = value; }
58         }
59
60         public string CompanyName
61         {
62             get { return companyName; }
63             set { companyName = value; }
64         }
65
66         public string ContactName
67         {
68             get { return contactName; }
69             set { contactName = value; }
70         }
71
72         public string ContactTitle
73         {
```

```
72         get { return contactTitle; }
73         set { contactTitle = value; }
74     }
75
76     public string Address
77     {
78         get { return address; }
79         set { address = value; }
80     }
81
82     public string City
83     {
84         get { return city; }
85         set { city = value; }
86     }
87
88     public string Region
89     {
90         get { return region; }
91         set { region = value; }
92     }
93
94     public string PostalCode
95     {
96         get { return postalCode; }
97         set { postalCode = value; }
98     }
99
100    public string Country
101    {
102        get { return country; }
103        set { country = value; }
104    }
105
106    public string Fax
107    {
108        get { return fax; }
109        set { fax = value; }
110    }
111
112    public string Phone
113    {
114        get { return phone; }
115        set { phone = value; }
116    }
117    }
118 }
119
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace ITU.BusinessEntities
6 {
7     public class Product
8     {
9         private string productID = string.Empty;
10        private string productName = string.Empty;
11        private string supplierID = string.Empty;
12        private string categoryID = string.Empty;
13        private string quantityPerUnit = string.Empty;
14        private string unitPrice = string.Empty;
15        private string unitsInStock = string.Empty;
16        private string unitsOnOrder = string.Empty;
17        private string reorderLevel = string.Empty;
18        private string discontinued = string.Empty;
19
20        public Product()
21        {
22        }
23
24        public Product(string productID, string productName, string supplierID, string
categoryID, string quantityPerUnit, string unitPrice, string unitsInStock, string
unitsOnOrder, string reorderLevel, string discontinued)
25        {
26            this.productID = productID;
27            this.productName = productName;
28            this.supplierID = supplierID;
29            this.categoryID = categoryID;
30            this.quantityPerUnit = quantityPerUnit;
31            this.unitPrice = unitPrice;
32            this.unitsInStock = unitsInStock;
33            this.unitsOnOrder = unitsOnOrder;
34            this.reorderLevel = reorderLevel;
35            this.discontinued = discontinued;
36        }
37
38        public string ProductID
39        {
40            get { return productID; }
41            set { productID = value; }
42        }
43
44        public string ProductName
45        {
46            get { return productName; }
47            set { productName = value; }
48        }
49
50        public string SupplierID
51        {
52            get { return supplierID; }
53            set { supplierID = value; }
54        }
55
56        public string CategoryID
57        {
58            get { return categoryID; }
59            set { categoryID = value; }
60        }
61
62        public string QuantityPerUnit
63        {
64            get { return quantityPerUnit; }
65            set { quantityPerUnit = value; }
66        }
67
68        public string UnitPrice
69        {
70            get { return unitPrice; }
71            set { unitPrice = value; }
```

```
72     }
73
74     public string UnitsInStock
75     {
76         get { return unitsInStock; }
77         set { unitsInStock = value; }
78     }
79
80     public string UnitsOnOrder
81     {
82         get { return unitsOnOrder; }
83         set { unitsOnOrder = value; }
84     }
85
86     public string ReorderLevel
87     {
88         get { return reorderLevel; }
89         set { reorderLevel = value; }
90     }
91
92     public string Discontinued
93     {
94         get { return discontinued; }
95         set { discontinued = value; }
96     }
97 }
98 }
99 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.Data;
5
6 namespace ITU.BusinessEntities
7 {
8     public class Order
9     {
10         private string orderID = string.Empty;
11         private string customerID = string.Empty;
12         private string employeeID = string.Empty;
13         private string orderDate = string.Empty;
14         private string requiredDate = string.Empty;
15         private string shippedDate = string.Empty;
16         private string shipVia = string.Empty;
17         private string freight = string.Empty;
18         private string shipName = string.Empty;
19         private string shipAddress = string.Empty;
20         private string shipCity = string.Empty;
21         private string shipRegion = string.Empty;
22
23         public Order()
24         {
25         }
26
27         public Order(string orderID, string customerID, string employeeID, string
28 orderDate, string requiredDate, string shippedDate, string shipVia, string
29 freight, string shipName, string shipAddress, string shipCity, string shipRegion)
30         {
31             this.orderID = orderID;
32             this.customerID = customerID;
33             this.employeeID = employeeID;
34             this.freight = freight;
35             this.orderDate = orderDate;
36             this.orderID = orderID;
37             this.shipAddress = shipAddress;
38             this.shipName = shipName;
39             this.shipCity = shipCity;
40             this.shippedDate = shippedDate;
41             this.shipVia = shipVia;
42             this.shipRegion = shipRegion;
43         }
44
45         public string OrderID
46         {
47             get { return orderID; }
48             set { orderID = value; }
49         }
50
51         public string CustomerID
52         {
53             get { return customerID; }
54             set { customerID = value; }
55         }
56
57         public string EmployeeID
58         {
59             get { return employeeID; }
60             set { employeeID = value; }
61         }
62
63         public string OrderDate
64         {
65             get { return orderDate; }
66             set { orderDate = value; }
67         }
68
69         public string RequiredDate
70         {
71             get { return requiredDate; }
72             set { requiredDate = value; }
73         }
74     }
75 }
```

```
72
73     public string ShippedDate
74     {
75         get { return shippedDate; }
76         set { shippedDate = value; }
77     }
78
79     public string ShipVia
80     {
81         get { return shipVia; }
82         set { shipVia = value; }
83     }
84
85     public string Freight
86     {
87         get { return freight; }
88         set { freight = value; }
89     }
90
91     public string ShipName
92     {
93         get { return shipName; }
94         set { shipName = value; }
95     }
96
97     public string ShipAddress
98     {
99         get { return shipAddress; }
100        set { shipAddress = value; }
101    }
102
103    public string ShipCity
104    {
105        get { return shipCity; }
106        set { shipCity = value; }
107    }
108
109    public string ShipRegion
110    {
111        get { return shipRegion; }
112        set { shipRegion = value; }
113    }
114 }
115 }
116
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace ITU.BusinessEntities
6 {
7     public class OrderDetail
8     {
9         private string orderID = string.Empty;
10        private string productID = string.Empty;
11        private string unitPrice = string.Empty;
12        private string quantity = string.Empty;
13        private string discount = string.Empty;
14
15        public OrderDetail()
16        {
17        }
18
19        public OrderDetail(string orderID, string productID, string unitPrice, string
20        quantity, string discount)
21        {
22            this.orderID = orderID;
23            this.productID = productID;
24            this.unitPrice = unitPrice;
25            this.quantity = quantity;
26            this.discount = discount;
27        }
28        public string OrderID
29        {
30            get { return orderID; }
31            set { orderID = value; }
32        }
33
34        public string ProductID
35        {
36            get { return productID; }
37            set { productID = value; }
38        }
39
40        public string UnitPrice
41        {
42            get { return unitPrice; }
43            set { unitPrice = value; }
44        }
45
46        public string Quantity
47        {
48            get { return quantity; }
49            set { quantity = value; }
50        }
51
52        public string Discount
53        {
54            get { return discount; }
55            set { discount = value; }
56        }
57    }
58 }
59
```

**ITU.DAL**

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace ITU.DAL
6 {
7     public interface IDataProviderFactory
8     {
9         ICustomerDataProvider GetCustomerProvider();
10        IOrderDataProvider GetOrderProvider();
11        IProductDataProvider GetProductProvider();
12    }
13 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.BusinessEntities;
5 using System.Data;
6
7 namespace ITU.DAL
8 {
9     public interface IOrderDataProvider
10    {
11        void Create(Order o);
12        void Update(Order o);
13        void Delete(Order o);
14
15        Order GetById(string orderId);
16
17        DataSet GetOrdersDataSet();
18        void UpdateOrdersDataSet(DataSet ds);
19
20        IList<Order> GetOrders(string customerId);
21        IList<OrderDetail> GetOrderDetails(string orderId);
22    }
23 }
24
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.BusinessEntities;
5 using System.Data;
6
7 namespace ITU.DAL
8 {
9     public interface ICustomerDataProvider
10    {
11        void Create(Customer c);
12        void Update(Customer c);
13        void Delete(Customer c);
14        Customer GetById(string customerId);
15
16        DataSet GetCustomersDataSet();
17        void UpdateCustomersDataSet(DataSet ds);
18
19        IList<Customer> GetCustomers();
20    }
21 }
22
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.BusinessEntities;
5
6 namespace ITU.DAL
7 {
8     public interface IProductDataProvider
9     {
10         Product GetById(string productId);
11         IList<Product> GetProducts(string[] productId);
12     }
13 }
14
```

**ITU.DAL.Factory**

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.Configuration;
5 using System.Reflection;
6 using ITU.DAL;
7
8 namespace ITU.DAL.Factory
9 {
10     public class DataAccessProviderFactory : IDataProviderFactory
11     {
12         private static Assembly activeDataProvider = null;
13         private static IDataProviderFactory activeDataProviderFactory = null;
14         private static ICustomerDataProvider customerDataProvider = null;
15         private static IOrderDataProvider orderDataProvider = null;
16         private static IProductDataProvider productDataProvider = null;
17
18         static DataAccessProviderFactory()
19         {
20             string providerName = ConfigurationSettings.AppSettings["DataProvider"];
21             string providerFactoryName = ConfigurationSettings["
22 "DataProviderFactory"];
23             activeDataProvider = Assembly.Load(providerName);
24             activeDataProviderFactory = (IDataProviderFactory)activeDataProvider.
25 CreateInstance(providerFactoryName);
26
27             ICustomerDataProvider IDataProviderFactory.GetCustomerProvider()
28         {
29             if (customerDataProvider != null)
30                 return customerDataProvider;
31             else
32             {
33                 customerDataProvider = activeDataProviderFactory.GetCustomerProvider();
34             }
35             return customerDataProvider;
36         }
37
38         IOrderDataProvider IDataProviderFactory.GetOrderProvider()
39         {
40             if (orderDataProvider != null)
41                 return orderDataProvider;
42             else
43             {
44                 orderDataProvider = activeDataProviderFactory.GetOrderProvider();
45                 return orderDataProvider;
46             }
47         }
48
49         IProductDataProvider IDataProviderFactory.GetProductProvider()
50         {
51             if (productDataProvider != null)
52                 return productDataProvider;
53             else
54             {
55                 productDataProvider = activeDataProviderFactory.GetProductProvider();
56                 return productDataProvider;
57             }
58         }
59     }
60 }
```

**ITU.DAL.SqlServer**

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.DAL;
5
6 namespace ITU.DAL.SqlServer
7 {
8     class DataProviderFactory : IDataProviderFactory
9     {
10
11         public ICustomerDataProvider GetCustomerProvider()
12         {
13             return new CustomerDataProvider();
14         }
15
16         public IOrderDataProvider GetOrderProvider()
17         {
18             return new OrderDataProvider();
19         }
20
21         public IProductDataProvider GetProductProvider()
22         {
23             return new ProductDataProvider();
24         }
25     }
26 }
27
```

```
1 using System;
2 using System.Collections;
3 using System.Collections.Specialized;
4 using System.Configuration;
5
6 namespace ITU.DAL.SqlServer
7 {
8     internal class DataProviderHelper
9     {
10         private static string ConfigNode = "ITU.DAL.SqlServer";
11         private static string connectionString = string.Empty;
12
13         static DataProviderHelper()
14         {
15             NameValueCollection values = (NameValueCollection) ConfigurationSettings.
16             GetConfig(ConfigNode);
17             connectionString = values["ConnectionString"];
18         }
19         public static string ConnectionString
20         {
21             get { return connectionString; }
22         }
23     }
24 }
25
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.DAL;
5 using ITU.BusinessEntities;
6 using System.Data.SqlClient;
7 using System.Data;
8
9 namespace ITU.DAL.SqlServer
10 {
11     public class OrderDataProvider : IOrderDataProvider
12     {
13         private static SqlConnection myConnection = new SqlConnection
14             (DataProviderHelper.ConnectionString);
15
16         public void Create(Order o)
17         {
18         }
19
20         public void Update(Order o)
21         {
22         }
23
24         public void Delete(Order o)
25         {
26         }
27
28         public Order GetById(string orderId)
29         {
30             return new Order();
31         }
32
33         public DataSet GetOrdersDataSet()
34         {
35             return new DataSet();
36         }
37
38         public void UpdateOrdersDataSet(DataSet ds)
39         {
40         }
41
42         public IList<OrderDetail> GetOrderDetails(string orderId)
43         {
44             IList<OrderDetail> orderDetails = new List<OrderDetail>();
45
46             try
47             {
48                 string query = String.Format("SELECT OrderID, ProductID, UnitPrice,
49 Quantity, Discount FROM \"Order Details\" WHERE OrderID='{0}'", orderId);
50                 SqlCommand myCommand = new SqlCommand(query, myConnection);
51
52                 myConnection.Open();
53
54                 SqlDataReader myReader;
55                 myReader = myCommand.ExecuteReader();
56                 // Always call Read before accessing data.
57
58                 while (myReader.Read())
59                 {
60                     orderDetails.Add(new OrderDetail(
61                         myReader["OrderID"].ToString(),
62                         myReader["ProductID"].ToString(),
63                         myReader["UnitPrice"].ToString(),
64                         myReader["Quantity"].ToString(),
65                         myReader["Discount"].ToString()
66                     ));
67                 }
68                 // always call Close when done reading.
69                 myReader.Close();
70                 // Close the connection when done with it.
71                 myConnection.Close();
72             }
73             finally
74             {
75             }
76         }
77     }
78 }
```

```
72         {
73             myConnection.Close();
74         }
75     }
76     return orderDetails;
77 }
78
79
80
81
82
83     public IList<Order> GetOrders(string customerId)
84     {
85         IList<Order> orders = new List<Order>();
86
87         try
88         {
89             string query = String.Format("SELECT OrderID, CustomerID, EmployeeID,
OrderDate, RequiredDate, ShippedDate, ShipVia, Freight, ShipName, ShipAddress,
ShipCity, ShipRegion FROM Orders WHERE CustomerID='{0}'", customerId);
90             SqlCommand myCommand = new SqlCommand(query, myConnection);
91
92             myConnection.Open();
93
94             SqlDataReader myReader;
95             myReader = myCommand.ExecuteReader();
96             // Always call Read before accessing data.
97
98             while (myReader.Read())
99             {
100                 orders.Add(new Order(
101                     myReader["OrderID"].ToString(),
102                     myReader["CustomerID"].ToString(),
103                     myReader["EmployeeID"].ToString(),
104                     myReader["OrderDate"].ToString(),
105                     myReader["RequiredDate"].ToString(),
106                     myReader["ShippedDate"].ToString(),
107                     myReader["ShipVia"].ToString(),
108                     myReader["Freight"].ToString(),
109                     myReader["ShipName"].ToString(),
110                     myReader["ShipAddress"].ToString(),
111                     myReader["ShipCity"].ToString(),
112                     myReader["ShipRegion"].ToString()
113                 ));
114             }
115             // always call Close when done reading.
116             myReader.Close();
117             // Close the connection when done with it.
118             myConnection.Close();
119         }
120         finally
121         {
122             myConnection.Close();
123         }
124     }
125     return orders;
126 }
127 }
128 }
129 }
```



```
67         myReader["city"].ToString(),
68         myReader["region"].ToString(),
69         myReader["postalCode"].ToString(),
70         myReader["country"].ToString(),
71         myReader["phone"].ToString(),
72         myReader["fax"].ToString()
73     ));
74     }
75     // always call Close when done reading.
76     myReader.Close();
77     // Close the connection when done with it.
78     conn.Close();
79 }
80 finally
81 {
82     conn.Close();
83 }
84
85     return customers;
86 }
87
88     public DataSet GetCustomersDataSet()
89     {
90         string query = "SELECT * FROM Customers";
91         SqlDataAdapter sda = new SqlDataAdapter(query, conn);
92         DataSet ds = new DataSet();
93         sda.Fill(ds, "Customers");
94
95         return ds;
96     }
97
98     public void UpdateCustomersDataSet(DataSet ds)
99     {
100         //string query = "SELECT * FROM Customers";
101         //SqlDataAdapter sda = new SqlDataAdapter(query, myConnection);
102         //SqlCommandBuilder scb = new SqlCommandBuilder(sda);
103         //myConnection.Open();
104         //sda.Update(ds, "Customers");
105         //myConnection.Close();
106
107         //SqlDataAdapter custDA = new SqlDataAdapter("SELECT CustomerID,
108         CompanyName FROM Customers ORDER BY CustomerID", myConnection);
109
110         // The Update command checks for optimistic concurrency violations in
111         the WHERE clause.
112         //custDA.UpdateCommand = new SqlCommand("UPDATE Customers (CustomerID,
113         CompanyName) VALUES(@CustomerID, @CompanyName) " +
114         "WHERE CustomerID = @oldCustomerID
115         AND CompanyName = @oldCompanyName", myConnection);
116         //custDA.UpdateCommand.Parameters.Add("@CustomerID", SqlDbType.NChar, 5,
117         "CustomerID");
118         //custDA.UpdateCommand.Parameters.Add("@CompanyName", SqlDbType.NVarChar,
119         30, "CompanyName");
120
121         // Pass the original values to the WHERE clause parameters.
122         //SqlParameter myParm;
123         //myParm = custDA.UpdateCommand.Parameters.Add("@oldCustomerID",
124         SqlDbType.NChar, 5, "CustomerID");
125         //myParm.SourceVersion = DataRowVersion.Original;
126         //myParm = custDA.UpdateCommand.Parameters.Add("@oldCompanyName",
127         SqlDbType.NVarChar, 30, "CompanyName");
128         //myParm.SourceVersion = DataRowVersion.Original;
129
130         // Add the RowUpdated event handler.
131         //custDA.RowUpdated += new SqlRowUpdatedEventHandler(OnRowUpdated);
132
133         //DataSet custDS = new DataSet();
134         //custDA.Fill(ds, "Customers");
135
136         // Modify the DataSet contents.
137         //custDA.Update(ds, "Customers");
138
139         //foreach (DataRow myRow in ds.Tables["Customers"].Rows)
```

```
132         //{
133         //     if(myRow.HasErrors)
134         //         throw new Exception(myRow[0] + "\n" + myRow.RowError);
135         //}
136
137         //The Adapter
138         SqlDataAdapter custDA = new SqlDataAdapter();
139
140         //The Insert commands
141         custDA.InsertCommand = new SqlCommand("INSERT INTO Customers (CustomerID,
142         CompanyName) Values(@CustomerID, @CompanyName)", conn);
143         custDA.InsertCommand.Parameters.Add("@CustomerID", SqlDbType.NChar, 5,
144         "CustomerID");
145         custDA.InsertCommand.Parameters.Add("@CompanyName", SqlDbType.NChar, 15,
146         "CompanyName");
147
148         //The Update commands
149         custDA.UpdateCommand = new SqlCommand("UPDATE Customers Set CustomerID =
150         @CustomerID, CompanyName = @CompanyName WHERE CustomerID = @OldCustomerID", conn)
151         ;
152         custDA.UpdateCommand.Parameters.Add("@CustomerID", SqlDbType.NChar, 5,
153         "CustomerID");
154         custDA.UpdateCommand.Parameters.Add("@CompanyName", SqlDbType.NChar, 15,
155         "CompanyName");
156         SqlParameter myParm = custDA.UpdateCommand.Parameters.Add("@OldCustomerID",
157         SqlDbType.NChar, 5, "CustomerID");
158         myParm.SourceVersion = DataRowVersion.Original;
159
160         //The Delete commands
161         custDA.DeleteCommand = new SqlCommand("DELETE FROM Customers WHERE
162         CustomerID = @CustomerID", conn);
163         myParm = custDA.DeleteCommand.Parameters.Add("@CustomerID", SqlDbType.
164         NChar, 5, "CustomerID");
165         myParm.SourceVersion = DataRowVersion.Original;
166
167         //Do the Update
168         custDA.Update(ds, "Customers");
169     }
170
171     protected static void OnRowUpdated(object sender, SqlRowUpdatedEventArgs
172     args)
173     {
174         if (args.RecordsAffected == 0)
175         {
176             args.Row.RowError = "Optimistic Concurrency Violation Encountered";
177             args.Status = UpdateStatus.SkipCurrentRow;
178         }
179     }
180
181     public Customer GetById(string customerId)
182     {
183         Customer customer = null;
184
185         string query = String.Format("SELECT customerID, companyName, contactName
186         , contactTitle, address, city, region, postalCode, country, phone, fax FROM
187         Customers WHERE CustomerID = '{0}'", customerId);
188
189         SqlCommand myCommand = new SqlCommand(query, conn);
190
191         conn.Open();
192
193         SqlDataReader myReader = myCommand.ExecuteReader();
194         while (myReader.Read())
195         {
196             customer = new Customer
197             (
198                 myReader["customerID"].ToString(),
199                 myReader["companyName"].ToString(),
200                 myReader["contactName"].ToString(),
201                 myReader["contactTitle"].ToString(),
202                 myReader["address"].ToString(),
203                 myReader["city"].ToString(),
```

```
192         myReader["region"].ToString(),
193         myReader["postalCode"].ToString(),
194         myReader["country"].ToString(),
195         myReader["phone"].ToString(),
196         myReader["fax"].ToString()
197     );
198     }
199
200     conn.Close();
201
202     return customer;
203 }
204
205 protected virtual SqlParameter[] GetVariablesByEntity(Customer item)
206 {
207     SqlParameter customerIdParameter = new SqlParameter("@CustomerId",
208     SqlDbType.NChar, 5);
209     customerIdParameter.Value = item.CustomerID;
210     SqlParameter companyNameParameter = new SqlParameter("@CompanyName",
211     SqlDbType.NVarChar, 40);
212     companyNameParameter.Value = item.CompanyName;
213     SqlParameter contactNameParameter = new SqlParameter("@ContactName",
214     SqlDbType.NVarChar, 30);
215     contactNameParameter.Value = item.ContactName;
216     SqlParameter contactTitleParameter = new SqlParameter("@ContactTitle",
217     SqlDbType.NVarChar, 30);
218     contactTitleParameter.Value = item.ContactTitle;
219     SqlParameter addressParameter = new SqlParameter("@Address", SqlDbType.
220     NVarChar, 60);
221     addressParameter.Value = item.Address;
222     SqlParameter cityParameter = new SqlParameter("@City", SqlDbType.NVarChar
223     , 15);
224     cityParameter.Value = item.City;
225     SqlParameter regionParameter = new SqlParameter("@Region", SqlDbType.
226     NVarChar, 15);
227     regionParameter.Value = item.Region;
228     SqlParameter postalCodeParameter = new SqlParameter("@PostalCode",
229     SqlDbType.NVarChar, 10);
230     postalCodeParameter.Value = item.PostalCode;
231     SqlParameter countryParameter = new SqlParameter("@Country", SqlDbType.
232     NVarChar, 15);
233     countryParameter.Value = item.Country;
234     SqlParameter phoneParameter = new SqlParameter("@Phone", SqlDbType.
235     NVarChar, 24);
236     phoneParameter.Value = item.Phone;
237     SqlParameter faxParameter = new SqlParameter("@Fax", SqlDbType.NVarChar,
238     24);
239     faxParameter.Value = item.Fax;
240
241     return new SqlParameter[]{
242         customerIdParameter,
243         companyNameParameter,
244         contactNameParameter,
245         contactTitleParameter,
246         addressParameter,
247         cityParameter,
248         regionParameter,
249         postalCodeParameter,
250         countryParameter,
251         phoneParameter,
252         faxParameter
253     };
254 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.DAL;
5 using ITU.BusinessEntities;
6 using System.Data.SqlClient;
7
8 namespace ITU.DAL.SqlServer
9 {
10     public class ProductDataProvider : IProductDataProvider
11     {
12         private static SqlConnection myConnection = new SqlConnection
(DataProviderHelper.ConnectionString);
13
14         public Product GetById(string productId)
15         {
16             Product p = null;
17
18             string query = String.Format("SELECT ProductID, ProductName, SupplierID,
CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel,
Discontinued FROM products WHERE ProductID = '{0}'", productId);
19             SqlCommand myCommand = new SqlCommand(query, myConnection);
20
21             myConnection.Open();
22
23             SqlDataReader myReader;
24             myReader = myCommand.ExecuteReader();
25
26             try
27             {
28                 while (myReader.Read())
29                 {
30                     p = new Product(
31                         myReader["ProductID"].ToString(),
32                         myReader["ProductName"].ToString(),
33                         myReader["SupplierID"].ToString(),
34                         myReader["CategoryID"].ToString(),
35                         myReader["QuantityPerUnit"].ToString(),
36                         myReader["UnitPrice"].ToString(),
37                         myReader["UnitsInStock"].ToString(),
38                         myReader["UnitsOnOrder"].ToString(),
39                         myReader["ReorderLevel"].ToString(),
40                         myReader["Discontinued"].ToString()
41                     );
42                 }
43                 // always call Close when done reading.
44                 myReader.Close();
45                 // Close the connection when done with it.
46                 myConnection.Close();
47             }
48             finally
49             {
50                 myConnection.Close();
51             }
52             return p;
53         }
54
55         public IList<Product> GetProducts(string[] productId)
56         {
57             IList<Product> products = new List<Product>();
58
59             try
60             {
61                 string query = String.Format("SELECT ProductID, ProductName,
SupplierID, CategoryID, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder,
ReorderLevel, Discontinued FROM products WHERE ProductID IN ('{0}']", productId);
62                 SqlCommand myCommand = new SqlCommand(query, myConnection);
63
64                 myConnection.Open();
65
66                 SqlDataReader myReader;
67                 myReader = myCommand.ExecuteReader();
68
```

```
69         // Always call Read before accessing data.
70
71         while (myReader.Read())
72         {
73             products.Add(new Product(
74                 myReader["ProductID"].ToString(),
75                 myReader["ProductName"].ToString(),
76                 myReader["SupplierID"].ToString(),
77                 myReader["CategoryID"].ToString(),
78                 myReader["QuantityPerUnit"].ToString(),
79                 myReader["UnitPrice"].ToString(),
80                 myReader["UnitsInStock"].ToString(),
81                 myReader["UnitsOnOrder"].ToString(),
82                 myReader["ReorderLevel"].ToString(),
83                 myReader["Discontinued"].ToString()
84             ));
85         }
86         // always call Close when done reading.
87         myReader.Close();
88         // Close the connection when done with it.
89         myConnection.Close();
90     }
91     finally
92     {
93         myConnection.Close();
94     }
95     return products;
96 }
97 }
98 }
99 }
```

**ITU.DAL.Hibernate**

```
1 using System;
2 using ITU.DAL;
3
4 namespace ITU.DAL.Hibernate
5 {
6     class DataProviderFactory : IDataProviderFactory
7     {
8
9         public ICustomerDataProvider GetCustomerProvider()
10        {
11            return new CustomerDataProvider();
12        }
13
14        public IOrderDataProvider GetOrderProvider()
15        {
16            return new OrderDataProvider();
17        }
18
19        public IProductDataProvider GetProductProvider()
20        {
21            return new ProductDataProvider();
22        }
23    }
24 }
25
```

```
1 using System;
2 using NHibernate;
3 using NHibernate.Cfg;
4
5 namespace ITU.DAL.Hibernate
6 {
7     internal class HibernateHelper
8     {
9         private static readonly Configuration config;
10        private static readonly ISessionFactory sessionFactory;
11
12        static HibernateHelper()
13        {
14            config = new Configuration();
15            config.AddAssembly("ITU.DAL.Hibernate");
16
17            sessionFactory = config.BuildSessionFactory();
18        }
19
20        public static Configuration GetConfig()
21        {
22            return config;
23        }
24
25        public static ISessionFactory GetSessionFactory()
26        {
27            return sessionFactory;
28        }
29    }
30 }
31
```

```
1 using NHibernate;
2
3 namespace ITU.DAL.Hibernate
4 {
5     public abstract class CommonDataProvider
6     {
7
8         protected ISession GetNewSession()
9         {
10            return HibernateHelper.GetSessionFactory().OpenSession();
11        }
12
13        protected void EndSession(ISession session)
14        {
15            session.Flush();
16            session.Close();
17        }
18    }
19 }
20
```

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
3   <class name="ITU.BusinessEntities.Customer, ITU.BusinessEntities" table="Customers">
4     <id name="CustomerID" column="CustomerId" type="String(5)">
5       <generator class="assigned" />
6     </id>
7     <property name="Address" column="Address" type="String(60)" not-null="false" />
8     <property name="City" column="City" type="String(15)" not-null="false" />
9     <property name="CompanyName" column="CompanyName" type="String(40)" not-null="true" ✎
10    " />
11    <property name="ContactName" column="ContactName" type="String(30)" not-null= ✎
12    "false" />
13    <property name="ContactTitle" column="ContactTitle" type="String(30)" not-null= ✎
14    "false" />
15    <property name="Country" column="Country" type="String(15)" not-null="false" />
16    <property name="Fax" column="Fax" type="String(24)" not-null="false" />
17    <property name="Phone" column="Phone" type="String(24)" not-null="false" />
18    <property name="PostalCode" column="PostalCode" type="String(10)" not-null="false" ✎
19    />
20    <property name="Region" column="Region" type="String(15)" not-null="false" />
21  </class>
22 </hibernate-mapping>
```

```
1 using System;
2 using ITU.DAL;
3 using ITU.BusinessEntities;
4 using NHibernate;
5 using System.Data;
6
7 namespace ITU.DAL.Hibernate
8 {
9     public class CustomerDataProvider : CommonDataProvider, ICustomerDataProvider
10    {
11
12        public void Create(Customer c)
13        {
14            ISession session = GetNewSession();
15            session.Save(c);
16            EndSession(session);
17        }
18
19        public void Update(Customer c)
20        {
21            ISession session = GetNewSession();
22            session.Update(c);
23            EndSession(session);
24        }
25
26        public void Delete(Customer c)
27        {
28            ISession session = GetNewSession();
29            session.Delete(c);
30            EndSession(session);
31        }
32
33        public Customer GetById(string customerId)
34        {
35            ISession session = GetNewSession();
36            Customer c = (Customer) session.Get(typeof(Customer), customerId);
37            EndSession(session);
38            return c;
39        }
40
41        public System.Collections.Generic.IList<Customer> GetCustomers()
42        {
43            ISession session = GetNewSession();
44            System.Collections.Generic.IList<Customer> customers = new System.
Collections.Generic.List<Customer>();
45            System.Collections.IList l = session.CreateCriteria(typeof(Customer)).List
();
46            System.Collections.IEnumerator e = l.GetEnumerator();
47            while(e.MoveNext())
48            {
49                customers.Add((Customer) e.Current);
50            }
51            EndSession(session);
52            return customers;
53        }
54
55        /// <summary>
56        /// Programmatically creates a simple dataset with just the CustomerID and
returns it
57        /// Just so that the WinClient won't crash and to show that it works with
Hibernate
58        /// </summary>
59        /// <returns></returns>
60        public System.Data.DataSet GetCustomersDataSet()
61        {
62            DataTable dt = new DataTable("Customers");
63            DataColumn dc = new DataColumn("CustomerID");
64            dc.Caption = "CustomerID";
65            dc.DataType = System.Type.GetType("System.String");
66
67            dt.Columns.Add(dc);
68
69            foreach (Customer c in GetCustomers())
```

```
70         {
71             DataRow dr = dt.NewRow();
72             dr["CustomerID"] = c.CustomerID.ToString();
73             dt.Rows.Add(dr);
74         }
75
76         DataSet ds = new DataSet();
77         ds.Tables.Add(dt);
78
79         return ds;
80     }
81
82     public void UpdateCustomersDataSet(System.Data.DataSet ds)
83     {
84         throw new NotImplementedException("The method or operation is not
85         implemented.");
86     }
87 }
88 }
89 }
```

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <hibernate-mapping xmlns="urn:hibernate-mapping-2.0">
3   <class name="ITU.BusinessEntities.Product, ITU.BusinessEntities" table="Products">
4     <id name="ProductID" column="ProductId" type="String(5)">
5       <generator class="assigned" />
6     </id>
7     <property name="ProductName" type="String(20)" not-null="false"></property>
8     <property name="SupplierID" type="String(20)" not-null="false"></property>
9     <property name="CategoryID" type="String(20)" not-null="false"></property>
10    <property name="QuantityPerUnit" type="String(20)" not-null="false"></property>
11    <property name="UnitPrice" type="String(20)" not-null="false"></property>
12    <property name="UnitsInStock" type="String(20)" not-null="false"></property>
13    <property name="UnitsOnOrder" type="String(20)" not-null="false"></property>
14    <property name="ReorderLevel" type="String(20)" not-null="false"></property>
15    <property name="Discontinued" type="String(20)" not-null="false"></property>
16
17   </class>
18 </hibernate-mapping>
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.DAL;
5 using ITU.BusinessEntities;
6 using NHibernate;
7
8 namespace ITU.DAL.Hibernate
9 {
10     class ProductDataProvider : CommonDataProvider, IProductDataProvider
11     {
12
13         Product IProductDataProvider.GetById(string productId)
14         {
15             ISession session = GetNewSession();
16             Product p = (Product)session.Get(typeof(Product), productId);
17             EndSession(session);
18             return p;
19         }
20
21         public IList<Product> GetProducts(string[] productId)
22         {
23             throw new NotImplementedException("The method or operation is not
24             implemented.");
25         }
26     }
27 }
```

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <hibernate-mapping xmlns="urn:hibernate-mapping-2.0">
3   <class name="ITU.BusinessEntities.Order, ITU.BusinessEntities" table="Orders">
4     <id name="OrderID" column="OrderID" type="String(5)">
5       <generator class="assigned" />
6     </id>
7     <property name="CustomerID" type="String(20)" not-null="false"></property>
8     <property name="EmployeeID" type="String(20)" not-null="false"></property>
9     <property name="OrderDate" type="String(20)" not-null="false"></property>
10    <property name="RequiredDate" type="String(20)" not-null="false"></property>
11    <property name="ShippedDate" type="String(20)" not-null="false"></property>
12    <property name="ShipVia" type="String(20)" not-null="false"></property>
13    <property name="Freight" type="String(20)" not-null="false"></property>
14    <property name="ShipName" type="String(20)" not-null="false"></property>
15    <property name="ShipAddress" type="String(20)" not-null="false"></property>
16    <property name="ShipCity" type="String(20)" not-null="false"></property>
17    <property name="ShipRegion" type="String(20)" not-null="false"></property>
18  </class>
19 </hibernate-mapping>
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.DAL;
5 using ITU.BusinessEntities;
6 using NHibernate;
7
8 namespace ITU.DAL.Hibernate
9 {
10     class OrderDataProvider : CommonDataProvider, IOrderDataProvider
11     {
12
13         public void Create(Order o)
14         {
15             ISession session = GetNewSession();
16             session.Save(o);
17             EndSession(session);
18         }
19
20         public void Update(Order o)
21         {
22             ISession session = GetNewSession();
23             session.Update(o);
24             EndSession(session);
25         }
26
27         public void Delete(Order o)
28         {
29             ISession session = GetNewSession();
30             session.Delete(o);
31             EndSession(session);
32         }
33
34         public Order GetById(string orderId)
35         {
36             ISession session = GetNewSession();
37             Order o = (Order)session.Get(typeof(Order), orderId);
38             EndSession(session);
39             return o;
40         }
41
42         public IList<Order> GetOrders(string customerId)
43         {
44             ISession session = GetNewSession();
45             System.Collections.Generic.IList<Order> orders = new System.Collections.
Generic.List<Order>();
46             ICriteria criteria = session.CreateCriteria(typeof(Order))
47                 .Add(NHibernate.Expression.Expression.Eq(
"CustomerID", customerId));
48
49             System.Collections.IList l = criteria.List();
50
51             System.Collections.IEnumerator e = l.GetEnumerator();
52             while (e.MoveNext())
53             {
54                 orders.Add((Order)e.Current);
55             }
56             EndSession(session);
57             return orders;
58         }
59
60         public System.Data.DataSet GetOrdersDataSet()
61         {
62             throw new NotImplementedException("The method or operation is not
implemented.");
63         }
64
65         public void UpdateOrdersDataSet(System.Data.DataSet ds)
66         {
67             throw new NotImplementedException("The method or operation is not
implemented.");
68         }
69     }

```

```
70     public IList<OrderDetail> GetOrderDetails(string orderId)
71     {
72         ISession session = GetNewSession();
73         System.Collections.Generic.IList<OrderDetail> details = new System.
Collections.Generic.List<OrderDetail>();
74         ICriteria criteria = session.CreateCriteria(typeof(OrderDetail))
75             .Add(NHibernate.Expression.Expression.Eq(
"OrderID", orderId));
76
77         //IQuery q = session.CreateSQLQuery("SELECT OrderID, ProductID, UnitPrice,
Quantity, Discount FROM \"Order Details\" WHERE OrderID='" + orderId + "'",
"OrderDetail", typeof(OrderDetail));
78         //System.Collections.IList l = q.List();
79
80         System.Collections.IList l = criteria.List();
81
82         System.Collections.IEnumerator e = l.GetEnumerator();
83         while (e.MoveNext())
84         {
85             details.Add((OrderDetail)e.Current);
86         }
87         EndSession(session);
88         return details;
89     }
90 }
91 }
92 }
```

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
3   <class name="ITU.BusinessEntities.OrderDetail, ITU.BusinessEntities" table="`Order
  Details`">
4     <id name="OrderID" column="OrderID" type="String(5)">
5       <generator class="assigned" />
6     </id>
7     <property name="ProductID" type="String(20)" not-null="false"></property>
8     <property name="UnitPrice" type="String(20)" not-null="false"></property>
9     <property name="Quantity" type="String(20)" not-null="false"></property>
10    <property name="Discount" type="String(20)" not-null="false"></property>
11  </class>
12 </hibernate-mapping>
```

**ITU.BLL**

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.Data;
5 using ITU.DAL;
6 using ITU.DAL.Factory;
7 using ITU.BusinessEntities;
8
9 namespace ITU.BLL
10 {
11     public class CustomerManager
12     {
13         private ICustomerDataProvider dataProvider;
14
15         public CustomerManager()
16         {
17             dataProvider = ((IDataProviderFactory) new DataAccessProviderFactory()).
18             GetCustomerProvider();
19
20         public IList<Customer> GetAllCustomers()
21         {
22             return dataProvider.GetCustomers();
23         }
24
25         public Customer GetById(string customerId)
26         {
27             if (customerId == null)
28                 throw new ArgumentNullException("The customer Id cannot be null");
29             else if (customerId.Length == 0)
30                 throw new ArgumentException("The customer Id cannot be blank");
31             return dataProvider.GetById(customerId);
32         }
33
34         public void Create(Customer c)
35         {
36             if (c == null)
37                 throw new ArgumentNullException("item", "The specified customer cannot
38 be null");
39             else
40                 dataProvider.Create(c);
41         }
42
43         public void Update(Customer c)
44         {
45             if (c == null)
46                 throw new ArgumentNullException("item", "The specified customer cannot
47 be null");
48             else
49                 dataProvider.Update(c);
50         }
51
52         public void Delete(Customer c)
53         {
54             if (c == null)
55                 throw new ArgumentNullException("item", "The specified customer cannot
56 be null");
57             else
58                 dataProvider.Delete(c);
59         }
60
61         public DataSet GetCustomersDataSet()
62         {
63             return dataProvider.GetCustomersDataSet();
64         }
65
66         public void UpdateCustomersDataSet(DataSet ds)
67         {
68             dataProvider.UpdateCustomersDataSet(ds);
69         }
70     }
71 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.DAL;
5 using ITU.DAL.Factory;
6 using ITU.BusinessEntities;
7
8 namespace ITU.BLL
9 {
10     public class ProductManager
11     {
12         private IProductDataProvider dataProvider;
13
14         public ProductManager()
15         {
16             dataProvider = ((IDataProviderFactory)new DataAccessProviderFactory()).
17             GetProductProvider();
18
19             public Product GetById(string productId)
20             {
21                 return dataProvider.GetById(productId);
22             }
23     }
24 }
25
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4 using ITU.DAL;
5 using ITU.DAL.Factory;
6 using ITU.BusinessEntities;
7
8 namespace ITU.BLL
9 {
10     public class OrderManager
11     {
12         private IOrderDataProvider dataProvider;
13
14         public OrderManager()
15         {
16             dataProvider = ((IDataProviderFactory)new DataAccessProviderFactory()).
17             GetOrderProvider();
18
19             public IList<Order> GetCustomerOrders(string customerId)
20             {
21                 return dataProvider.GetOrders(customerId);
22             }
23
24             public IList<OrderDetail> GetOrderDetails(string orderId)
25             {
26                 return dataProvider.GetOrderDetails(orderId);
27             }
28     }
29 }
30
```

**Website**

```
1 <%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
  Inherits="masterpages_MasterPage" %>
2
3 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
  xhtml1/DTD/xhtml1-transitional.dtd">
4
5 <html xmlns="http://www.w3.org/1999/xhtml" >
6 <head runat="server">
7   <title>Untitled Page</title>
8 </head>
9 <body>
10  <form id="form1" runat="server">
11  <div>
12    &nbsp;<asp:SiteMapPath ID="SiteMapPath1" runat="server" Font-Names="Verdana"
  Font-Size="0.8em"
13      PathSeparator=" : ">
14      <PathSeparatorStyle Font-Bold="True" ForeColor="#507CD1" />
15      <CurrentNodeStyle ForeColor="#333333" />
16      <NodeStyle Font-Bold="True" ForeColor="#284E98" />
17      <RootNodeStyle Font-Bold="True" ForeColor="#507CD1" />
18    </asp:SiteMapPath>
19    <table style="width: 800px; height: 497px">
20      <tr>
21        <td style="width: 100px; height: 500px" valign="top">
22          <asp:TreeView ID="TreeView1" runat="server" DataSourceID=
  "SiteMapDataSource1" ImageSet="Simple"
23            ShowLines="True">
24            <ParentNodeStyle Font-Bold="False" />
25            <HoverNodeStyle Font-Underline="True" ForeColor="#5555DD" />
26            <SelectedNodeStyle Font-Underline="True" ForeColor="#5555DD"
  HorizontalPadding="0px"
27              VerticalPadding="0px" />
28            <NodeStyle Font-Names="Tahoma" Font-Size="10pt" ForeColor=
  "Black" HorizontalPadding="0px"
29              NodeSpacing="0px" VerticalPadding="0px" />
30          </asp:TreeView>
31          <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
32        </td>
33        <td style="width: 100px" valign="top">
34          <asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
35            </asp:ContentPlaceHolder>
36        </td>
37      </tr>
38    </table>
39  </div>
40 </form>
41 </body>
42 </html>
43
```

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
3   <siteMapNode url="Default.aspx" title="Menu" description="Main Menu">
4     <siteMapNode url="pages/CustomerList.aspx" title="Customer list" description=
      "List of all customers" />
5   </siteMapNode>
6 </siteMap>
7
```

```
1 using System;
2 using System.Web;
3 using System.Web.Services;
4 using System.Web.Services.Protocols;
5 using System.Collections.Generic;
6 using System.Data;
7 using ITU.BusinessEntities;
8 using ITU.DAL;
9 using ITU.BLL;
10 using System.Threading;
11
12 [WebService(Namespace = "http://tempuri.org/")]
13 [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
14 public class Service : System.Web.Services.WebService
15 {
16     CustomerManager cManager;
17     OrderManager oManager;
18     ProductManager pManager;
19
20     public Service ()
21     {
22
23         //Uncomment the following line if using designed components
24         //InitializeComponent();
25         cManager = new CustomerManager();
26         oManager = new OrderManager();
27         pManager = new ProductManager();
28     }
29
30     [WebMethod]
31     public Product GetProduct(string productId)
32     {
33         return pManager.GetById(productId);
34     }
35
36     [WebMethod]
37     public List<OrderDetail> GetOrderDetails(string orderId)
38     {
39         return (List<OrderDetail>)oManager.GetOrderDetails(orderId);
40     }
41
42     [WebMethod]
43     public List<Order> GetCustomerOrders(string customerId)
44     {
45         return (List<Order>)oManager.GetCustomerOrders(customerId);
46     }
47
48     [WebMethod]
49     public List<Customer> GetAllCustomers()
50     {
51         Thread tr = new Thread(doSomething);
52         tr.Start();
53         lock (tr)
54         {
55             Thread.Sleep(1000);
56         }
57         return (List<Customer>)cManager.GetAllCustomers();
58     }
59
60     [WebMethod]
61     public void Update(Customer c)
62     {
63         cManager.Update(c);
64     }
65
66     [WebMethod]
67     public void Create(Customer c)
68     {
69         cManager.Create(c);
70     }
71
72     [WebMethod]
73     public void Delete(Customer c)
```

```
74     {
75         cManager.Delete(c);
76     }
77
78     [WebMethod]
79     public Customer GetCustomerById(string customerId)
80     {
81         return cManager.GetById(customerId);
82     }
83
84
85     [WebMethod]
86     public DataSet GetCustomersDataSet()
87     {
88         Thread tr = new Thread(doSomething);
89         tr.Start();
90         lock (tr)
91         {
92             Thread.Sleep(1000);
93         }
94
95         return cManager.GetCustomersDataSet();
96     }
97
98     public void doSomething()
99     {
100
101     }
102
103     [WebMethod]
104     public void UpdateCustomersDataSet(DataSet ds)
105     {
106         cManager.UpdateCustomersDataSet(ds);
107     }
108
109 }
110
```

```
1 <%@ Page Language="C#" MasterPageFile="~/masterpages/MasterPage.master" AutoEventWireup
  = "true" CodeFile="OrderDetails.aspx.cs" Inherits="pages_OrderDetails" Title=
  "Untitled Page" %>
2 <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
3 </asp:Content>
4
5
```

```
1 <%@ Page Language="C#" MasterPageFile="~/masterpages/MasterPage.master"
  AutoEventWireup="true" CodeFile="CustomerOrders.aspx.cs" Inherits=
  "pages_CustomerOrders" Title="Untitled Page" %>
2 <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
3   <asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod=
  "GetCustomerOrders"
4     TypeName="ITU.BLL.OrderManager">
5     <SelectParameters>
6       <asp:QueryStringParameter Name="customerId" QueryStringField="CustomerID"
  Type="String" />
7     </SelectParameters>
8   </asp:ObjectDataSource>
9   <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
  CellPadding="4"
10     DataSourceID="ObjectDataSource1" Font-Size="Small" ForeColor="#333333"
  GridLines="None">
11     <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
12     <Columns>
13       <asp:HyperLinkField DataNavigateUrlFields="OrderID"
  DataNavigateUrlFormatString="OrderDetails.aspx?OrderID={0}"
14         Text="See Order Details" />
15       <asp:BoundField DataField="CustomerID" HeaderText="CustomerID"
  SortExpression="CustomerID" />
16       <asp:BoundField DataField="OrderID" HeaderText="OrderID" SortExpression=
  "OrderID" />
17       <asp:BoundField DataField="EmployeeID" HeaderText="EmployeeID"
  SortExpression="EmployeeID" />
18       <asp:BoundField DataField="OrderDate" HeaderText="OrderDate"
  SortExpression="OrderDate" />
19       <asp:BoundField DataField="RequiredDate" HeaderText="RequiredDate"
  SortExpression="RequiredDate" />
20       <asp:BoundField DataField="ShipAddress" HeaderText="ShipAddress"
  SortExpression="ShipAddress" />
21       <asp:BoundField DataField="ShipName" HeaderText="ShipName" SortExpression=
  "ShipName" />
22       <asp:BoundField DataField="ShippedDate" HeaderText="ShippedDate"
  SortExpression="ShippedDate" />
23       <asp:BoundField DataField="ShipCity" HeaderText="ShipCity" SortExpression=
  "ShipCity" />
24       <asp:BoundField DataField="Freight" HeaderText="Freight" SortExpression=
  "Freight" />
25       <asp:BoundField DataField="ShipRegion" HeaderText="ShipRegion"
  SortExpression="ShipRegion" />
26       <asp:BoundField DataField="ShipVia" HeaderText="ShipVia" SortExpression=
  "ShipVia" />
27     </Columns>
28     <RowStyle BackColor="#EFF3FB" />
29     <EditRowStyle BackColor="#2461BF" />
30     <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />
31     <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
32     <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
33     <AlternatingRowStyle BackColor="White" />
34   </asp:GridView>
35 </asp:Content>
36
37
```

```
1 <%@ Page Language="C#" MasterPageFile="~/masterpages/MasterPage.master"           ↵
  AutoEventWireup="true" CodeFile="CustomerList.aspx.cs" Inherits=             ↵
  "pages_CustomerList" Title="Untitled Page" %>
2 <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
3   <asp:ObjectDataSource ID="ObjectDataSource1" runat="server" DataObjectTypeName= ↵
  "ITU.BusinessEntities.Customer"
4     DeleteMethod="Delete" SelectMethod="GetAllCustomers" TypeName="ITU.BLL.    ↵
  CustomerManager">
5   </asp:ObjectDataSource>
6   <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"       ↵
  DataSourceID="ObjectDataSource1" AllowPaging="True" CellPadding="4" Font-Size= ↵
  "Small" ForeColor="#333333" GridLines="None">
7     <Columns>
8       <asp:HyperLinkField DataNavigateUrlFields="CustomerID"                   ↵
  DataNavigateUrlFormatString="CustomerDetail.aspx?CustomerID={0}"
9         Text="Edit" />
10      <asp:HyperLinkField DataNavigateUrlFields="CustomerID"                   ↵
  DataNavigateUrlFormatString="CustomerOrders.aspx?CustomerID={0}"
11        Text="See Orders" />
12      <asp:BoundField DataField="CustomerID" HeaderText="CustomerID"           ↵
  SortExpression="CustomerID" />
13      <asp:BoundField DataField="CompanyName" HeaderText="CompanyName"         ↵
  SortExpression="CompanyName" />
14      <asp:BoundField DataField="Address" HeaderText="Address" SortExpression=  ↵
  "Address" />
15      <asp:BoundField DataField="PostalCode" HeaderText="PostalCode"           ↵
  SortExpression="PostalCode" />
16      <asp:BoundField DataField="City" HeaderText="City" SortExpression="City"  ↵
  />
17      <asp:BoundField DataField="Country" HeaderText="Country" SortExpression=  ↵
  "Country" />
18      <asp:BoundField DataField="Phone" HeaderText="PhoneNumber" SortExpression= ↵
  "PhoneNumber" />
19      <asp:BoundField DataField="Fax" HeaderText="FaxNumber" SortExpression=   ↵
  "FaxNumber" />
20      <asp:BoundField DataField="ContactTitle" HeaderText="ContactTitle"       ↵
  SortExpression="ContactTitle" />
21      <asp:BoundField DataField="ContactName" HeaderText="ContactName"         ↵
  SortExpression="ContactName" />
22      <asp:BoundField DataField="Region" HeaderText="Region" SortExpression=   ↵
  "Region" />
23    </Columns>
24    <FooterStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
25    <RowStyle BackColor="#EFF3FB" />
26    <EditRowStyle BackColor="#2461BF" />
27    <SelectedRowStyle BackColor="#D1DDF1" Font-Bold="True" ForeColor="#333333" />
28    <PagerStyle BackColor="#2461BF" ForeColor="White" HorizontalAlign="Center" />
29    <HeaderStyle BackColor="#507CD1" Font-Bold="True" ForeColor="White" />
30    <AlternatingRowStyle BackColor="White" />
31  </asp:GridView>
32 </asp:Content>
33
34
```

```
1 <%@ Page Language="C#" MasterPageFile="~/masterpages/MasterPage.master"
  AutoEventWireup="true" CodeFile="CustomerDetail.aspx.cs" Inherits=
  "pages_CustomerDetail" Title="Untitled Page" %>
2 <asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
3   <table id="TABLE1" style="width: 369px; height: 249px">
4     <tr>
5       <td align="right" style="height: 20px" valign="top" width="150">
6         Customer ID:</td>
7       <td align="left" style="width: 100px; height: 20px" valign="top">
8         <asp:TextBox ID="tbx_CustomerID" runat="server"></asp:TextBox></td>
9     </tr>
10    <tr>
11      <td align="right" style="height: 25px" valign="top" width="150">
12        Company Name:</td>
13      <td align="left" style="width: 100px; height: 25px" valign="top">
14        <asp:TextBox ID="tbx_CompanyName" runat="server"></asp:TextBox></td>
15    </tr>
16    <tr>
17      <td align="right" style="height: 20px" valign="top" width="150">
18        Contact Name:</td>
19      <td align="left" style="width: 100px; height: 20px" valign="top">
20        <asp:TextBox ID="tbx_ContactName" runat="server"></asp:TextBox></td>
21    </tr>
22    <tr>
23      <td align="right" style="height: 20px" valign="top" width="150">
24        Address:</td>
25      <td align="left" style="width: 100px; height: 20px" valign="top">
26        <asp:TextBox ID="tbx_Address" runat="server"></asp:TextBox></td>
27    </tr>
28    <tr>
29      <td align="right" style="height: 20px" valign="top" width="150">
30        City:</td>
31      <td align="left" style="width: 100px; height: 20px" valign="top">
32        <asp:TextBox ID="tbx_City" runat="server"></asp:TextBox></td>
33    </tr>
34    <tr>
35      <td align="right" style="height: 20px" valign="top" width="150">
36        Region:</td>
37      <td align="left" style="width: 100px; height: 20px" valign="top">
38        <asp:TextBox ID="tbx_Region" runat="server"></asp:TextBox></td>
39    </tr>
40    <tr>
41      <td align="right" style="height: 20px" valign="top" width="150">
42        Postal Code:</td>
43      <td align="left" style="width: 100px; height: 20px" valign="top">
44        <asp:TextBox ID="tbx_PostalCode" runat="server"></asp:TextBox></td>
45    </tr>
46    <tr>
47      <td align="right" style="height: 20px" valign="top" width="150">
48        Country:</td>
49      <td align="left" style="width: 100px; height: 20px" valign="top">
50        <asp:TextBox ID="tbx_Country" runat="server"></asp:TextBox></td>
51    </tr>
52    <tr>
53      <td align="right" style="height: 20px" valign="top" width="150">
54        Phone:</td>
55      <td align="left" style="width: 100px; height: 20px" valign="top">
56        <asp:TextBox ID="tbx_Phone" runat="server"></asp:TextBox></td>
57    </tr>
58    <tr>
59      <td align="right" style="height: 20px" valign="top" width="150">
60        Fax:</td>
61      <td align="left" style="width: 100px; height: 20px" valign="top">
62        <asp:TextBox ID="tbx_Fax" runat="server"></asp:TextBox></td>
63    </tr>
64    <tr>
65      <td align="right" colspan="2" style="height: 5px" valign="top" width="150"
66    >
67      <asp:Button ID="btn_UpdateCustomer" runat="server" Text="Update
68      Customer" OnClick="btn_UpdateCustomer_Click" /></td>
69  </table>
</asp:Content>
```

```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Collections;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.WebControls;
9 using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11 using localhost;
12
13 public partial class pages_CustomerDetail : System.Web.UI.Page
14 {
15     localhost.Service wS;
16     Customer customer;
17
18     protected void Page_Load(object sender, EventArgs e)
19     {
20         if (!IsPostBack && ReqCustomerID != String.Empty)
21         {
22             wS = new localhost.Service();
23             customer = wS.GetCustomerById(ReqCustomerID);
24             this.ActiveCustomer = customer;
25         }
26         //Load the user interface with data for a given customer
27         this.DoUiSetup();
28     }
29
30
31     private void DoUiSetup()
32     {
33         tbx_CustomerID.Text = ActiveCustomer.CustomerID;
34         tbx_ContactName.Text = ActiveCustomer.ContactName;
35         tbx_Country.Text = ActiveCustomer.Country;
36         tbx_Address.Text = ActiveCustomer.Address;
37         tbx_City.Text = ActiveCustomer.City;
38         tbx_Fax.Text = ActiveCustomer.Fax;
39         tbx_Phone.Text = ActiveCustomer.Phone;
40         tbx_PostalCode.Text = ActiveCustomer.PostalCode;
41         tbx_Region.Text = ActiveCustomer.Region;
42     }
43
44     private string ReqCustomerID
45     {
46         get
47         {
48             return Request.QueryString["CustomerID"] != null ? Request.QueryString[
49 "CustomerID"] : String.Empty;
50         }
51     }
52     private Customer ActiveCustomer
53     {
54         get
55         {
56             return (Customer)Session["ThisCustomer"] !=null ? (Customer)Session[
57 "ThisCustomer"] : null;
58         }
59         set
60         {
61             Session["ThisCustomer"] = value;
62         }
63     }
64     protected void btn_UpdateCustomer_Click(object sender, EventArgs e)
65     {
66         string customerId = this.ReqCustomerID;
67         Customer customer = null;
68
69         if (customerId == string.Empty)
70             customer = new Customer();
71         else
```

```
72         customer = this.ActiveCustomer;
73
74         customer.CustomerID = tbx_CustomerID.Text;
75         customer.ContactName = tbx_ContactName.Text;
76         customer.Country = tbx_Country.Text;
77         customer.Address = tbx_Address.Text;
78         customer.City = tbx_City.Text;
79         customer.Fax = tbx_Fax.Text;
80         customer.Phone = tbx_Phone.Text;
81         customer.PostalCode = tbx_PostalCode.Text;
82         customer.Region = tbx_Region.Text;
83
84         new localhost.Service().Update(customer);
85     }
86 }
87
```

**ITU.WinClient**

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <configuration>
3   <configSections>
4     <sectionGroup name="userSettings" type="System.Configuration.UserSettingsGroup
      , System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >
5       <section name="ITU.DAL.SqlServer.App" type="System.Configuration.
      ClientSettingsSection, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=
      b77a5c561934e089" allowExeDefinition="MachineToLocalUser" requirePermission="false"
6     />
7     </sectionGroup>
8   </configSections>
9   <userSettings>
10    <ITU.DAL.SqlServer.App>
11      <setting name="myConnection" serializeAs="String">
12        <value>Data Source=(local)\SQLEXPRESS;Initial Catalog=Northwind;
      Integrated Security=True</value>
13      </setting>
14    </ITU.DAL.SqlServer.App>
15  </userSettings>
16 </configuration>
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Collections.ObjectModel;
4 using System.ComponentModel;
5 using System.Data;
6 using System.Drawing;
7 using System.Text;
8 using System.Windows.Forms;
9
10
11 namespace ITU.WinClient
12 {
13     public partial class Customers : Form
14     {
15         private localhost.Service ws;
16         private DataSet ds;
17         private localhost.Customer customer;
18
19         public Customers()
20         {
21             InitializeComponent();
22             waitStatus1.Hide();
23             ws = new localhost.Service();
24         }
25
26         /*
27          * Sync call to webservice, get all the customers in a DataSet.
28          * This will only work in ASP.NET 2.0
29          */
30         private void btn_GetAllCustomersSync_Click(object sender, EventArgs e)
31         {
32             waitStatus1.Show();
33             waitStatus1.Waiting(sender, e);
34             this.UpdateView();
35         }
36
37
38         /*
39          *
40          * Async call to webservice, get all the customers in a DataSet.
41          * This will only work in ASP.NET 2.0
42          *
43          */
44         private void btn_GetAllCustomersAsync_Click(object sender, EventArgs e)
45         {
46             waitStatus1.Show();
47             ws.GetCustomersDataSetCompleted += new localhost.
48             GetCustomersDataSetCompletedEventHandler(GetCustomersDataSetCompleted);
49             ws.GetCustomersDataSetAsync();
50         }
51         void GetCustomersDataSetCompleted(Object source, localhost.
52         GetCustomersDataSetCompletedEventArgs e)
53         {
54             this.UpdateView(e.Result);
55         }
56
57         /*
58          *
59          * Async call to webservice, update the customers in the DataSet.
60          * This will only work in ASP.NET 2.0
61          *
62          */
63         private void btn_UpdateAllCustomersAsync_Click(object sender, EventArgs e)
64         {
65             waitStatus1.Show();
66             ws.UpdateCustomersDataSetCompleted += new localhost.
67             UpdateCustomersDataSetCompletedEventHandler(UpdateCustomersDataSetCompleted);
68             ws.UpdateCustomersDataSetAsync(this.ds);
69         }
70         void UpdateCustomersDataSetCompleted(object sender, AsyncCompletedEventArgs e)
```

```
e)
71     {
72         this.btn_GetAllCustomersAsync_Click(sender, e);
73     }
74
75
76     /*
77     *
78     *     Async call to webservice, get all the customers in a Collection.
79     *     This will only work in ASP.NET 2.0
80     *
81     */
82     private void btn_CollectionCustomerAsync_Click(object sender, EventArgs e)
83     {
84         ws.GetAllCustomersCompleted += new localhost.
85         GetAllCustomersCompletedEventHandler(GetAllCustomersCompleted);
86         ws.GetAllCustomersAsync();
87     }
88     void GetAllCustomersCompleted(object sender, localhost.
89     GetAllCustomersCompletedEventArgs e)
90     {
91         localhost.Customer[] customers = e.Result;
92         this.UpdateView(customers);
93     }
94
95     // Update the dataset (synchronous call)
96     private void btn_UpdateAllCustomersSync_Click(object sender, EventArgs e)
97     {
98         waitStatus1.Show();
99         ws.UpdateCustomersDataSet(this.ds);
100        this.UpdateView();
101    }
102
103
104    // Use a dataset
105    private void UpdateView()
106    {
107        waitStatus1.Show();
108        this.ds = ws.GetCustomersDataSet();
109        dataGridView1.DataSource = ds;
110        dataGridView1.DataMember = "Customers";
111        waitStatus1.Hide();
112    }
113
114    // Use a dataset, that is at hand
115    private void UpdateView(DataSet ds)
116    {
117        waitStatus1.Show();
118        this.ds = ds;
119        dataGridView1.DataSource = ds;
120        dataGridView1.DataMember = "Customers";
121        waitStatus1.Hide();
122    }
123
124    // Use a collection
125    private void UpdateView(localhost.Customer[] customers)
126    {
127        BindingSource bs = new BindingSource();
128        BindingList<localhost.Customer> cList = new BindingList<localhost.
129        Customer>();
130
131        foreach (localhost.Customer c in customers)
132        {
133            cList.Add(c);
134        }
135
136        bs.DataSource = cList;
137        dataGridView1.DataSource = bs;
138
139        dataGridView1.Columns["CustomerID"].DisplayIndex = 0;
140        dataGridView1.Columns["CompanyName"].DisplayIndex = 1;
```

```
140         dataGridView1.Columns["ContactName"].DisplayIndex = 2;
141         dataGridView1.Columns["Address"].DisplayIndex = 3;
142         dataGridView1.Columns["City"].DisplayIndex = 4;
143         dataGridView1.Columns["Region"].DisplayIndex = 5;
144         dataGridView1.Columns["PostalCode"].DisplayIndex = 6;
145     }
146
147
148     /*
149     *         This is where the DataGridView showing the Orders get updated
150     */
151     private void UpdateViewOrders(localhost.Order[] Orders)
152     {
153         BindingSource bs = new BindingSource();
154         BindingList<localhost.Order> oList = new BindingList<localhost.Order>();
155
156         foreach (localhost.Order ord in Orders)
157         {
158             oList.Add(ord);
159         }
160
161         bs.DataSource = oList;
162         dataGridView2.DataSource = bs;
163
164         dataGridView2.Columns["OrderID"].DisplayIndex = 0;
165         dataGridView2.Columns["CustomerID"].DisplayIndex = 1;
166         dataGridView2.Columns["EmployeeID"].DisplayIndex = 2;
167         dataGridView2.Columns["OrderDate"].DisplayIndex = 3;
168     }
169
170
171     /*
172     *         This is where the DataGridView showing the Order Details get updated
173     */
174     private void UpdateViewOrderDetails(localhost.OrderDetail[] OrderDetails)
175     {
176         BindingSource bs = new BindingSource();
177         BindingList<localhost.OrderDetail> oList = new BindingList<localhost.
OrderDetail>();
178
179         foreach (localhost.OrderDetail ord in OrderDetails)
180         {
181             oList.Add(ord);
182         }
183
184         bs.DataSource = oList;
185         dataGridView3.DataSource = bs;
186     }
187
188
189     /*
190     *         This is where the DataGridView showing the Product get updated
191     */
192     private void UpdateViewProduct(localhost.Product p)
193     {
194         BindingSource bs = new BindingSource();
195         BindingList<localhost.Product> pList = new BindingList<localhost.Product>
();
196
197         pList.Add(p);
198
199         bs.DataSource = pList;
200         dataGridView4.DataSource = bs;
201     }
202
203
204     /*
205     *         This is when the Customer is clicked, hence get the Orders
206     */
207     private void btn_GetCustomerOrders_Click(object sender, EventArgs e)
208     {
209         this.selectedCustomer_Click(sender, e);
210     }
```

```
211
212
213     private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
214     {
215         this.selectedCustomer_Click(sender, e);
216     }
217
218     /*
219     *       This is when the Order is clicked, hence get the Order Details
220     */
221     private void dataGridView2_CellContentClick(object sender,
DataGridViewCellEventArgs e)
222     {
223         this.selectedOrder_Click(sender, e);
224     }
225
226     /*
227     *       This is when the Order Detail is clicked, hence get the Product
228     */
229     private void dataGridView3_CellContentClick(object sender,
DataGridViewCellEventArgs e)
230     {
231         this.selectedProduct_Click(sender, e);
232     }
233
234     /*
235     *       This is where the Orders get loaded
236     */
237     private void selectedCustomer_Click(object sender, System.EventArgs e)
238     {
239         Int32 selectedCellCount = dataGridView1.GetCellCount
(DataGridViewElementStates.Selected);
240         if (selectedCellCount == 1)
241         {
242             //Hence the Column chosen is CustomerID
243             if (dataGridView1.Columns[dataGridView1.SelectedCells[0].ColumnIndex]
.HeaderText.ToString() == "CustomerID")
244             {
245                 localhost.Order[] Orders = wS.GetCustomerOrders(dataGridView1.
SelectedCells[0].Value.ToString());
246                 UpdateViewOrders(Orders);
247             }
248         }
249     }
250
251     /*
252     *       This is where the Order Details get loaded
253     */
254     private void selectedOrder_Click(object sender, System.EventArgs e)
255     {
256         Int32 selectedCellCount = dataGridView2.GetCellCount
(DataGridViewElementStates.Selected);
257         if (selectedCellCount == 1)
258         {
259             //Hence the Column chosen is CustomerID
260             if (dataGridView2.Columns[dataGridView2.SelectedCells[0].ColumnIndex]
.HeaderText.ToString() == "OrderID")
261             {
262                 localhost.OrderDetail[] OrderDetails = wS.GetOrderDetails
(dataGridView2.SelectedCells[0].Value.ToString());
263                 UpdateViewOrderDetails(OrderDetails);
264             }
265         }
266     }
267
268     /*
269     *       This is where the Order Details get loaded
270     */
271     private void selectedProduct_Click(object sender, System.EventArgs e)
272     {
273     }
274 }
```

```
275     }
276
277
278     /*
279     *
280     *     This is where the Product get loaded
281     *
282     */
283     private void selectedProduct_Click(object sender, System.EventArgs e)
284     {
285
286         Int32 selectedCellCount = dataGridView3.GetCellCount           ✎
287         (DataGridViewElementStates.Selected);
288         if (selectedCellCount == 1)
289         {
290             //Hence the Column chosen is CustomerID                    ✎
291             if (dataGridView3.Columns[dataGridView3.SelectedCells[0].ColumnIndex]
292             .HeaderText.ToString() == "ProductID")
293             {
294                 localhost.Product p = wS.GetProduct(dataGridView3.SelectedCells
295                 [0].Value.ToString());                                ✎
296                 UpdateViewProduct(p);
297             }
298         }
299     }
300 }
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Drawing;
5 using System.Data;
6 using System.Text;
7 using System.Windows.Forms;
8
9 namespace ITU.WinClient
10 {
11     public partial class WaitStatus : UserControl
12     {
13         public WaitStatus()
14         {
15             InitializeComponent();
16         }
17
18         public void Waiting(object sender, EventArgs e)
19         {
20             this.Visible = true;
21         }
22
23         public void NotWaiting(object sender, EventArgs e)
24         {
25             this.Visible = false;
26         }
27     }
28 }
29 }
30 }
```