I LIBRI di

ioPROGRAMMO

IL MANUALE FONDAMENTALE PER PROGRAMMARE SUBITO
IL TUO PRIMO SITO WEB CON LE TECNOLOGIE MICROSOFT

IMPARARE ASP.NET

Antonio Pelleriti

EDIZIONI MASTER

i libri di

ioProgrammo

IMPARARE ASP.NET

di Antonio Pelleriti

EDIZIONI MASTER

INDICE

Introduzione alle Web application

Iniziare a programmare in Asp.Net 8
1.1 Hello Asp.Net 9
1.2 Coding model 12
1.3 Compilazione e distribuzione 14
1.3.1 Precompilazione 14
1.4 Creare pagine con Visual Studio .Net 16

Struttura delle applicazioni Asp.Net

2.1 Locazione delle applicazioni Asp.Net 25
2.1.1 File System 25
2.1.2 IIS 26
2.1.3 FTP 27
2.1.4 Remote Site 27
2.2 Il file Web.Config 27
2.3 Strumenti di amministrazione 28
2.4 Directory riservate 30
2.4.1 La directory Bin 31
2.4.2 La directory App_Code 31
2.4.3 La directory App_Data 34
2.4.4 La directory App_Themes 34
2.4.5 La directory App_GlobalResources 34
2.4.6 La directory App_LocalResources 35
2.4.7 La directory App_WebReferences 36
2.4.8 La directory App_Browsers 38
2.5 Ciclo di vita delle applicazioni 38
2.6 Ciclo di vita di una pagina 39
2.7 Struttura di una pagina Asp.Net 40
2.8 Le direttive di pagina 40
2.8.1 La direttiva Assembly 42
2.8.2 La direttiva Page 43

2.8.3 La direttiva Implements 48
2.8.4 La direttiva Import 48
2.8.5 La direttiva Master 49
2.8.6 La direttiva MasterType 49
2.8.7 La direttiva Control 50
2.8.8 La direttiva Register 51
2.8.9 La direttiva Reference 52
2.8.10 La direttiva OutputCache 53
2.8.11 La direttiva PreviousPageType 53

I controlli server di Asp.Net

3.1 La classe control 57
3.1.1 Proprietà di Control 57
3.2 I controlli HTML lato server 59
3.2.1 La classe HtmlControl 61
3.2.2 Gerarchia dei controlli HTML 62
3.3 I WebControl 66
3.3.1 La classe WebControl 67
3.3.2 I controlli Web standard 68
3.3.3 I nuovi controlli di Asp.Net 2.0 96
3.4 Controlli personalizzati 114
3.4.1 User Control 114
3.5 I Custom Control 116
3.6 Validazione dell'imput 119
3.6.1 Associare un validatore ad un controllo 119
3.6.2 Campi obbligatori: il controllo RequiredFieldValidator . .120
3.6.3 Confronto di valori: il controllo CompareValidator 120
3.6.4 Intervalli di valori: il controllo RangeValidator 122
3.6.5 Espressioni regolari: il controllo RegularExpressionValidator 123
3.6.6 Validazione personalizzata: il controllo CustomValidator .124
3.6.7 Riepilogare gli errori: il controllo ValidationSummary . . .125

Layout delle pagine

4.1 Le master page 129
4.2 Creare una master page 130
4.3 Creare le content page 132
4.3.1 Impostare diverse master page 134
4.3.2 Accedere alla master page 135
4.4 Lavorare con i temi 137
4.4.1 I file di un tema 138
4.4.2 Livelli di tema 138
4.4.3 Creare un tema 140

Accesso ai database e data binding

5.1 Ado.Net 145
5.2 Data binding in Asp.Net 2.0 145
5.2.1 I controlli DataSource 146
5.2.2 I controlli Data Bound 147
5.2.3 Il controllo SqlDataSource 147
5.2.4 Visualizzare i dati in una GridView 148
5.2.5 Aggiornare ed eliminare i dati 149

INTRODUZIONE ALLE WEB APPLICATION

Nei primi anni di storia del web, e per tutti i primi anni novanta, i siti web erano semplici collezioni di pagine html e magari di immagini, raggiungibili e collegate una all'altra da hyperlink, e dunque non esisteva alcun modo di generare un contenuto dinamico, o di inserire nelle pagine dei controlli più sofisticati di quelli classici forniti da HTML puro, come pulsanti, caselle di testo, tabelle, frame.

Microsoft fece il primo passo in avanti creando la prima versione delle ASP, Active Server Pages, con la possibilità di eseguire degli script sul server, alla richiesta di una pagina con estensione .asp, costituite da un insieme di html per definire la struttura e l'aspetto della pagina, e di istruzioni di codice vbscript che venivano inserite in blocchi delimitati dai caratteri <% e %>.

Con l'arrivo della piattaforma di programmazione .NET, nel luglio 2000, si passa ad una nuova versione delle pagine dinamiche, grazie al lavoro ed alle idee apportate da Marc Anders e Scott Guthrie, che decisero di abbandonare la programmazione procedurale classica di ASP, creando quella tecnologia che in origine fu chiamata ASP+ e solo in seguito ASP.NET.

ASP.NET 1.0 prima e 1.1 subito dopo, introducono una serie di importanti novità nei modelli di programmazione Web Oriented.

ASP.NET 2.0 aggiunge alla versione precedente una lunga serie di nuove caratteristiche e nuovi controlli, ma si concentra anche al miglioramento di produttività, prestazioni, configurazione e sicurezza.

Lungo le pagine del libro verrà illustrato ASP.NET 2.0, in quasi tutti i suoi aspetti, e soprattutto in maniera da mettere in grado anche i lettori poco esperti nella creazione di pagine dinamiche, di iniziare a creare applicazioni web dinamiche, con quello che costituisce forse lo stato dell'arte delle piattaforme per la programmazione internet.

Dedico questo libro alla mia famiglia, che mi ha permesso di studiare ed arrivare fino a qui, ed a Caterina che mi permette di continuare a studiare, per farmi arrivare ancora più lontano.

INIZIARE A PROGRAMMARE IN ASP.NET

Questo capitolo mostrerà i primi passi da muovere per creare la prima applicazione ASP.NET, mostrando anche come utilizzare Visual Studio 2005, che è attualmente lo strumento principe per lo sviluppo di simili applicazioni.

Fino ad ASP.NET 1.1, la presenza di IIS (Internet Information Server) era necessaria per sviluppare, eseguire, testare le applicazioni web utilizzando Visual Studio .NET 2003.

Con Visual Studio .NET 2005 invece è possibile utilizzare il web server incorporato nell'ambiente, chiamato appunto ASP.NET Development Server, per un test immediato e per finalità di debugging.

Naturalmente è ancora possibile utilizzare IIS, fondamentale per applicazioni di una certa dimensione, quando si apre un progetto da una directory virtuale esistente, e al momento di mandare in produzione un sito fatto in ASP.NET 2.0.

Chi non avesse a disposizione Visual Studio 2005 in versione standard o superiore, che è rivolto a professionisti dello sviluppo e che ha dunque un certo costo, può iniziare a sviluppare applicazioni ASP.NET 2.0 scaricando la versione trial, oppure partendo dalla versione express di Visual Web Developer 2005, completamente gratuita per ogni tipologia di utilizzo e disponibile anche in lingua italiana (l'url per ottenere VWD è http://www.microsoft.com/italy/msdn/prodotti/vs2005/editions/download/wdd.mspx).

È opportuno sottolineare comunque che non è necessario alcun ambiente di sviluppo, ma sarebbe sufficiente, così come per le applicazioni desktop, il solo compilatore a riga di comando, fornito naturalmente insieme al framework .NET 2.0, ed un editor di testo qualunque. Gli strumenti ideali per eseguire i vari esempi che si incontreranno lungo le pagine del libro sono in ogni caso i seguenti:

- Windows 2000, Windows Server 2003, o Windows XP come sistema operative.

- Visual Studio 2005 come ambiente di sviluppo.
- SQL Server 2000, SQL Server 2005 o SQL Server 2005 Express Edition per gli esempi che riguardano l'accesso ai database.

Naturalmente è necessario avere anche una minima conoscenza dell'HTML per creare il layout grafico delle pagine web.
Il codice che sta dietro alle pagine sarà scritto in C#, ma è anche possibile utilizzare il linguaggio Visual Basic 2005, o Visual J#.

1.1 HELLO ASP.NET

Prima di iniziare ad utilizzare Visual Studio .NET 2005, si vedrà come realizzare la prima pagina dinamica usando un qualunque editor di testo. Ciò darà modo di eseguire la pagina, utilizzando IIS, e parlare quindi dei meccanismi e della struttura di ogni applicazione ASP.NET.
Ogni pagina ASP.NET è formata in generale da tre sezioni: le direttive, il codice ed il layout html.
In questo primo esempio il tutto è contenuto in un singolo file con estensione aspx, ma per applicazioni più complesse, il codice sarà contenuto in un file separato, ad esempio con estensione .cs nel caso di codice scritto in C#, da quello aspx che definirà l'aspetto grafico.
Si apra dunque un editor di testo, ad esempio notepad e si scriva il codice seguente:

```
<%@ Page Language="C#"%>

<script runat="server">
  private void Saluta(object sender,EventArgs e)
  {
     Response.Write("Hello " + inputNome.Value);
  }

```

```
</script>
<html>
<head runat="server">
  <title>Hello ASP.NET</title>
</head>
<body>

  <form id="form1" runat="server">
 scrivi il tuo nome:

<input id="inputNome" runat="server" type="text"
/>
 <input id="buttonSaluta" runat="server" type="button"

value="Clicca..." onserverclick="Saluta" />

  </form>
</body>
</html>
```

Le direttive sono racchiuse fra le sequenze <%@ e %>, e servono ad impostare l'ambiente di esecuzione della pagina, ad esempio il linguaggio in cui sarà scritto il codice, oppure per importare dei namespace da utilizzare, o ancora per definire quale file conterrà il codice sorgente se esso è differente da quello attuale.
La sezione del codice, se presente e nel qual caso si parla di codice inline, contiene i gestori degli eventi scatenati dai controlli contenuti nella pagina, o dalla pagina stessa.
Per scrivere il codice bisogna racchiuderlo fra i tag <script> e </script>, e se il codice dovrà essere eseguito dal server, sarà presente anche l'attributo runat="server".
Infine la sezione layout, definisce l'aspetto della pagina, e conterrà dunque del testo, dei server controls, o altri tag HTML classici.

Prima di esaminare il funzionamento della pagina si provi ad eseguirla. Basta salvarla in un file, per esempio con nome helloworld.aspx, e copiarla dentro la directory radice di Internet Information Server, generalmente situata in c:\inetpub\wwwroot, oppure in una directory virtuale di IIS.
A questo punto si deve puntare il browser all'url della pagina stessa, ad esempio http://localhost/helloworld.aspx.
Se la pagina visualizzata è simile a quella in figura 1.1, si sarà sviluppata ed eseguita la prima pagina ASP.NET.

Figura 1.1: La prima pagina ASP.NET

La pagina contiene una etichetta, una casella di testo ed un pulsante, che sono definiti e posizionati nella sezione html del file.
Perché il server possa accedere ai controlli posizionati nella pagina, ogni tag che definisce un elemento, ad esempio un input, contiene l'attributo runat="server".
Inserendo il proprio nome e facendo clic sul pulsante, la pagina stessa sarà aggiornata, ed in testa ad essa verrà visualizzata la stringa Hello seguita dal nome inserito, come nella figura 1.2.
Cosa è accaduto dietro le quinte, cioè dal lato del server?
Il pulsante contiene un attributo OnServerClick il cui valore è impo-

stato al nome di un metodo implementato nella sezione script, il metodo Saluta, scritto come un qualunque altro metodo C#.
Al clic sul pulsante stesso, la pagina viene rieseguita, o come si imparerà a dire, effettua un PostBack su se stessa, perché anche l'elemento form possiede l'attributo runat.
A questo punto i dati contenuti negli elementi della pagina vengono inviati al server, sul quale può essere eseguito il metodo Saluta.
Chi ha già programmato in C# noterà che il metodo gestore del click ha la firma classica del delegate EventHandler.
Il metodo può accedere al valore contenuto nella casella di testo inputNome, quindi lo concatena con la stringa Hello, ed infine esegue il metodo Response. Write, che serve a scrivere sul flusso http in uscita, cioè in parole povere su ciò che verrà visualizzato poi nel browser. Quello che è fondamentale sottolineare fin da subito, è che tutto avviene sul server, mentre il client, cioè il browser, invia i dati da elaborare e visualizza i risultati dell'elaborazione remota.

Figura 1.2: Hello World!

1.2 CODING MODEL

La pagina creata nella sezione precedente ha una sezione di codice cosiddetta inline, cioè contenuta in un apposito blocco <script>

della pagina stessa, e con l'attributo runat, per specificare che esso verrà eseguito lato server.
Questo modello di scrittura è anche detto single file, perché tutto, HTML e codice, è contenuto in un solo file. L'alternativa, usata nelle applicazioni web più complesse, è quella di separare il codice, scrivendolo in un file separato da quello che definisce il layout HTML.
In questo caso il codice che supporta la pagina è scritto in un file sorgente che sta "dietro" la pagina stessa, e la direttiva Page serve ad indicare quale sia il nome del file mediante l'attributo Src o Codebehind, e da quale classe derivare la pagina tramite l'attributo Inherits.
Ad esempio la direttiva Page potrebbe essere:

```
<%@ Page Language="C#" Inherits=

"HelloWorld" src="HelloWorld.cs">
```

La classe implementata nel file sorgente deve derivare dalla classe Page. La pagina aspx invece rappresenta una classe totalmente diversa, a sua volta derivata da quella definita nella classe code-behind. Ciò implica che la classe code-behind deve contenere delle variabili di classe che replicano tutti i controlli contenuti nella pagina, e dunque ad ogni aggiunta o modifica dei controlli nel file aspx, deve corrispondere una modifica nelle variabili della classe che sta dietro.
La nuova modalità code-behind, introdotta invece in quest'ultima versione, è detta anche code-beside per distinguerla, e sfrutta il concetto di partial class introdotto in .NET 2.0.
In questo caso infatti la classe implementata nel file sorgente è appunto una classe parziale, derivante ancora da Page, ma che contiene solo la parte di codice strettamente necessaria, ad esempio i gestori degli eventi, mentre al contrario le variabili di classe che rappresentano i controlli verranno generati automaticamente dal motore ASP.NET, evitando che i file aspx e di codice sorgente possano non essere in

sincronia, e permettendo di concentrarsi dunque su parti di codice più importanti e che riguardano la logica applicativa.
In questo modello l'attributo Src o Codebehind è sostituito da CodeFile:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="CodeBeside.aspx.cs" Inherits="CodeBeside" %>
```

Il nuovo modello code behind, o code-beside, consente dunque una maggiore produttività, una migliore organizzazione dello sviluppo, ed una più facile gestione delle modifiche e delle correzioni.

1.3 COMPILAZIONE E DISTRIBUZIONE

Le pagine aspx, alla prima richiesta fatta, vengono trasformate in vere classi .NET e compilate dunque come qualunque altra classe.
In ASP.NET 2.0 il processo di compilazione è differente da quello delle precedenti release.
Il Code Behind delle pagine aspx delle applicazioni ASP.NET 1.0 e 1.1, viene compilato costituendo un unico assembly da pubblicare nella directory Bin della radice del sito.
Ciò significa che ad ogni modifica, anche al codice di una sola pagina, cioè di ogni singola classe, l'intera applicazione deve essere ricompilata e spedita nuovamente sul server.
ASP.NET 2.0 invece compila ogni pagina, e dunque ogni corrispondente classe, in un assembly separato dagli altri. Inoltre tale compilazione avviene solo la prima volta che una pagina viene acceduta, ed eventualmente se e quando la pagina verrà modificata.

1.3.1 Precompilazione

ASP.NET 2.0 permette di precompilare l'intera applicazione, evitando dunque il tempo di latenza necessario alla compilazione che avviene alla prima richiesta di ogni pagina, e la necessità di pubblica-

re sul server di produzione anche i file sorgente di code-behind.
La precompilazione può avvenire in due differenti modalità, quella in-place e quella di deployment. Nel primo caso avviene la compilazione di tutte le pagine aspx, per forzarne la compilazione, ed il sito è dunque compilato nell'ambiente di produzione finale, ma prima di essere reso pubblico. La compilazione in-place avviene utilizzando il comando aspnet_compiler fornito dall'SDK di .NET nella seguente maniera:

```
aspnet_compiler –v /PercorsoApplicazione
```

il percorso specificato dall'opzione –v rappresenta il percorso della directory virtuale del sito in IIS. Se il comando viene lanciato una seconda volta, solo le pagine modificate saranno compilate nuovamente.Nel caso della compilazione per il deployment, viene generata una rappresentazione statica del sito, fatta di tutti i file necessari per il suo funzionamento, e che potranno essere distribuiti sulla macchina server che ospiterà l'applicazione web in produzione.
In questo caso il comando da eseguire è in generale

```
aspnet_compiler  [-m metabasePath | -v virtualPath] [-p physicalDir]

[targetDir]
```

Una spiegazione completa delle varie opzioni si può ottenere utilizzando il comando

```
aspnet_compiler -?
```

Almeno una delle due opzioni –m oppure –v deve essere specificata. Per esempio:

```
aspnet_compiler -v /PercorsoApp -p c:\WebSite c:\PrecompiledWebSite
```

in questo caso l'opzione –p specifica il percorso fisico dell'applicazione, mentre la directory destinazione dei file è indicato come ultimo parametro, in questo caso c:\PrecompiledWebSite.
Eseguito il comando basta copiare il contenuto di tale cartella destinazione sul server di produzione.
Si tenga presente che nel caso di nuove pagine aggiunte o modificate, deve essere rieseguita una precompilazione completa ed una nuova distribuzione dei file prodotti.

1.4 CREARE PAGINE CON VISUAL STUDIO .NET

Dato che Visual Studio 2005, o Visual Web Developer Express, saranno compagni di viaggio fondamentali, si vedrà ora come creare una nuova applicazione web ASP.NET in questi ambienti, in maniera da prendere confidenza con i comandi e gli strumenti di sviluppo.
Per creare il primo progetto ASP.NET è sufficiente selezionare il menù File, e quindi fare clic su New e poi su Web Site. A questo punto dalla schermata che si può vedere in figura 1.3, si selezioni il template ASP.NET Web Site, e si scelga la posizione in cui creare il sito, cioè una fra File System, http, o FTP, ed il linguaggio preferito fra Visual C# , Visual Basic, Visual J#. Per comodità e semplicità, si scelga per ora File System, specificando dunque la cartella dentro la quale creare il progetto web.

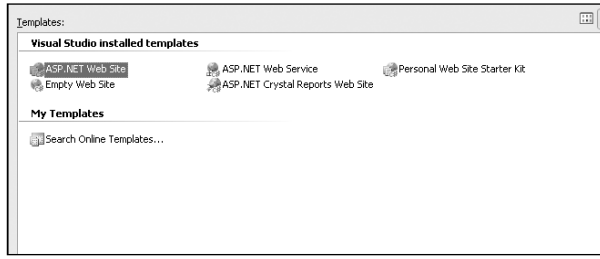

Figura 1.3: La creazione di un nuovo sito ASP.NET.

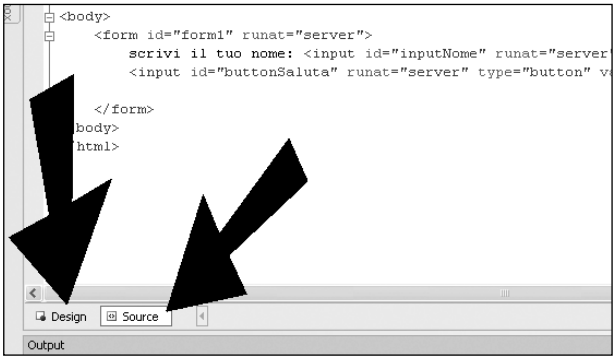

Figura 1.4: modalità design e source

Visual Studio creerà automaticamente una pagina aspx, con nome default.aspx ed il file di code behind, che sarà default.aspx.cs nel caso di C#, o con la relativa estensione del linguaggio scelto.
Inoltre verrà creata la directory App_Data, di cui si vedrà in seguito la funzione.
La finestra Solution Explorer sulla destra consente di esplorare tutti gli elementi del sito web.
Con un doppio click su un elemento, ad esempio su default.aspx, esso verrà aperto nel designer.
Nel caso di una pagina aspx, sarà possibile visualizzarne il sorgente HTML facendo clic su Source, oppure il layout grafico facendo invece clic sul pulsante Design. (figura 1,4)
La pagina aspx creata da Visual Studio avrà la seguente struttura HTML:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>
 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
```

```
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
  </form>
</body>
</html>
```

È possibile notare la presenza dell'unico elemento form, con l'attributo runat. In tal modo la pagina sarà eseguita sul server.
Sempre dal Solution Explorer, si può espandere la struttura della pagina default.aspx, visualizzando il file di Code Behind. (figura 1.5)

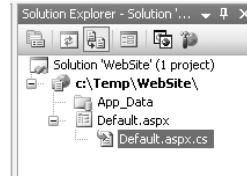

Figura 1.5: visualizzare il file di code behind

Facendo doppio click sul file sorgente, questo verrà aperto nell'editor di codice, che nel caso di una pagina in linguaggio C# visualizzerà il seguente codice:

```
using System;

public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
  }
}
```

Ogni classe che rappresenta una pagina aspx, deriva dunque dalla classe System.Web.UI.Page ed è una partial class..
La ToolBox sulla sinistra contiene i controlli, raggruppati in categorie, eventualmente personalizzabili, che è possibile aggiungere ad una pagina aspx, basta selezionarne uno e trascinarlo sulla pagina, sia in modalità Design che Source.
Si selezioni ad esempio una Label, e la si trascini sul designer.
Verrà creato un controllo Label con posizionamento e testo di default.Il codice HTML sarà aggiornato con l'aggiunta di un tag asp:Label:

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
```

Anche il controllo Label sarà naturalmente dotato dell'attributo runat.
La Properties Window mostrerà le proprietà dell'elemento selezionato, ad esempio selezionando dal designer la Label appena aggiunta, si vedranno le proprietà dell'istanza del controllo che potranno essere modificate a proprio piacimento, impostando ad esempio la proprietà Text. (figura 1.6)

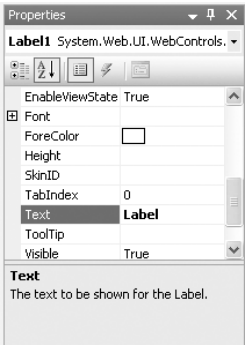

Figura 1.6: impostare le proprietà di un controllo

Un controllo può essere aggiunto anche trascinandolo direttamente nella pagina di sorgente HTML, penserà Visual Studio in questo caso a generare il tag, ad esempio se si seleziona un controllo TextBox e lo si trascina sull'editor HTML verrà generato il codice per creare una TextBox nel punto di rilascio del mouse:

```
<asp:TextBox ID="TextBox1" runat="server">
</asp:TextBox>
```

Ritornando al designer grafico, naturalmente verrà visualizzata graficamente la TextBox. Si selezioni ora allo stesso modo un controllo Button ed un altro controllo Label, posizionandoli dove si vuole ed impostando il loro testo, ad esempio come mostrato in figura 1.7

Figura 1.7: i controlli posizionati su una pagina

Il corrispondente codice HTML, all'interno del tag form, assomiglierà al seguente:

```
<asp:Label ID="Label1" runat="server" Text="Inserisci il tuo
nome:"></asp:Label>

<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:Button ID="Button1" runat="server" Text="Clicca qui" />

<br />
<asp:Label ID="Label2" runat="server"></asp:Label></div>
```

Fino ad ora si è lavorato solo all'aspetto grafico della pagina aspx, non toccando minimamente il codice nel file di Code Behind, ed inoltre Visual Studio 2005 si è occupato autonomamente di generare il codice HTML.
Supponiamo adesso di voler eseguire un'azione al click sul pulsante Button1.
Facendo doppio click su di esso, nel designer verrà generato il codice del metodo di gestione dell'evento di default, che per un pulsante è l'evento Click.
Aprendo il file default.aspx.cs si vedrà il codice seguente:

```
protected void Button1_Click(object sender, EventArgs e)

{
}
```


All'interno del metodo è possibile effettuare le operazioni desiderate, accedendo ai controlli della pagina tramite il loro ID autogenerato o quello che è possibile impostare nelle proprietà, per ognuno di essi.
Si può ad esempio ricavare il contenuto della TextBox, e poi usarlo per visualizzare il testo della seconda label, basta scrivere:

```
protected void Button1_Click(object sender, EventArgs e)
{
    string nome=this.TextBox1.Text;

    this.Label2.Text = "Ciao " + nome;
}
```

Tutto ciò è possibile grazie al fatto che la classe generata è una classe parziale, ed è nella parte non visibile in formato C#, corrispondente alla parte di file aspx, che vengono generati ed

eventualmente aggiornati tutti i campi privati su cui vengono mappati i controlli della pagina stessa. Tornando ad esaminare l'HTML della pagina, si noterà che l'elemento Button1 possiede adesso un attributo OnClick, con valore uguale al nome del metodo appena generato:

```
<asp:Button ID="Button1" runat="server" Text="Clicca qui"

OnClick="Button1_Click" />
```

Se si vuol invece gestire un evento diverso da quello di default, basta selezionare il controllo, cliccare sull'icona Events nella finestra delle proprietà e scegliere l'evento da gestire.
Un doppio click su di esso creerà un metodo gestore con un nome di default, del tipo IDControllo_NomeEvento.
Per eseguire, la pagina in modalità Debug, è a questo punto sufficiente cliccare il pulsante con il simbolo Play sulla Toolbar, o premere F5.
La prima volta che si esegue il debug, Visual Studio avvertirà che non esiste un file di configurazione web.config, e chiederà se si vuole generarlo impostando in esso anche la modalità di Debug. Confermando tale scelta, verrà dunque creato un file in formato XML che contiene tutte le configurazioni dell'applicazioni, fra le quali l'impostazione:

```
<compilation debug="true"/>
```

Ogni applicazione può contenere un file web.config nella directory radice del sito, ed ogni sottodirectory può contenere un proprio web.config, per effettuare configurazioni locali alle directory in maniera gerarchica, vale a dire che se una sottodirectory non contiene un file web.config, allora sarà utilizzato quello della directory superiore, eventualmente risalendo fino a

quello principale della radice. Non è necessario che un'applicazione ASP.NET abbia un file Web.Config, ma senza non è possibile effettuarne il debug, ecco perché al primo avvio Visual Studio ne propone la creazione.
Si vedranno in seguito le possibili impostazioni che i file di configurazione permettono di effettuare.
Avviato il debug, verrà eseguito il Web Server di sviluppo, avendo in questo caso scelto di creare l'applicazione su File System, e quindi verrà aperto il browser internet all'indirizzo sul quale verrà mandata in esecuzione la pagina creata (figura 1.8).
Creando l'applicazione direttamente in una directory virtuale di IIS naturalmente verrà utilizzato IIS per eseguire l'applicazione.

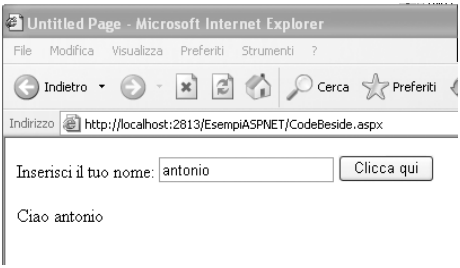

Figura 1.8: L'esecuzione in debug di una pagina.

STRUTTURA DELLE APPLICAZIONI ASP.NET

Dopo aver dato un rapido sguardo alle possibilità che ASP.NET 2.0 offre, si comincerà ora l'approfondimento dei singoli aspetti di un'applicazione, a partire dalle opzioni possibili per ospitarle su un server fino ad analizzarne il ciclo di vita.

2.1 LOCAZIONE DELLE APPLICAZIONI ASP.NET

Visual Studio 2005, alla creazione di un nuovo sito web in ASP.NET 2.0, oppure quando si vuole aprirne uno esistente, offre diverse opzioni per scegliere la sua posizione e la modalità di connessione. Tale scelta può essere fatta una volta che si è selezionata la voce di menù File‡New‡Web Site, facendo clic sul pulsante Browse, che presenterà la finestra di dialogo Choose Location mostrata in figura 2.1

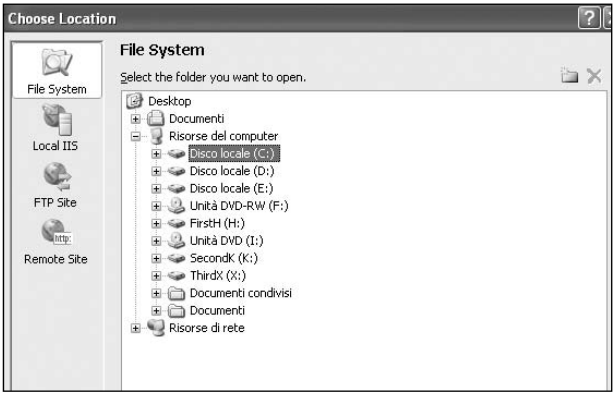

Figura 2.1: Selezione della posizione in cui creare il sito.

2.1.1 File System

Per default Visual Studio 2005 propone di creare l'applicazione in una directory classica del File System, al di fuori di Internet Infor-

mation Server. In questo caso verrà utilizzato il Web Server di sviluppo integrato nell'IDE stesso (l'erede del vecchio Cassini Web Server) per eseguire l'applicazione.
Questa opzione è utile per chi non possiede una macchina ed un sistema operativo con una installazione di IIS, e quindi permette, a differenza delle vecchie versioni, lo sviluppo in ASP.NET ad esempio anche su Windows XP Home, in quanto l'applicazione non sarà eseguita da una directory virtuale di IIS ma dalla directory stessa in cui è stata creata.Quando viene avviato il debug di un'applicazione creata su File System, è possibile visualizzare la schermata del web server di sviluppo, cliccando sull'icona presente sulla barra delle applicazioni di Windows. La figura mostra la schermata del server in esecuzione (figura2.2)

2.1.2 IIS

L'opzione Local IIS permette di sfogliare il web server IIS locale, di selezionare una directory virtuale esistente in cui creare l'applicazione, oppure di creare una nuova radice virtuale del sito Web predefinito, o infine di creare una nuova directory virtuale. In questo caso Visual Studio utilizzerà IIS per eseguire l'applicazione, utilizzando un url del tipo http://localhost/NomeApplicazione/default.aspx.

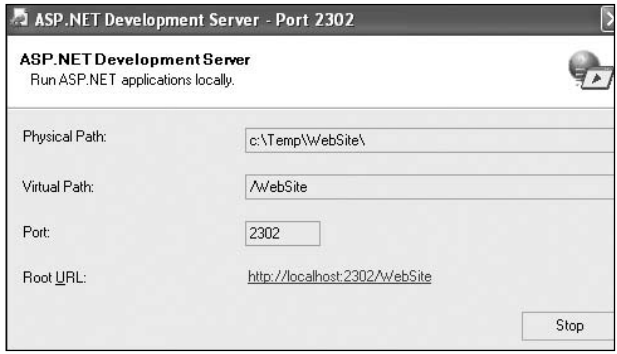

Figura 2.2: Il server di sviluppo in esecuzione

2.1.3 FTP

Con le opzioni precedenti era implicito il fatto che l'applicazione creata dovesse risiedere sulla macchina locale di sviluppo.
Con l'opzione FTP è possibile invece creare un sito web ASP.NET in remoto, connettendosi ad un server FTP, su cui si hanno naturalmente privilegi di lettura e scrittura.
Esso può essere sia un server della intranet aziendale, ma anche un qualsiasi server FTP in giro per il mondo, ad esempio quello di un provider internet che ospiterà il vostro sito.

2.1.4 Remote Site

L'ultima opzione è quella di creazione del sito su un web server, locale o remoto, con installate le FPSE (FrontPage Server Extensions), e con la possibilità di connessione anche utilizzando Secure Sockets Layer (SSL), cioè su protocollo HTTPS invece di http.
In questo caso è necessario avere i permessi di amministrazione delle estensioni FrontPage, per creare file e directory sul server.
Per utilizzare questa opzione è necessario specificare l'url completo del server ed il nome del sito web da creare.

2.2 IL FILE WEB.CONFIG

È possibile configurare una applicazione ASP.NET utilizzando uno o più file in formato XML, con nome obbligatorio Web.Config. Il file di configurazione è posizionato direttamente nella radice dell'applicazione, ma nel caso in cui esistano più sottodirectory, in cui risiedono altre pagine aspx, possono essere eseguite delle impostazioni solo per queste pagine, inserendo altri Web.Config nelle relative directory di appartenenza.
Nel caso in cui una sottodirectory non contenga un file di configurazione, il runtime di ASP.NET risalirà alla directory madre, fino a trovare un file Web.Config, o fermandosi al massimo alla radice del sito.Per gestire il file Web.Config è possibile evitare la

digitazione manuale delle diverse opzioni, ed utilizzare uno strumento visuale. Tale strumento è un'estensione della console di gestione di Internet Information Server (eseguibile dal pannello di controllo, poi strumenti di amministrazione), ed è dunque attivabile dopo aver aperto la finestra delle proprietà di un sito da configurare, navigando sulla schermata ASP.NET, e facendo clic poi sul pulsante Edit Configuration (figura2.3). Dalla schermata che si aprirà a questo punto possono essere modificate praticamente tutte le impostazioni del file web.config. Per esempio nella schermata seguente è mostrata la pagina per configurare le stringhe di connessione ai database (figura 2.4)

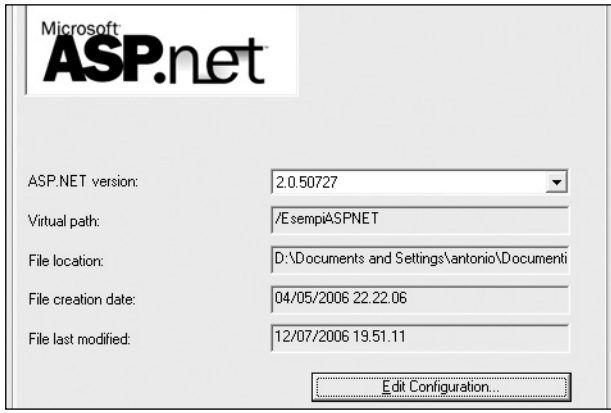

Figura 2.3: Configurazione delle proprietà ASP.NET

2.3 STRUMENTI DI AMMINISTRAZIONE

ASP.NET 2.0 mette a disposizione uno strumento di amministrazione, sotto forma di un'applicazione web, per configurare e gestire diversi aspetti di un sito ASP.NET, come quelli di sicurezza, o per definire le pagine di errore, o ancora le opzioni di debug.

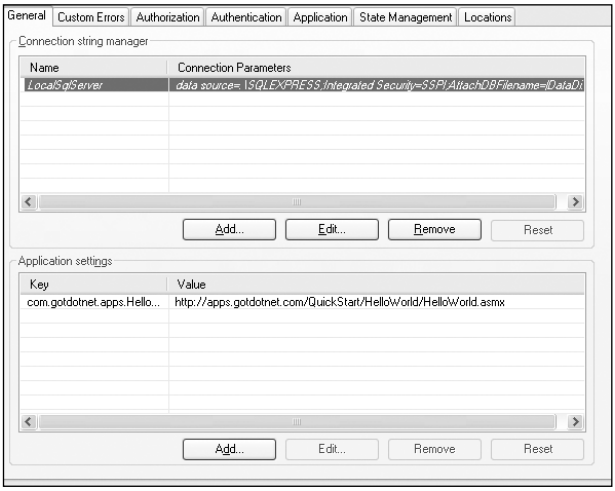

Figura 2.4: Impostazioni del file web.config

Se si utilizza Visual Studio, è possibile lanciare i Web Site Administration Tool, selezionando la voce ASP.NET Configuration dal menù Website.

Lo strumento di amministrazione contiene diversi menù e comandi, nella figura seguente (figura 2.5) è mostrata la schermata principale del tool. Il collegamento in altro a destra, "How do I use this tool?", fornisce un dettagliato aiuto sulle diverse opzioni.

Se non si possiede Visual Studio è possibile lanciare il tool aprendolo direttamente nel browser, e specificando l'applicazione da configurare.

Per esempio, se il server locale è eseguito sulla porta 1234, per configurare un'applicazione denominata EsempiASPNET creata sul file system locale nella directory c:\EsempiASPNET, si potrà utilizzare l'URL seguente:

http://localhost:1234/ASP.NETwebadminfiles/default.aspx?applicationPhysicalPath=c:\EsempiASPNET\&applicationUrl=/EsempiASPNET

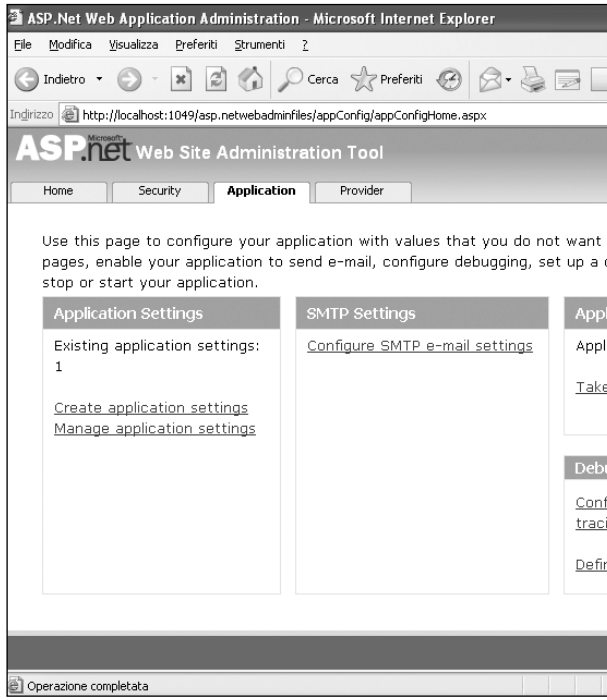

Figura 2.5: Il Web Site Administation Tool.

2.4 DIRECTORY RISERVATE

Una volta creata una applicazione ASP.NET 2.0, Visual Studio 2005, oltre ad aggiungere la prima pagina default.aspx, crea anche una directory App_Data.

L'icona stessa, diversa da quella di una classica cartella, indica che essa è una cartella speciale.

ASP.NET 2.0 utilizza otto diverse cartelle riservate, di cui una sola, la directory Bin, era già presente nella versione 1.1.

Tali directory non sono obbligatoriamente presenti in un'applicazione Web, e quindi devono essere create appositamente, sia a

mano, sia tramite Visual Studio 2005.
L'IDE permette tale operazione facendo clic con il tasto destro sul nome dell'applicazione nel Solution Explorer, e scegliendo dal menù contestuale attivato la voce "Add ASP.NET Folder", e dal sottomenù successivo il nome della reserved folder da aggiungere. La figura seguente (figura 2.6) mostra tutte le directory riservate di ASP.NET nel Solution Explorer.

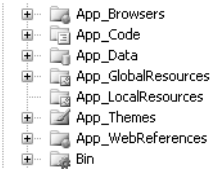

Figura 2.6: Le cartelle riservate di ASP.NET 2.0.

2.4.1 La directory Bin

Già presente in ASP.NET 1.1, la cartella Bin conserva anche nella versione 2.0 la stessa funzione, quella di contenere assembly .NET di terze parti o sviluppate in progetti di librerie, che includono controlli web, componenti, o altro codice che è possibile utilizzare nell'applicazione web.
Inoltre, se si decide di precompilare l'applicazione stessa, la directory Bin conterrà gli assembly risultato della precompilazione.
Il runtime ASP.NET permetterà di utilizzare nelle pagine aspx qualunque assembly presente nelle directory Bin, senza alcuna registrazione o configurazione particolare.
Se viene rilasciata una nuova versione di un assembly utilizzato dall'applicazione, basta sostituirlo nella Bin e l'applicazione la utilizzerà in maniera trasparente.

2.4.2 La directory App_Code

La directory riservata App_Code è utilizzata per contenere il sor-

gente di classi che saranno utilizzate dall'applicazione, ad esempio classi che implementano la logica di business, la logica di accesso ai database, o ancora generiche classi helper.
Non è possibile inserire nella directory App_Code elementi diversi da file sorgenti, siano essi scritti in C#, Visual Basic, o Visual J#, né dunque pagine aspx, nè Web Controls, immagini o altro, ed inoltre un solo linguaggio può essere utilizzato, dato che un tipo di compilatore può trattare solo il relativo tipo di sorgenti.
Se fosse necessario, però, utilizzare classi scritte in diversi linguaggi, ad esempio perché nel team alcuni sviluppatori scrivono in C# , altri preferiscono VB, altri hanno scritto delle utilità in J#, è sempre possibile creare più sottodirectory, ad esempio CSharpCode e VbCode, ed aggiungere al file di configurazione Web.Config una sezione codeSubDirectories, sottosezione di compilation, in cui elencare appunto le precedenti sottodirectory, fra l'altro in tal modo l'icona nel Solution Explorer delle sottodirectory verrà modficata.

```
<compilation debug="true">
    <codeSubDirectories>
      <!-- directory contenente i sorgenti C#-->
      <add directoryName="CSharpCode"/>
      <!-- directory contenente i sorgenti VB-->
      <add directoryName="VBCode" />
    </codeSubDirectories>
</compilation>
```

Il web.config contenente tale sezione deve essere sempre e solo quello contenuto nella directory root..
Si provi a scrivere dunque una classe, nella directory App_Code, in maniera da utilizzarla in una Web Form dell'applicazione.
Supponiamo di voler visualizzare in una data pagina delle citazioni di poeti o autori famosi, scelte una alla volta da un insieme.
Si crei la classe Quote in questa maniera:

```
public class Quote
{
  private List<string> quotes;
  private Random random;
    public Quote()
    {
      random = new Random();
      quotes = new List<string>();
      quotes.Add("Homo homini lupus");
      quotes.Add("Nemo profeta in patria");
      quotes.Add("Dura lex sed lex");
  }
  public string GetNextQuote()
  {
      int next=random.Next(0, quotes.Count);
      return quotes[next];
  }
      }
```

A questo punto si può aggiungere una pagina aspx, chiamandola ad esempio GetQuote.aspx, e con un pulsante al cui click si invocherà il metodo GetNextQuote della classe Quote per impostare il testo di una Label con la citazione ottenuta. Il gestore del click sarà il seguente:

```
protected void btShowQuote_Click(object sender, EventArgs e)
{
    Quote quote = new Quote();
    labQuote.Text = quote.GetNextQuote();
}
```

Si può verificare che è possibile aggiungere una classe VB o VJ#, ed utilizzarla dalla stessa pagina, o comunque in una pagina che ha invece il code-behind scritto ad esempio in C#.

2.4.3 La directory App_Data

La directory App_Data conterrà i file di dati utilizzati dall'applicazione, siano essi file di Microsoft Sql Server 2005 (.mdf), file di database Microsoft Access (.mdb), file XML e così via, contenenti ad esempio dati utilizzati per l'autenticazione degli utenti ed i relativi ruoli. L'account utente utilizzato dall'applicazione ASP.NET, per default l'account ASPNET, avrà accesso sia in lettura che in scrittura.

2.4.4 La directory App_Themes

ASP.NET 2.0 permette di definire e configurare l'aspetto delle pagine utilizzando temi e skin, per mezzo di file contenuti nella directory App_Themes.

Essa dunque potrà contenere diverse sottodirectory, una per ogni tema utilizzabile per le pagine, tema che dunque sarà il complesso di file .skin e fogli di stile .css, o anche di immagini ed altre risorse grafiche e non.

2.4.5 La directory App_GlobalResources

I file di risorse sono delle tabelle che possono contenere stringhe, immagini ed altri file, utilizzabili dall'applicazione.

I file di risorse hanno estensione resx, e vengono compilati in degli assembly, che inclusi nella directory App_GlobalResources avranno scope globale, cioè utilizzabili da ogni punto dell'applicazione, e sono fortemente tipizzati, quindi utilizzabili programmaticamente come una qualunque classe. Le risorse sono memorizzate come coppie chiave-valore, qualunque sia il tipo, in maniera da poter essere ricavate accedendo mediante la chiave. Per esempio, se nella directory App_GlobalResources si è creato un file Resource.resx, contenente delle risorse stringhe, con chiave "ApplicationName" e "PageTitle" il suo valore potrà essere ricavato nel code behind di una qualsiasi pagina, dal namespace Resources:

```
protected void Page_Load(object sender, EventArgs e)
{
    Page.Title = Resources.Resource.PageTitle;
    labAppName.Text= Resources.Resource.ApplicationName;
}
```

Nella parte HTML possiamo invece utilizzare analogamente le risorse globali, utilizzando la sintassi seguente:

```
<asp:Label ID="Label1" Text="
<%$ Resources:Resource, Messaggio
%>"  runat="server" />
```

Supponendo che il file Resource.resx contenga una risorsa stringa, con nome Messaggio, il suo valore verrà utilizzato per impostare la proprietà Text del controllo Label1.
I file risorse possono essere usati per localizzare applicazioni ASP.NET, e quindi visualizzare le stringhe nella lingua impostata per il browser dell'utente, senza dover ricorrere ad alcuna modifica nel codice. Cio è possibile semplicemente aggiungendo nella stessa App_GlobalResources, un file di risorse con lo stesso nome, ed aggiungendo un suffisso prima dell'estensione resx, in questo caso Resource.it-IT.resx.
In questo modo, l'applicazione utilizzerà le stringhe ricavate da Resources.resx per default, mentre le ricaverà da quest'ultimo file, se il browser ha impostazioni italiane.

2.4.6 La directory App_LocalResources

Così come visto per le risorse globali, realizzabili con i file della directory App_GlobalResources, si possono anche creare file di risorse locali ad una singola pagina.
In questo caso essi saranno memorizzati nella directory App_LocalResources, che, nel caso in cui la pagina si trovi in una sottodirectory,

dovrà essere copiata nella stessa directory della pagina.
Se la pagina che deve usare risorse locali si chiama ad esempio localresources.aspx, i file di risorse ad essa associati saranno del tipo:

localresources.aspx.resx
localresources.aspx.it.resx
localresources.aspx.ja.resx

e così via per eventuali altri linguaggi da supportare.
La pagina localresource.aspx utilizzerà il file localresources.aspx.resx se non esiste un file associato alla lingua impostata, altrimenti ricaverà ed utilizzerà quello relativo a questa.

2.4.7 La directory App_WebReferences

La directory App_WebReferences contiene i file che permettono l'utilizzo dei Web Service in un'applicazione ASP.NET, dunque file wsdl, xsd, e documenti per il discovery dei servizi, cioè disco e discomap. Utilizzare un web service di cui si conosce l'url, in Visual Studio 2005 è semplicissimo.
Dopo aver creato la directory App_WebReferences, facendo clic su di essa con il tasto destro è sufficiente selezionare poi la voce di menù Add Web Reference, e nella dialog che si apre a questo punto, è inserire l'indirizzo di un web service esistente, oppure ricercarlo nella soluzione, sulla macchina locale, o nella rete a cui è connessa la macchina.
Per questo esempio si può utilizzare un servizio hello world, raggiungibile all'indirizzo seguente:

http://apps.gotdotnet.com/QuickStart/HelloWorld/HelloWorld.asmx?WSDL

Cliccando sul pulsante Go, verrà visualizzato l'unico metodo esposto dal web service, SayHelloWorld, mentre il web reference sarà creato con un nome inverso dell'url inserito, in questo caso cm.got-

dotnet.apps, come mostrato in figura 2.7

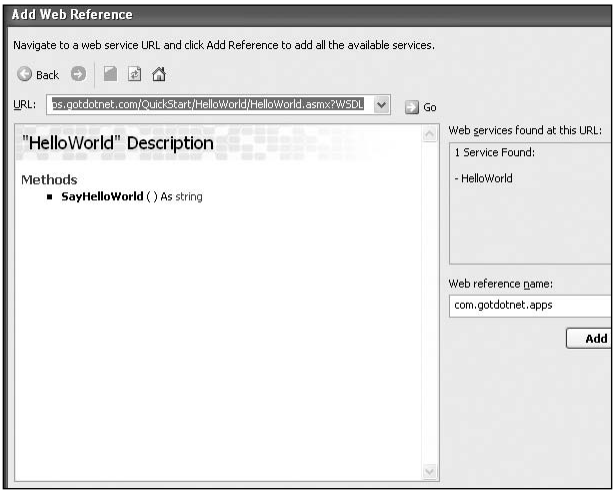

Figura 2.7: Aggiungere un riferimento ad un Web Service

A questo punto con un click sul pulsante Add Reference verrà generato nella directory App_WebReferences un albero con la struttura com/gotdotnet/apps.
Nella cartella apps saranno creati due file: HelloWorld.discomap ed HelloWorld.wsdl.
Il file web.config dell'applicazione sarà inoltre modificato con l'aggiunta delle righe:

```
<appSettings>
<add key="com.gotdotnet.apps.HelloWorld" value="http://apps.gotdot
net.com/QuickStart/HelloWorld/HelloWorld.asmx"/>
</appSettings>
```

Per consumare il web service, trattandosi di un'applicazione ASP.NET, si può utilizzare una nuova Web Form, con ad esempio un pulsante,

e nel gestore dell'evento click di questo si istanzierà la classe com.gotdotnet.apps.HelloWorld, come si farebbe una qualsiasi altra classe: Sull'oggetto creato è sufficiente invocare il metodo SayHelloWorld esposto dal servizio, ad esempio per impstare il testo di una Label:

```
protected void Button1_Click(object sender, EventArgs e)
{
    com.gotdotnet.apps.HelloWorld hello = new
com.gotdotnet.apps.HelloWorld();
  Label1.Text=hello.SayHelloWorld();
}
```

Niente di più semplice in questo caso, ma continuando a sviluppare web service o utilizzandone di più complessi si vedranno altre potenzialità, e modalità di utilizzo.

2.4.8 La directory App_Browsers

Nella directory App_Browser vengono conservati file di definizione dei browser (con estensione .browser), che non sono altro che file in formato XML contenenti informazioni sui browser che effettuano la richiesta delle pagine in maniera da identificarne il tipo e le capacità di rendering delle pagine stesse.
Ciò è utile per creare pagine che adattino il loro comportamento, e quello dei web control che contengono, alle possibilità del browser, ed in particolare è utile per i browser dei dispositivi mobili, come palmari e cellulari, notoriamente con possibilità più ristrette rispetto a quelli di un computer normale.

2.5 CICLO DI VITA DELLE APPLICAZIONI

Durante il ciclo di vita di un'applicazione, vengono generati diversi eventi, gestibili per mezzo di metodi, come un qualsiasi altro even-

to .NET. Per gestire un evento a livello di applicazione è però necessario creare un file Global.asax nella radice del sito.
Per far ciò, basta aggiungere un nuovo elemento alla soluzione dell'applicazione, e selezionare come tipo una Global Application Class, chiamandola con il nome proposto di default.
Ogni applicazione può avere un solo file Global.asax, che verrà compilato da ASP.NET in una classe derivata da HttpApplication, e che rappresenterà appunto l'applicazione. Gli eventi dell'applicazione saranno automaticamente collegati ai gestori del file Global.asax, usando la convenzione Application_Evento.
Application_Start viene invocato alla prima richiesta di una risorsa dell'applicazione, ad esempio di pagina aspx, ed è quindi utilizzato per operazioni di inizializzazione che dovranno avere scope globale per tutte le successive richieste.
Analogamente, l'evento Application_End sarà invocato solo quando l'applicazione verrà terminata. In generale il ciclo di vita inizia quando al web server, arriva una richiesta da un browser, per una certa pagina. A questo punto vengono creati gli oggetti fondamentali dell'applicazione, come HttpContext, HttpRequest ed HttpResponse.
HttpContext contiene gli oggetti specifici della richiesta corrente, vale a dire proprio un HttpRequest ed un HttpResponse, e cioè rispettivamente il contenitore delle informazioni relative alla richiesta stessa, come i cookie e le caratteristiche del browser, ed il contenitore della risposta da rispedire al browser, come i cookie da scrivere e l'html da visualizzare. Dopo aver creato questi oggetti fondamentali l'applicazione inizia la sua esecuzione creando un oggetto HttpApplication, se esiste però un file Global.asax viene creata un'istanza di tale classe derivata da HttpApplication.

2.6 CICLO DI VITA DI UNA PAGINA

Quando una pagina ASP.NET viene eseguita in seguito ad una richiesta del browser, il motore ASP.NET verifica se la pagina stessa

deve essere compilata, o viceversa se ne esiste una versione compilata, ed a questo punto inizia il suo ciclo di vita.
Durante questo ciclo di vita, vengono generati degli eventi che è possibile gestire via codice.
Gli eventi utilizzati più di frequente sono i seguenti, nell'ordine in cui si verificano:

- Page_PreInit.
- Page_Init
- Page_Load
- Page_LoadComplete
- Page_PreRender
- Page_PreRenderComplete.
- Page_Unload.

2.7 STRUTTURA DI UNA PAGINA ASP.NET

ASP.NET 2.0 consente due diverse modalità di creazione di una pagina, la prima consente di inserire codice direttamente nel file aspx, inframmezzandolo ai tag HTML ed al testo, mentre la seconda opzione usa il nuovo modello Code Behind, in cui il codice è scritto in un file differente, cosa che consente di separare dunque la logica di presentazione da quella di business.
In questa seconda modalità per ogni pagina con estensione aspx, esisterà un file con estensione aspx.cs o aspx.vb, e così via a seconda del linguaggio utilizzato.

2.8 LE DIRETTIVE DI PAGINA

Ogni pagina ASP.NET che contiene del codice, sia esso inline che code behind, contiene anche delle direttive, cioè delle istruzioni in un particolare formato, che controllano il comportamento della pagi-

na, dalla sua compilazione all'esecuzione stessa.
Le direttive di pagina sono racchiuse fra le sequenze <%@ e % >, e possono essere dotate di diversi attributi. La sintassi generica è la seguente:

```
<%@ Direttiva [attributo1=valore1] [attributo2=valore2] … %>
```

Un esempio di direttiva di pagina, creata automaticamente da Visual Studio 2005, potrebbe essere il seguente:

```
<%@ Page Language="C#" AutoEventWireup="false"

CodeFile="Default.aspx.cs"
Inherits="_Default" %>
```

In ASP.NET 2,0 sono a disposizione dello sviluppatore ben undici direttive utilizzabili in una pagina, o come vedremo in seguito, in uno user control.
Eccone un elenco in ordine puramente alfabetico:

- Assembly: collega un assembly alla pagina o al controllo utente in cui è inserita.
- Page: permette di specificare attributi per la pagina, che saranno utilizzati per la sua analisi o la compilazione. Utilizzabile solo con pagine aspx.
- Implements: indica che la pagina deve implementare una specifica interfaccia.
- Import: importa degli namespace nella pagina o nel controllo.
- Master: permette di specificare degli attributi per una Master Page, utilizzabile quindi solo con file .master.
- MasterType: permette di creare un riferimento fortemente tipizzato alla pagina master, che sarà accessibile attraver-

so la proprietà Master della classe Page.
- Control: direttiva utilizzabile con gli user control (.ascx) per specificarne degli attributi.
- Register: permette di associare degli alias ai namespace e ai nomi di controlli custom, per utilizzarli all'interno della pagina in maniera più semplice.
- Reference: collega una pagina o uno user control alla pagina o controllo attuale.
- OutputCache: controlla i meccanismi di caching di una pagina o di un controllo utente.
- PreviousPageType: abilita il postback su una pagina proveniente da un'altra pagina dell'applicazione.

Di seguito si darà una breve spiegazione sull'utilizzo e sugli attributi di ognuna di esse, soffermandosi maggiormente sulle direttive più utilizzate.

2.8.1 La direttiva Assembly

La direttiva Assembly permette di collegare un assembly .NET qualunque ad una pagina, consentendo dunque di utilizzare i tipi in esso contenuti dall'interno della pagina stessa.
La direttiva supporta gli attributi Name oppure Src, che definiscono rispettivamente il nome dell'assembly (senza estensione) oppure il file contenente i sorgenti dell'assembly da compilare.
Per esempio se si vuole utilizzare l'assembly MioAssembly.dll si può scrivere la direttiva nel seguente modo:

```
<%@ Assembly Name="MioAssembly" %>
```

Se invece si ha a disposizione solo il sorgente, ad esempio contenuto nel file MioAssembly.vb, è possibile scrivere:

```
<%@ Assembly Src="MioAssembly.vb" %>
```

Gli attribute Name e Src sono mutuamente esclusivi.
Se fosse necessario utilizzarli entrambi, devono essere scritte delle direttive Assembly separate, ognuna contenente uno solo dei due attributi.

2.8.2 La direttiva Page

È sicuramente la direttiva più utilizzata, in quanto permette di impostare il valore di più quaranta di attributi, da utilizzare al momento del parsing e della compilazione di una pagina aspx.
Ad esempio, ogni pagina creata da Visual Studio 2005 si presenterà con una direttiva Page che specifica il linguaggio utilizzato per il codice, la gestione automatica del nome degli eventi, il file contenente il code behind, ed il nome della classe definita nel code behind da cui la pagina deriva:

```
<%@ Page Language="C#" AutoEventWireup="true"

CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

L'elenco seguente da una rapida descrizione degli attributi permessi per la direttiva Page.

AspCompat: Se true permette l'esecuzione della pagina in un single thread apartment, per compatibilità con il vecchio ASP. Ciò degrada le prestazioni, dunque per default è false.

Async: Specifica se la pagina debba essere eseguita in modalità sincrona (se false, come di default) oppure asincrona (true).

AutoEventWireUp: Specifica se gli eventi di pagina debbano essere autowired, cioè se i gestori degli eventi debbano essere denominati automaticamente secondo la convenzione Page_NomeEvento. Per default è true.

Buffer: Abilita il buffering delle risposte http, per default è true

ClassName: Specifica il nome della classe che sarà compilata alla richiesta della pagina.

CodeFile: Specifica il path del file sorgente di code behind che sta dietro la pagina aspx. Tale attributo è utilizzato congiuntamente all'attributo Inherits.

CodeFileBaseClass: Specifica il path del file che contiene la classe da cui deriva la pagina.

CodePage: Indica un identificatore che definisce lo schema di encoding utilizzato per la risposta.
Il valore è un intero, ad esempio 20127 indica la codifica ASCII, 65000 indica Unicode UTF-7.

CompilationMode: Specifica se la pagina debba essere compilata, secondo uno dei valori dell'enumerazione CompilationMode. Per default è Always, quindi la pagina è compilata per default. Con il valore Never le pagina non viene mai compilata, ma se contiene degli script o del codice, restituirebbe un errore. Con il valore Auto è il runtime che si occupa di capire se è necessaria la compilazione.

CompilerOptions: È una stringa contenente le opzioni da passare al compilatore.

ContentType: Specifica il tipo di contenuto http della pagina, secondo gli standard MIME.

Culture: Specifica le impostazioni internazionali, è può essere un ID valido (ad esempio "it-IT" per l'italiano) oppure il valore Auto per eseguire la rilevazione automatica.

Debug: Specifica se la pagina debba essere compilata in modalità di debug, o meno. Per questioni di performance è bene impostare true solo durante lo sviluppo.

Description: Permette di fornire una descrizione della pagina, che comunque verrà ignorata dal parser di ASP.NET.

EnableEventValidation: Abilita la validazione degli eventi al postback, true per default.

EnableSessionState: Abilita il session state, true per default. Può essere anche impostato a ReadOnly per abilitare il session state in modalità di sola lettura, senza permettere alcuna modifica.

EnableTheming: Abilita l'utilizzo dei temi nella pagina, per default è true.

EnableViewState: Indica se mantenere il View State della pagina ai postback. Per default è true.

EnableViewStateMac: Abilita il View State con cifratura, su cui la pagina esegue un MAC (machine authentication check) ai postback, per verificare che non sia stato manomesso dal client.

ErrorPage: Definisce la pagina di errore da eseguire in caso di eccezioni non gestite.

Explicit: Indica se deve essere effettuata la compilazione con l'opzione Option Explicit di Visual Basic impostata a true (è ignorato se non si usa VB come linguaggio).
Per default è false.

Language: Indica il linguaggio utilizzato per i blocchi di script in-

line e per il code behind. Può essere utilizzato un qualunque linguaggio supportato da .NET, in genere uno fra C#, Visual Basic, VJ#, ma uno solo per pagina.

LCID: Definisce l'identificatore di localizzazione della pagina, ed è mutuamente esclusivo con l'attributo Culture.

LinePragmas: Indica al compilatore se utilizzare linee di pragma, cioè delle opzioni per marcare il codice sorgente usate in genere dai debugger.

MaintainScrollPositionOnPostback: Indica se la posizione di scrolling della pagina debba essere mantenuta al postback, per default è false.

MasterPageFile: L'attributo è utilizzato con le content page per indicare il percorso della master page.

ResponseEncoding: Il nome dello schema di encoding utilizzato per la risposta.

SmartNavigation: Indica se la pagina supporta le funzionalità di smart navigation consentite da Internet Explorer 5.5 o superiori. Per default è false.

Src: Specifica il percorso del file sorgente collegato alla pagina e contenente il code behind. In ASP.NET 2.0 l'approccio preferito è comunque l'utilizzo di CodeFile e Inherits.

Strict: Indica se deve essere effettuata la compilazione con l'opzione Option Strict di Visual Basic impostata a true (è ignorato se non si usa VB come linguaggio).
Per default è false.

StyleSheetTheme: Specifica l'identificatore del tema da applicare alla pagina, permettendo ai singoli controlli di sovrascrivere tali impostazioni.

Theme: Specifica l'identificatore del tema da applicare. Se utilizzato senza l'atttributo StyleSheetTheme, il tema specificato sovrascrive eventuali impostazioni effettuate dai singoli controlli.

Title: Specifica il titolo della pagina da renderizzare mediante il tag <title>.

Trace: Indica se il tracing per la pagina debba essere attivato. È false per default.

TraceMode: Specifica come visualizzare i messaggi di tracing, le opzioni possibili sono SortByTime e SortByCategory.

Transaction: Indica se la pagina supporta le transazioni. Per default è Disabled. Gli altri valori possibili sono NotSupported, Supported, Required, e RequiresNew.

UICulture: Specifica le impostazioni internazionali per l'interfaccia grafica. È possibile specificare Auto per la rilevazione automatica.

ValidateRequest: Indica se effettuare operazioni di validazione contro potenziali pericoli derivanti dall'input dell'utente, se una tale possibilità si verifica viene generata una eccezione.
Per default è true.

ViewStateEncryptionMode: Specifica se è come effettuare la cifratura del View State. Per default è Auto, che indica la pos-

sibilità per ogni singolo controllo di richiederla. Può assumere anche i valori Always, e Never.

WarningLevel: Indica il livello dei warning di compilazione della pagina, da 0 a 4, da trattare come errori e quindi fermare la compilazione stessa.

2.8.3 La direttiva Implements

Con la direttiva Implements è possibile implementare una determinata interfaccia nella pagina stessa. La direttiva ha il solo attributo Interface che specifica il nome completo dell'interfaccia da implementare. Per esempio scrivendo la direttiva in una pagina aspx o un controllo ascx, la pagina e il controllo dovranno implementare eventi, proprietà, metodi dell'interfaccia IWebPart qui indicata:

```
<%@ Implements
  interface="System.Web.UI.WebControls.WebParts.IWebPart" %>
```

Utilizzando la direttiva Implements, i membri da implementare non possono essere inseriti nel code behind, ma devono essere inclusi all'interno di blocchi <script> del file aspx.

2.8.4 La direttiva Import

La direttiva Import consente di importare un namespace in una pagina o controllo, ed utilizzare così le classi e le interfacce dichiarate al loro interno, analogamente all'istruzione using di C# o Imports di VB. L'unico attributo
Namespace serve a specificare il namespace da importare, e nel caso in cui sia necessario importarne più di uno, bisognerà utilizzera diverse direttive Import nello stesso file:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
```

Con le due direttive precedenti sarà possibile utilizzare un qualsiasi tipo dei namespace indicati, senza dovere specificare il nome completo. Ad esempio si potrà scrivere:

```
SqlConnection connection;
```

invece di:

```
System.Data.SqlClient.SqlConnection connection;
```

2.8.5 La direttiva Master

Le master page, definite in file .master, che vedremo più avanti nel dettaglio, possono essere configurate mediante la direttiva Master in maniera analoga a quanto visto per la direttiva Page usata nelle normali pagine aspx..
Ogni pagina di contenuto, detta content page, che è una pagina normale e dunque con una direttiva Page, può ereditare ed usare una parte di contenuto comune, definito nella pagina Master, grazie all'attributo MasterPageFile visto a proposito della direttiva Page.
Ogni file .master può contenere una sola direttiva Master, all'interno della quale può essere specificato un sottoinsieme di attributi rispetto a quelli della direttiva Page.
Anche per una master page può essere specificato un attributo MasterPageFile, e questo perché una pagina Master può a sua volta utilizzare un'altra pagina Master, creando così delle master page nidificate una dentro l'altra.
Un possibile utilizzo della direttiva è il seguente:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

2.8.6 La direttiva MasterType

Grazie alla direttiva MasterType, è possibile dare una tipizzazione

forte alla pagina master di una pagina di contenuto, ed in tal modo la pagina master potrà essere ricavata semplicemente accedendo alla proprietà Master della classe Page.
La direttiva MasterType ha due possibili attributi, utilizzabili uno solo per volta.
Il primo è TypeName che permette di specificare il nome completo della classe che implementa la pagina Master, il secondo, VirtualPath permette invece di specificarne il percorso virtuale.
Ad esempio:

```
<%@ MasterType VirtualPath= "~/MasterPage.master"%>
```

La pagina di contenuto che contiene questa direttiva, potrà riferirsi alla sua pagina master, implementata nel file MasterPage.master specificato, tramite la proprietà Master, ed usarne quindi metodi e proprietà:

```
protected void Page_Load(object sender, EventArgs e)

{
    Label1.Text = "La mia pagina master è " + Master.ToString();

}
```

2.8.7 La direttiva Control

Con la direttiva Control è possibile specificare gli attributi di uno user control, implementato in un file ascx, in maniera molto simile all'utilizzo della direttiva Page utilizzata con le pagine aspx.
Gli attributi possibili per la direttiva Control sono un sottoinsieme di quelli visti per la direttiva Page, ed hanno praticamente lo stesso significato a livello del singolo controllo.
Ecco l'elenco, rinviando al paragrafo sulla direttiva Page, per una loro descrizione:

- AutoEventWireUp
- ClassName
- CodeFile
- CodeFileBaseClass
- CompilationMode
- CompilerOptions
- Debug
- Description
- EnableTheming
- EnableViewState
- Explicit
- Inherits
- Language
- LinePragmas
- Src
- Strict
- WarningLevel

Il seguente è un esempio di possibile utilizzo della direttiva Control in uno user control ASP.NET:

```
<%@ Control Language="C#" AutoEventWireup="true"

CodeFile="WebUserControl.ascx.cs" Inherits="WebUserControl"
Description="Un controllo utente"  %>
```

2.8.8 La direttiva Register

La direttiva Register permette di creare un'associazione fra un cosiddetto tag prefix ed uno user control, analogamente a quando si crea un alias per un namespace o per una classe.
In questa maniera è possibile utilizzare un custom control chiamandolo con un nome specifico.
La direttiva Register permette di utilizzare cinque possibili attributi:

Attributo	Descrizione
Assembly	L'assembly contenente il namespace da associare al TagPrefix scelto.
Namespace	Il namespace da associare al TagPrefix.
Src	Il percorso relativo o assoluto del file ascx che implementa il controllo.
TagName	Un alias da associare con il nome della classe.
TagPrefix	Un alias da associare al namespace.

L'esempio seguente mostra un utilizzo della direttiva Register per registrare un controllo WebUserControl in una pagina aspx:

```
<%@ Register Src="WebUserControl.ascx"
 TagName="WebUserControl"
TagPrefix="uc1" %>
```

In questa maniera il controllo potrà essere incluso in qualsiasi punto della pagina, utilizzando un tag come il seguente:

```
<uc1:WebUserControl ID="WebUserControl1" runat="server" />
```

2.8.9 La direttiva Reference

La direttiva Reference dichiara che una pagina, uno user control, o un altro file, deve essere compilato e linkato alla pagina o controllo in cui la direttiva è utilizzata.
Sono utilizzabili tre attributi. L'attributo Page specifica la pagina che deve essere compilata insieme al file che contiene la direttiva Reference.

L'attributo Control specifica invece un controllo da compilare. Infine l'attributo VirstualPath permette di specificare il percorso di un file a cui si fa riferimento e che è quindi da compilare. Gli esempi seguenti mostrano dei possibili utilizzi della direttiva:

```
<%@ Reference Page="~/pagina.aspx">
<%@ Reference Control="~/WebUserControl.ascx">
<%@ Reference VirtualPath="~/Folder/helloworld.aspx" %>
```

2.8.10 La direttiva OutputCache

Con la direttiva OutputCache è possibile controllare i meccanismi di caching di una pagina aspx o di un controllo. Gli attributi utilizzabili con la direttiva sono elencati in tabella 2.1 L'attributo Duration è obbligatorio, ed indica la durata in secondi per cui l'elemento è mantenuto nella cache. La direttiva seguente inserisce nella cache una pagina od uno user control con un timeout di 60 secondi, ed inoltre ogni richiesta http che arriva con un parametro"nomeparametro" uguale viene soddisfatto con l'elemento incluso nella cache. Se il parametro è diverso la nuova pagina o controllo viene creato rinfrescando la cache, con una nuova durata di 60 secondi.

```
<%@ OutputCache Duration="60" VaryByParam="nomeparametro" %>
```

2.8.11 La direttiva PreviousPageType

La direttiva PreviousPageType permette di sfruttare la nuova caratteristica di postback cross-page, cioè il postback fra pagine diverse, introdotta in ASP.NET 2.0.

È possibile specificare la pagina di origine da cui si verifica l'evento di postback utilizzando uno dei due attributi possibili in questa direttiva. L'attributo TypeName permette di definire il nome della classe da cui viene originato il postback, mentre l'attributo VirtualPath permette di specificare il percorso virtuale della pagina che genera il postback.

Attributo	Descrizione
CacheProfile	Il nome del profilo di caching da utilizzare così come definito nel file web.config, per default è " ".
DiskCacheable	Specifica se la cache può essere memorizzata su disco.
Duration	Attributo obbligatorio, che indica la durata del contenuto della cache.
Location	Specifica la posizione della cache, con uno dei valori dell'enumerazione OutputCacheLocation, per default è "Any". È utilizzabile solo in file aspx.
NoStore	Specifica che la pagina non deve, essere memorizzata in cache.
SqlDependency	Abilita una nuova funzionalità di ASP.NET 2.0, con la possibilità di specificare la dipendenza della cache dal contenuto di una tabella di database
VaryByControl	Una lista di stringhe separate da; che permette di definire politiche di caching separate per i controlli inclusi nella lista
VaryByCustom	Permette di definire la politica di caching in maniera personalizzata, ad esempio

Attributo	Descrizione
VaryByCustom	in base al browser utilizzato per una richiesta.
VaryByHeader	Specifica una lista di header HTTP separati da ; utilizzati per modificare la cache.
VaryByParam	Specifica una lista di parametri separati da ; che corrispondono a una query string spedita con una richiesta GET o ai parametri di una POST e che vengono utilizzati per variare la cache.

Tabella 2.1

I due attributi sono mutuamente esclusivi.
L'esempio seguente specifica che la pagina precedente di quella corrente è definita nel file SourcePage.aspx che si trova nella directory radice:

```
<%@ PreviousPageType VirtualPath="~/SourcePage.aspx"%>
```

Utilizzando la proprietà PreviousPage dalla pagina sarà possibile ottenere un riferimento alla pagina che ha originato il postback.

I CONTROLLI SERVER DI ASP.NET

Una pagina ASP.NET, oltre a tag HTML, puro testo, e codice inline, può contenere dei server controls, cioè degli elementi programmabili dal lato del server e che tipicamente rappresentano elementi dell'interfaccia grafica, come una casella di testo, o un pulsante.
Come si è già mostrato, la chiave per la programmazione lato server in ASP.NET è l'attributo runat.
Grazie ad esso, un tag presente in una pagina aspx viene mappato su una classe server-side, e processato quindi come controllo server durante la costruzione della pagina, per generare del codice HTML che verrà poi rispedito al client, ed infine visualizzato sul browser.

3.1 LA CLASSE CONTROL

Ogni controllo server, deriva dalla classe System.Web.UI.Control, che fornisce le funzionalità minime che ogni controllo deve possedere, e quindi proprietà, metodi ed eventi comuni.
Se invece è necessario implementare un controllo che deve presentare informazioni all'utente, quindi con una interfaccia grafica, è meglio derivare la sua classe da WebControl o comunque da uno dei controlli del namespace System.Web.UI.WebControls, che forniscono un punto di partenza più appropriato.
La classe Control è la classe base anche per le pagine aspx, che derivano dalla classe Page, la quale è a sua volta figlia di Control.

3.1.1 Proprietà di Control

Le proprietà di Control non hanno alcuna funzionalità legata all'interfaccia grafica, ma rappresentano l'insieme di proprietà comuni ad ogni controllo server, siano esse di tipo grafico o meno.
Il seguente elenco mostra quelle principalmente utilizzate per un controllo qualunque fornito dalla libreria ASP.NET o per l'implementazione di un controllo derivato da Control.

- AppRelativeTemplateSourceDirectory: La directory relativa alla radice dell'applicazione dell'oggetto Page o UserControl che contiene il controllo. Se ad esempio la pagina risiede in http://www.sito.com/application/subdir la proprietà restituisce ~/subdir.

- BindingContainer: Il controllo padre che contiene le informazioni di data-binding per il controllo corrente.

- ClientID: L'identificatore univoco del controllo generato da ASP.NET. Esso è in genere la combinazione della proprietà ID del controllo contenitore e della proprietà UniqueID, sostituendo eventuali caratteri non permessi come '$' con l'underscore.

- Controls: La collezione di controlli figli del controllo corrente.

- EnableTheming: Indica se bisogna applicare dei temi al controllo.

- EnableViewState: Utilizzata per indicare se deve essere memorizzato il view-state, in parole povere lo stato attuale, del controllo fra una richiesta e l'altra della pagina.

- ID: Restituisce o imposta il nome del controllo, che può essere utilizzato programmaticamente per utilizzarlo nel codice della pagina.

- NamingContainer: Restituisce il NamingContainer del controllo, cioè il controllo contenitore del quale viene utilizzato l'ID per creare un namespace univoco nel caso di controlli con lo stesso nome nella pagina.

- Page: È la pagina che ospita il controllo.
- Parent: Se il controllo è contenuto a sua volta in un server control, questa proprietà lo restituisce.
- Site: È il contenitore che ospita fisicamente il controllo renderizzato sulla pagina.
- SkinID: Restituisce o imposta il nome dello Skin utilizzato per il controllo.
- TemplateControl: È il template contenitore del controllo
- TemplateSourceDirectory: Il percorso della directory virtuale in cui è inserita la pagina o lo UserControl contenitore del controllo corrente. Se ad esempio la pagina risiede in http://www.sito.com/application/subdir la proprietà restituisce application/subdir
- UniqueID: L'identificatore univoco e gerarchico del controllo, ottenuto componendo gli ID del NamingContainer e del controllo stesso, separati in genere dal carattere '$' (il carattere separatore è ottenuto dalla proprietà IdSeparator).
- Visible: Indica se il controllo deve essere visualizzato sulla pagina.

3.2 I controlli HTML lato server

I controlli HTML lato server, ad un primo approccio, sono esattamente uguali ad un tag HTML classico, con l'unica differenza dell'attributo runat. In Visual Studio è possibile aggiungere un controllo HTML ad una pagina, trascinandolo dalla Toolbox, ed esattamente dalla categoria HTML.

Aggiungendo ad esempio un controllo Button, ed andando ad osservare il codice html generato, dovrebbe verificarsi che nel codice della pagina è stato aggiunto il tag seguente:

```
<input id="Button1" type="button" value="button" />
```

Questo è un classico pulsante HTML, non è un controllo server.
È possibile renderlo un controllo lato server, semplicemente aggiungendo l'attributo runat="server":

```
<input id="Button1" runat="server" type="button" value="button" />
```

La stessa cosa può essere ottenuta semplicemente andando in modalità Design, e facendo clic con il tasto destro sul pulsante qui visualizzato. Dal menù contestuale bisogna solo fare clic su Run as Server Control, ciò farà in modo che venga aggiunto l'attributo runat mentre graficamente il pulsante potrà essere riconosciuto come un controllo server da una freccetta verde.
Il controllo adesso potrà essere utilizzato programmaticamente in maniera totalmente orientata agli oggetti.
Ad esempio per impostare il testo del pulsante, basterà semplicemente utilizzare la proprietà Value. Nel Page_Load della pagina si può dunque scrivere:

```
protected void Page_Load(object sender, EventArgs e)

{

this.Button1.Value = "Clicca qui";
}
```

Eseguendo la pagina si noterà che il pulsante avrà effettivamente il nuovo testo come etichetta.

3.2.1 La classe HtmlControl

Il .NET Framework fornisce dei controlli server predefiniti per tutti i comuni tag HTML, ad esempio body della pagina, input, tabelle, immagini, collegamenti ipertestuali.
Ognuno di essi deriva dalla classe HtmlControl, ed ognuno di essi naturalmente fornisce le proprietà per manipolare da codice lato server gli attributi html.
La classe HtmlControl deriva anch'essa da Control, aggiungendo a quest'ultima le seguenti proprietà specifiche:

- Attributes: la collezione che rappresenta gli attributi del controllo con i relativi valori.
- Disabled: indica se il controllo è disabilitato o meno.
- Style: una collezione di oggetti che rappresenta gli stili CSS applicati al controllo.
- TagName: il nome del tag html.

La collezione Attributes permette non solo di impostare i valori degli attributi di un tag html, ma anche di utilizzare quei tag html che non hanno un corrispondente predefinito in ASP.NET, oppure di impostare proprietà non accessibili tramite l'interfaccia della classe.
Ad esempio se si aggiunge l'attributo runat al tag <body> della pagina ed un identificatore per utilizzarlo via codice, in questa maniera:

```
<body runat="server" id="body">
 …
</body>
```

sarà possibile impostare un attributo qualunque programmaticamente, anche non accessibile tramite una proprietà diretta di HtmlControl.

Per cambiare ad esempio il colore di sfondo della pagina, in html è possibile utilizzare nel tag <body> l'attributo bgcolor.
Utilizzando la collezione Attributes, possiamo ottenere lo stesso effetto semplicemente aggiungendo bgcolor ed il suo valore alla collezione:

```
body.Attributes["bgcolor"]="red";
```

Tale assegnazione non farà altro che generare il tag body con l'attributo bgcolor ed il suo valore:

```
<body id="body" bgcolor="red">
```

Bisogna ricordare che gli elementi della collezione Attributes sono sempre e comunque trattati come stringhe.

3.2.2 Gerarchia dei controlli HTML

Quasi tutti i controlli server HTML di ASP.NET, non discendono direttamente dalla classe HtmlControl, ma da ulteriori sotto classi, che li raggruppano in due categorie, cioè i controlli di input e quelli contenitori.
Le classi derivate da HtmlControl sono dunque HtmlInputControl ed HtmlContainerControl, che rappresentano le due categorie, e le classi HtmlImage, HtmlLink, ed HtmlTitle, HtmlMeta, non raggruppabili in nessuna delle due categorie menzionate. I controlli di input, derivati da HtmlInputControl sono riassunti nell'elenco seguente, (tabella pag.63) insiemi ai corrispondenti tag HTML.
Si può notare che tutte le classi rappresentano le possibili varianti del tag <input>, configurabili tramite l'attributo type.
La classe madre HtmlInputControl espone tre proprietà specifiche per trattare i controlli appena elencati, oltre a quelle derivate dalle classi HtmlControl e Control.
La proprietà Name rappresenta il nome del controllo, o meglio il suo

Classe	Tag
HtmlInputButton	
HtmlInputCheckBox	<input type="checkbox" />
HtmlInputFile	<input type="file" />
HtmlInputHidden	<input type="hidden" />
HtmlInputRadioButton	<input type="radio" />
HtmlInputText	<input type="text" />
HtmlInputReset	<input type="reset" />
HtmlInputSubmit	<input type="submit" />
HtmlInputPassword	<input type="password" />

identificatore univoco, infatti essa restituisce attualmente il valore della proprietà UniqueID derivata da Control.
Da notare che sebbene sia una proprietà accessibile anche in scrittura, l'eventuale valore dato in ingresso non verrà considerato, il motivo di tale scelta implementativa è proprio il fatto che i valori UniqueID e Name devono coincidere per garantire il corretto funzionamento. La proprietà Type restituisce il tipo del controllo di input, così come visto nella tabella precedente, sotto forma di stringa, ad esempio per un controllo HtmlInputText la proprietà Type restituirà la stringa "text", per un HtmlInputFile essa darà invece "file" e così via. Infine, la proprietà Value rappresenta il valore attuale del controllo di input, e dunque il suo contenuto.

Pulsanti

Per quanto riguarda il controllo HtmlInputButton è possibile gestirne il clic lato server, gestendo l'evento OnServerClick.
Per fare ciò basta aggiungere un attributo OnServerClick ed impostarne il valore al nome del metodo che appunto gestirà l'evento, sia esso scritto nel code behind che contenuto in un tag script nella pagina stessa, ad esempio:

```
<input id="Button1" type="button" value="button" runat="server"
```

```
onserverclick="Button1Click" />
```

In questo caso al clic sul pulsante sarà invocato il metodo Button1Click, che utilizzando il codice inline potrebbe essere scritto in questo modo:

```
<script type="text/C#" runat=server>
 void Button1Click(object sender, EventArgs e)

{
 Button1.Value="Cliccato";

}
</script>
```

Il terzo tipo di pulsante HTML, HtmlInputReset, al clic su di esso, automaticamente riporta lo stato dei controlli contenuti nella pagina a quello originale, cioè quello che avevano al caricamento della pagina stessa.

I controlli contenitori

I controlli contenitori, derivati da HtmlContainerControl sono invece i seguenti:

- HtmlAnchor
- HtmlButton
- HtmlForm
- HtmlGenericControl
- HtmlSelect
- HtmlTable
- HtmlTableCell
- HtmlTableRow
- HtmlTextArea

I controlli contenitori sono dunque quei controlli corrispondenti ad elementi HTML che richiedono sia un tag di apertura che un tag di chiusura.
Per esempio per creare un collegamento ipertestuale in HTML si deve scrivere qualcosa del genere:

```
<a href="pagina.aspx">testo</a>
```

Esiste poi la classe HtmlHead, che non è figlia diretta di HtmlContainerControl, ma deriva a sua volta da HtmlGenericControl.
Essa permette l'accesso all'elemento <head> di una pagina dal lato del server.
Per utilizzare una qualunque delle precedenti classi, si è già visto che l'operazione fondamentale è l'aggiunta di un attributo runat al corrispondente tag HTML.
Ad esempio per accedere al tag title basta scrivere:

```
<title id="TitoloPagina" runat="server">
```

Ed accedere dunque da codice all'elemento title utilizzando la classe HtmlTitle e le sue proprietà:

```
TitoloPagina.Text="Titolo della pagina";
```

Le proprietà fondamentali della classe HtmlContainerControl e delle classi dunque da essa derivate, sono InnerHtml e InnerText.
Entrambe permettono di impostare il codice HTML contenuto fra i tag di apertura e chiusura, con una fondamentale differenza.
Nel caso della proprietà InnerHtml, se il codice contiene caratteri speciali, ad esempio altri tag HTML, esso verrà inviato al browser senza alcuna codifica, ma così com'è.
Per esempio, si consideri il seguente codice nella pagina HTML, che definisce un collegamento ipertestuale:

```
<a href=htmlcontrols.aspx id="IdCollegamento1"
 runat=server>collegamento</a>
```

Il tag sarà accessibile lato server utilizzando la classe HtmlAnchor corrispondente al tag <a>, ed impostando la proprietà HTML così:

```
IdCollegamento1.InnerHtml="<b>Testo del collegamento</b>";
```

Il testo contenuto tra tag di apertura e chiusura, in particolare i caratteri speciali < e >, non sarà codificato, ma interpretato appunto come HTML e dunque il testo sarà visualizzato in grassetto.

3.3 I WEBCONTROL

I controlli web di ASP.NET sono l'approccio alternativo agli appena visti controlli HTML, e sono più potenti e flessibili in quanto non legati appunto a specifici tag HTML, anzi introducono delle funzionalità non presenti nell'HTML classico, basti pensare ad esempio al controllo calendario, ed inoltre sono personalizzabili per rispondere a qualsiasi necessità, sia creando dei controlli compositi, cioè formati da più controlli web, oppure creando dei controlli totalmente ex-novo, se la funzionalità cercata non fosse presente nei controlli forniti da ASP.NET ASP.NET 2.0 inoltre introduce una serie di nuovi controlli non presente nelle precedenti versioni, ma i controlli precedenti sono ancora utilizzabili senza alcun problema.
Come qualunque controllo, i controlli web sono utilizzabili lato server grazie all'attributo runat.
Ogni controllo ASP.NET standard utilizza un tag che inizia con <asp:, ad esempio il webcontrol TextBox, per creare una casella di testo, può essere inserito in una web form nel seguente modo:

```
 <asp:TextBox ID="TextBox1" runat="server"
Text="Testo"></asp:TextBox>
```

Come i controlli HTML anche in questo caso è necessario, per poter referenziare il controllo da codice, un ID, oltre all'ormai immancabile runat="server".
Tutti i controlli web, inoltre, devono avere un tag di apertura e chiusura, anche se in questo caso, o in genere quando il controllo web non prevede altri elementi al suo interno, il tag di chiusura può essere incluso all'interno dello stesso elemento di apertura:

```
<asp:TextBox ID="TextBox1" runat="server" Text="Testo"/>
```

3.3.1 La classe WebControl

Ogni controllo web deriva dalla classe WebControl, che a sua volta deriva dalla già citata Control. La classe WebControl fornisce diverse proprietà che permettono di gestire ogni aspetto del controllo, soprattutto dal punto di vista del layout e dell'aspetto, e dunque font, colori, fogli di stile, e così via. La tabella seguente mostra le proprietà della classe WebControl (tabella pag.69). Utilizzando le proprietà elencate è possibile modificare a proprio piacimento l'aspetto grafico del controllo web.
Ad esempio, dato il precedente controllo TextBox1, è possibile definirne il colore di sfondo, quello del testo, ed assegnare un ToolTip da codice, nel seguente modo:

```
TextBox1.BackColor = Color.Yellow;
TextBox1.ForeColor = Color.Blue;
TextBox1.ToolTip = "un tooltip";
```

Oppure ottenere lo stesso effetto utilizzando gli attributi sulla pagina HTML:

```
<asp:TextBox ID="TextBox1" runat="server" Text="Testo"

BackColor="Yellow" ForeColor="Blue" ToolTip="un tooltip"/>
```

I metodi pubblici della classe WebControl, non derivati da Control, consentono ulteriori possibilità di personalizzazione dello stile, ma che vengono raramente utilizzati da uno sviluppatore di applicazioni web, e lasciate invece a chi si occupa di creare nuovi controlli.

3.3.2 I controlli web standard

Di seguito vengono analizzati, mediante semplici esempi, i controlli standard presenti in genere nella maggior parte delle pagine di un'applicazione web ASP.NET, rinviando ai successivi capitoli, invece, per controlli dedicati a particolari aree tematiche, come l'accesso ai database, la validazione delle pagine, e così via.

Il controllo Label

Il controllo Label rappresenta la classica etichetta di testo, con la facoltà, come per tutti i controlli server, di impostare il testo anche dal lato server, cioè dal code behind. È implementato tramite il tag span. Un controllo Label viene inserito in una pagina aspx, utilizzando il tag <asp:Label>:

```
<asp:Label id="Label1" runat="server" Text="Testo
 dell'etichetta"></asp:Label>
```

È possibile impostare il testo, oltre che tramite l'attributo Text, inserendolo fra i tag di apertura e chiusura:

```
<asp:Label id="Label1" runat="server">
 Testo dell'etichetta <asp:Label>
```

Oppure naturalmente da codice:

```
Label1.Text=" Testo dell'etichetta";
```

Il controllo Literal

Il controllo Literal è molto simile al controllo Label, in quanto permette

Proprietà	Accesso	Descrizione
AccessKey	get, set	La lettera da usare insieme al tasto ALT come scorciatoia da tastiera per dare il focus al controllo.
Attributes	get	La collezione di attributi che non corrispondono alle proprietà del controllo e che verranno renderizzati come attributi HTML sulla pagina.
BackColor	get, set	Il colore di sfondo del controllo
BorderColor	get, set	Il colore del bordo
BorderStyle	get, set	Lo stile del bordo
BorderWidth	get, set	La dimensione del bordo
ControlStyle	get	Lo stile del controllo, oggetto di tipo Style
ControlStyleCreated	get	Indica se la proprietà ControlStyle è stata imposta con un oggetto Style
CssClass	get, set	Il nome della classe CSS associata al controllo
Enabled	get, set	Indica se il controllo è abilitato
Font	get	Le proprietà del carattere usato nel controllo
ForeColor	get, set	Il colore utilizzato per il testo
Height	get, set	L'altezza del controllo
Style	get	Una collezione di CssStyle che saranno renderizzati in HTML con l'attributo style
TabIndex	get, set	L'indice di tabulazione
ToolTip	get, set	Il testo visualizzato quando il puntatore del mouse è posizionato sul controllo
Width	get, set	La larghezza del controllo

Tabella: Proprietà della classe WebControl

di inviare al browser un semplice testo da visualizzare.
La differenza è che Literal deriva direttamente da Control e mentre il controllo Label viene implementato per mezzo di un tag span, un Literal invia il testo senza alcun tag, mostrando dunque il puro testo.

```
<asp:Literal id="Literal1" runat="server" Text="testo"></asp:Literal>
```

Il controllo TextBox

Il controllo TextBox è utilizzato per visualizzare una classica casella di testo.
L'utilizzo principale è quello che permette di creare la casella di testo su una singola linea:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

Se il testo supera la larghezza della casella, automaticamente verrà effettuato lo scrolling orizzontale.
È possibile specificare la lunghezza massima del testo inseribile, impostando la proprietà MaxLength:

```
<asp:TextBox ID="TextBox1" runat="server"

MaxLength="20"></asp:TextBox>
```

Oppure da codice C#, ad esempio:

```
TextBox1.MaxLength=20;
```

Se si vuole utilizzare la TextBox in modalità a sola lettura, impedendo quindi l'inserimento da parte dell'utente, è possibile impostare a true la proprietà ReadOnly.

La proprietà TextMode permette di variare il comportamento e l'aspetto della casella, impostando uno dei valori dell'enumerazione TextBoxMode.
Se non specificato, esso è per default impostato a SingleLine, in cui ricade il caso precedente.
Se si vuole utilizzare la casella di testo per inserire una password, è necessario utilizzare il valore Password per la proprietà TextMode:

```
<asp:TextBox ID="TextBox2" runat="server"
TextMode="Password"></asp:TextBox>
```

In questo caso i caratteri digitati vengono mascherati con un carattere * o con il punto, e naturalmente il valore digitato non viene conservato nel ViewState della pagina, per questioni di sicurezza.
Il terzo caso possibile permette di creare caselle di testo con più linee, tramite il valore MultiLine.
Per impostare il numero di colonne e righe è invece necessario utilizzare le proprietà Columns e Rows, e se si vuole che il testo vada automaticamente a capo raggiunta la larghezza della casella, è possibile impostare a true la proprietà Wrap, in caso contrario la casella scrollerà automaticamente.
Per scrivere più righe di testo è necessario separarle tramite le sequenze di escape "\r\n":

```
TextBox3.Text = "linea1\r\nlinea2\r\nlinea3";
```

È possibile rispondere agli eventi di modifica del contenuto di una TextBox. Ad esempio l'evento TextChanged si verifica quando viene modificata la proprietà Text ed il controllo perde il focus.
L'evento non viene sollevato immediatamente, ma al successivo PostBack della pagina.
Se si vuole invece un PostBack immediato, è possibile impostare a true la proprietà AutoPostBack.

```
<asp:TextBox ID="TextBox1" runat="server" AutoPostBack=true
OnTextChanged="TextBox1_TextChanged">
```

Nell'esempio seguente, al PostBack, che si verificherà al momento in cui la casella TextBox1 perde il focus, il testo viene convertito in maiuscolo:

```
protected void TextBox1_TextChanged(object sender, EventArgs e)
{
    TextBox1.Text = TextBox1.Text.ToUpper();
}
```

Il controllo Button

Insieme alla TextBox, il controllo Button è uno dei controlli web più utilizzati in qualsiasi applicazione web.
La classe Button rappresenta il classico pulsante, implementato tramite il tag input. L'etichetta del pulsante viene impostata tramite la proprietà Text, derivata da Control:

```
<asp:Button ID="Button1" Text="Invia" runat="server" />
```

L'utilizzo più semplice è naturale del controllo Button è quello che prevede la gestione dell'evento Click, tanto che esso è l'evento predefinito in Visual Studio, dove, nella finestra di design se si fa doppio clic sul pulsante, viene automaticamente generato il gestore dell'evento, nella seguente maniera:

```
protected void Button1_Click(object sender, EventArgs e)
{
}
```

Mentre all'interno del tag asp:Button verrà aggiunto l'attributo OnClick:

```
<asp:Button ID="Button1" runat="server" Text="Invia"

OnClick="BtSalva_Click" />
```

In questo caso il controllo Button viene utilizzato per il semplice "submit" della pagina, cioè come un pulsante che effettua il postback della pagina che lo contiene inviandola al server.
Un secondo modo di utilizzo è quello che prevede di associare al pulsante un "command", e dunque in questo caso ci sono altre proprietà della classe Button che possono tornare utili.
La proprietà CommandName permette di associare ad ogni pulsante della pagina aspx un nome, attraverso il quale sarà ad esempio possibile distinguere il pulsante che ha generato un dato evento, in genere l'evento Command.
Supponendo che la web form contenga due pulsanti, è possibile assegnare ad essi un CommandName diverso ma lo stesso gestore dell'evento, chiamato Button_Command:

```
<asp:Button ID="BtOrdina" Text="Ordina" runat="server"

CommandName="Ordina" OnCommand="Button_Command" />

<asp:Button ID="BtSalva" Text="Salva" runat="server"

CommandName="Salva" OnCommand="Button_Command" />

 <br /><br />

 <asp:Label ID="Label1" runat="server"></asp:Label>
```

Si potrà distinguere il pulsante su cui l'utente ha faccto clic, per esempio semplicemente con una istruzione switch, ed eseguendo l'operazione voluta corrispondente:

```
protected void Button_Command(object sender, CommandEventArgs e)
{
    Button bt=sender as Button;
    if(bt==null)
        return;
    switch (bt.CommandName)
    {
        case "Salva":
            Label1.Text = "Hai cliccato il pulsante Salva";
            break;
        case "Ordina":
            Label1.Text = "Hai cliccato il pulsante Ordina";
            break;
    }
}
```

Generalmente assieme a CommandName, ma non è obbligatorio, può essere utilizzata la proprietà CommandArgument, che serve ad associare un argomento al pulsante.
Ad esempio un Button che serva ad effettuare l'ordinamento degli elementi di una lista di stringhe, potrebbe avere la proprietà CommandName impostata a "Ordina", mentre un possibile CommandArgument potrebbe essere il valore "Alfabetico", oppure "Lunghezza", il primo per ordinare alfabeticamente le stringhe, il secondo invece in base alla lunghezza.
È possibile, oltre che eseguire del codice sul server, grazie all'evento Click visto finora, eseguire anche del codice lato client, utilizzando questa volta l'attributo OnClientClick e definendo una funzione da eseguire, per esempio scritta in JavaScript.
Gli attributi OnClick ed OnClientClick sono utilizzabili contemporaneamente.
Nell'esempio che segue, al clic sul pulsante viene eseguito sia il codice definito nel gestore Button1_Click, che la funzione ShowAlert:

```
<script type="text/javascript">
function DisableButton(ctl)
{
alert('attendi');
}
</script>

<html>
<head runat="server"><title>Pagina</title></head>
<body>
<form id="form1" runat="server">
<asp:Button ID="Button1" runat="server"
OnClientClick="DisableButton('Button1')" OnClick="Button1_Click"

Text="Button" />
</form>
</body>
</html>
```

Un'altra proprietà interessante del controllo Button, novità di ASP.NET 2.0, permette di eseguire un postback su una pagina diversa (o come viene chiamato un postback crosspage) da quella che contiene il controllo.
Per ottenere tale obiettivo basta impostare la proprietà PostBackUrl, con l'indirizzo della pagina destinazione.

Il controllo ImageButton

Un altro controllo simile al precedente Button è il controllo ImageButton, che permette di creare un pulsante con l'aspetto di un'immagine. Quest'ultima viene impostata tramite la proprietà ImageUrl:

```
<asp:ImageButton ID="ImageButton1" runat="server"
ImageUrl="~/Images/redgreenyellow.gif"/>
```

L'immagine visualizzata sul controllo ImageButton, può essere utilizzata come una mappa, determinando dunque le coordinate del punto cliccato con il mouse.

Il gestore dell'evento Click utilizza infatti come argomento un oggetto ImageClickEventArgs, invece di EventArgs, da cui è possibile ricavare tramite le proprietà X e Y le coordinare del punto espresse in pixel, con il punto d'origine (0,0) situato nell'angolo superiore sinistro dell'immagine. In questo caso, è sufficiente la sola X per determinare il colore e scrivere sulla pagina un messaggio con il colore scelto:

```
protected void ImageButton1_Click
(object sender, ImageClickEventArgs e)
{
    if (e.X < 101)
    Response.Write("Hai cliccato il rosso");
    else if (e.X < 201)
    Response.Write("Hai cliccato il verde");
    else Response.Write("Hai cliccato il giallo");
}
```

Il controllo LinkButton

Il controllo LinkButton è funzionalmente identico al Button classico appena visto, con l'unica differenza nell'aspetto, in quanto viene visualizzato come un collegamento ipertestuale sulla pagina.
Sono dunque supportate tutte le proprietà viste per il controllo Button, e preferire il suo utilizzo è solo questione di gusto grafico, o di volontà di mettere ordine quando si hanno molti controlli Button sulla stessa pagina.
Per creare un LinkButton si potrebbe scrivere:

```
<asp:linkbutton ID="LinkButton1" Text="Clicca" runat="server"/>
```

Il controllo HyperLink

La classe HyperLink permette di aggiungere ad una pagina web il classico collegamento ipertestuale, collegando dunque un testo oppure un'immagine ad una pagina di destinazione.
L'esempio seguente utilizza un controllo Button per creare un hyperlink, che punta alla pagina indicata da una TextBox, e con un'immagine indicata da una seconda TextBox.
Gli HyperLink creati vengono poi aggiunti ad un contenitore PlaceHolder, semplicemente inserendoli nella sua collezione Controls.
La pagina HTML è fatta così:

```
<asp:TextBox ID="TextBox1"
 runat="server">~/default.aspx</asp:TextBox><br />
 <asp:TextBox ID="TextBox2"

 runat="server">~/images/go.gif</asp:TextBox><br />
 <asp:Button ID="btAggiungi" runat="server"
 OnClick="btAggiungi_Click" Text="Aggiungi" />
 <br />
 <asp:PlaceHolder ID="PlaceHolderLinks"
  runat="server"></asp:PlaceHolder>
```

L'evento Click del pulsante btAggiungi invece effettua l'operazione di creazione e di inserimento nel PlaceHolder:

```
 protected void btAggiungi_Click(object sender, EventArgs e)
 {

     HyperLink link = new HyperLink();
     link.NavigateUrl = TextBox1.Text;
     link.ImageUrl = TextBox2.Text;
     PlaceHolderLinks.Controls.Add(link);
 }
```

Il controllo DropDownList

Il controllo DropDownList rappresenta la classica casella combinata, quella che nel mondo delle applicazioni Windows Forms è la ComboBox, e che consente di selezionare un elemento singolo da un elenco a discesa.

La procedura per aggiungere degli elementi ad un controllo come DropDownList, vale per tutti i controlli web che rappresentano liste di elementi, vale a dire, oltre a DropDownList, i controlli ListBox, CheckBoxList, RadioButtonList, BulletedList.

Gli elementi sono rappresentati da oggetti ListItem, ognuno dei quali è caratterizzato da una proprietà Text che rappresenta il testo visualizzato in corrispondenza dell'elemento, una proprietà Value che rappresenta un eventuale valore corrispondente, ed una proprietà Selected che stabilisce l'elemento selezionato nella lista.

La lista di elementi può essere definita in maniera statica, a design-time, da codice Html basta creare dei tag ListItem, con le proprietà desiderate:

```
<asp:DropDownList ID="DropDownList1" runat="server">
        <asp:ListItem Value="BO" Text="Bologna"

Selected="True"></asp:ListItem>
        <asp:ListItem Value="CT" Text="Catania"></asp:ListItem>
        <asp:ListItem Value="MI" Text="Milano"></asp:ListItem>
</asp:DropDownList>
```

La seconda possibilità per popolare la lista di elementi è da codice, ottenuta aggiungendo gli elementi alla collezione Items della classe DropDownList:

```
 ListItem item = new ListItem(txtProvincia.Text, txtSigla.Text);
DropDownList1.Items.Add(item);
DropDownList1.SelectedIndex = DropDownList1.Items.Count-1;
```

L'elemento viene creato a partire dal contenuto di due TextBox, quindi aggiunto alla collezione degli elementi, e mostrato come elemento selezionato, grazie alla proprietà SelectedIndex, che viene impostata all'indice dell'ultimo elemento della lista.
Esiste una proprietà SelectedItem, che però è di sola lettura, e non viene utilizzata la proprietà Selected della classe ListItem, perché sarebbe prima necessario deselezionare tutti gli altri elementi.
Un metodo più veloce, se non interessa assegnare un valore ad ogni elemento, è l'overload del metodo Add che prende in ingresso una stringa:

```
DropDownList1.Items.Add("Elemento");
```

La terza modalità prevede l'uso del data binding, assegnando una sorgente dati alla proprietà DataSource.
Ecco come collegare la DropDownList ad un array di stringhe:

```
string[] province =  new string[]{"Roma","Torino","Trieste"};

DropDownList1.DataSource = province;
DropDownList1.DataBind();
```

Dopo aver impostato la proprietà DataSource è necessario invocare il metodo DataBind.

Il controllo ListBox

Il controllo ListBox svolge una funzione simile a DropDownList, ma consente la selezione di più elementi contemporaneamente, oltre a mostrarne più di uno, senza necessità di fare clic su di esso per visualizzare l'elenco a discesa.
Le procedure per aggiungere gli elementi sono le stesse viste per DropDownList, per consentire però la selezione multipla è possibile impostare la proprietà SelectionMode a Multiple:

```
<asp:ListBox ID="ListBox1" Runat="server"
SelectionMode="Multiple" Rows="3">
<asp:ListItem>Roma</asp:ListItem>
<asp:ListItem>Messina</asp:ListItem>
<asp:ListItem>Torino</asp:ListItem>
</asp:ListBox>
```

Per selezionare più elementi basta impostare la proprietà Selected degli elementi interessati.
Viceversa per ottenere quali elementi della ListBox sono selezionati bisogna utilizzare il metodo GetSelectedIndices, oppure ciclare su tutti gli elementi e verificare la proprietà Selected.
Il seguente esempio usa il primo metodo e stampa il testo degli elementi selezionati:

```
int[] indices= ListBox1.GetSelectedIndices();
Response.Write("Hai selezionato gli elementi<br>");
foreach (int i in indices)
{
  Response.Write(ListBox1.Items[i].Text+"<br>");
}
```

La proprietà Rows invece serve a specificare il numero di elementi visualizzati nella ListBox, mentre per i rimanenti bisognerà agire sulla barra di scorrimento

Il controllo CheckBox

Per consentire al visitatore della pagina web di effettuare delle selezioni da un insieme di elementi, oppure di impostare dei valori del tipo vero/false oppure accetto/non accetto, è di grande utilità il controllo CheckBox.
Esso consente infatti di inserire la classica casella di controllo singola, su cui si può mettere o togliere un segno di spunta.

La seguente pagina aspx, contiene una CheckBox per permettere all'utente di leggere le condizioni di una licenza e magari continuare ad utilizzare il sito:

```
<asp:TextBox ID="TextBox1" runat="server" Height="70px"
Width="277px">Testo della licenza</asp:TextBox>
<br />
<asp:CheckBox ID="chkAccetto" runat="server"
Text="Accetto la licenza"
/>
<br />
<asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Text="Continua" />
```

Al clic sul pulsante Button1 è possibile verificare lo stato della casella di controllo, leggendo la proprietà Checked, per esempio:

```
if (!chkAccetto.Checked)
{
    Response.Write("È necessario accettare la licenza");
}
```

Un'altra possibilità è quella di reagire all'evento e quindi abilitare il pulsante solo quando la CheckBox è selezionata:

```
protected void chkAccetto_CheckedChanged
(object sender, EventArgs e)
{
    Button1.Enabled = chkAccetto.Checked;
}
```

Per quest'ultimo caso è però necessario che il cambiamento di stato del controllo CheckBox provochi automaticamente un PostBack del-

la pagina, e quindi bisogna impostare a true la proprietà AutoPostBack, o il corrispondente attributo:

```
<asp:CheckBox ID="chkAccetto" runat="server"

Text="Accetto la licenza"
AutoPostBack="true"
 OnCheckedChanged="chkAccetto_CheckedChanged" />
```

L'attributo OnCheckedChanged punta naturalmente al nome del metodo che gestisce l'evento, visto qualche riga più in alto.

Il controllo CheckBoxList

Se fosse necessario permettere all'utente una scelta multipla fra diverse alternative, è naturale utilizzare un controllo CheckBoxList. Il funzionamento è simile ai controlli lista già visti.
Per creare ad esempio una CheckBoxList con diverse scelte, basta inserire dei sottoelementi con il tag <asp:ListItem>.
Nell'esempio seguente viene creata una lista di 4 linguaggi di programmazione:

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server">
        <asp:ListItem>C#</asp:ListItem>
        <asp:ListItem>VB</asp:ListItem>

        <asp:ListItem>Java</asp:ListItem>
        <asp:ListItem>C++</asp:ListItem>
  </asp:CheckBoxList>
```

Se si vogliono conoscere da codice le caselle della lista che l'utente ha selezionato, basta effettuare un ciclo su tutti gli elementi, e verificare il valore della proprietà Selected di ogni elemento.
Ad esempio al clic di un pulsante:

```
protected void Button1_Click(object sender, EventArgs e)
{
    string selezione="Linguaggi scelti:<br>";
    foreach (ListItem item in CheckBoxList1.Items)
    {
        if(item.Selected)
            selezione += item.Text + "<br>";
    }
    labScelte.Text = selezione;
}
```

Il codice precedente visualizzerà in una Label l'elenco dei linguaggi scelti dall'utente, come mostrato nella successiva immagine (figura 3.1).

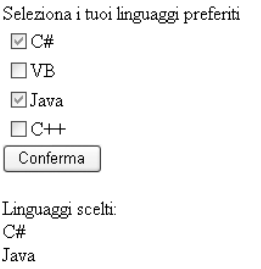

Figura 3.1: Un controllo CheckBoxList

Il controllo RadioButton e RadioButtonList

Il controllo RadioButton è simile ad una CheckBox, in quanto consente ad un utente di selezionare un elemento, esso però non ha senso usato da solo in una pagina.

Infatti il controllo RadioButton permette di effettuare una sola scelta esclusiva, fra diverse disponibili, e dunque è necessario utilizzare un controllo RadioButton per ogni scelta, e raggruppare i controlli che fanno parte di uno stesso gruppo di scelte correlate.

Per creare dei RadioButton è necessario quindi utilizzare almeno le proprietà Text e GroupName.
La prima rappresenta il testo associato al controllo, mentre utilizzando un GroupName comune è possibile raggruppare diversi RadioButton, in maniera che quando uno di essi viene selezionato, tutti gli altri automaticamente vengano deselezionati.

```
<asp:RadioButton ID="RadioButtonCS" runat="server"
AutoPostBack="True"  GroupName="Lang"
OnCheckedChanged="RadioButtonCS_CheckedChanged" Text="C#" />
 <asp:RadioButton ID="RadioButtonVB" runat="server"
AutoPostBack="True" GroupName="Lang"
OnCheckedChanged="RadioButtonVB_CheckedChanged" Text="VB" />
```

L'utilizzo della proprietà GroupName consente di raggruppare RadioButton anche non fisicamente adiacenti o vicini, consentendo di posizionarli come meglio si crede.
Per rispondere alla selezione di un particolare controllo RadioButton, basta gestire l'evento CheckedChanged:

```
protected void RadioButtonCS_CheckedChanged
(object sender, EventArgs e)
{
   labCode.Text = "class NomeClasse<br>{}";
}
protected void RadioButtonVB_CheckedChanged
(object sender, EventArgs e)
{
   labCode.Text = "Class NomeClasse<br>End Class";
}
```

Un'altra possibilità oltre a quella dei RadioButton individuali, è quella di utilizzare il controllo RadioButtonList, che rappresenta appun-

to una lista di controlli RadioButton, ed è in generale scelto quando si ha una sorgente dati da cui creare la collezione.

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server">
<asp:ListItem>C#</asp:ListItem>
<asp:ListItem>VB</asp:ListItem>
<asp:ListItem>C++</asp:ListItem>
</asp:RadioButtonList>
```

Così come per la CheckBoxList è possibile impostare e modificare l'aspetto della lista di RadioButton, mediante le proprietà RepeatLayout, RepeatDirection e RepeatColumns.
Per ricavare invece l'eventuale elemento selezionato basta utilizzare, per esempio in corrispondenza dell'evento SelectedIndexChanged, la proprietà SelectedItem, oppure SelectedIndex se interessa solo l'indice:

```
protected void RadioButtonList1_SelectedIndexChanged
(object sender, EventArgs e)
{
  Response.Write("Selezionato " + RadioButtonList1.SelectedItem.Text);
}
```


Il controllo Image

Il controllo Image consente di visualizzare immagini su una pagina web, e gestirle da codice.
L'immagine da visualizzare è impostata, sia a design time che a runtime tramite la proprietà ImageUrl.
Un controllo Image può essere inserito in questa maniera nella pagina:

```
<asp:Image ID="Image1" runat="server" Height="100px"
Width="100px" ImageUrl="~/Images/immagine.gif" />
```

Come nell'esempio è possibile impostare la larghezza e l'altezza dell'immagine, ridimensionandola prima che essa venga visualizzata nel browser.
Se invece si vogliono mantenere le dimensioni originali, basta non impostare le proprietà Height e Width.
Se l'immagine non venisse visualizzata, ad esempio perché la proprietà ImageUrl è impostata in modo errato, o perché il file immagine non è più disponibile, è possibile mostrare del testo alternativo tramite la proprietà AlternateText.
La proprietà Tooltip invece permette di impostare il tooltip da visualizzare quando il puntatore del mouse si sposta sull'immagine.
La proprietà ImageAlign, consente invece di impostare la modalità di allineamento dell'immagine rispetto ad altri elementi della pagina, ad esempio testo o altri controlli.
I possibili valori sono quelli dell'enumerazione omonima ImageAlign. Il valore di default è NotSet.

Il controllo Panel

Per organizzare dei controlli su una pagina web, specialmente se questi ultimi sono generati dinamicamente, il controllo Panel calza a pennello, permettendo di visualizzare e nascondere in un solo colpo interi gruppi di controlli.
È possibile impostare un'immagine di sfondo utilizzando la proprietà BackImageUrl.

```
<asp:Panel ID="Panel1" runat="server"
BackImageUrl="~/Capitolo4/Tramonto.jpg" Height="300px"
Width="500px">
</asp:Panel>
```

L'allineamento orizzontale degli elementi contenuti nel Panel è determinato dalla proprietà HorizontalAlignment, mentre tramite la proprietà Direction è possibile dire la direzione con cui i controlli con

del testo vengono visualizzati sul pannello, cioè se da sinistra verso destra, con il valore LeftToRight o al contrario se il valore utilizzato è RightToLeft.
In questo esempio vengono creati ed aggiunti dieci pulsanti al pannello Panel1:

```
Panel1.HorizontalAlign = HorizontalAlign.Right;
Panel1.Direction = ContentDirection.RightToLeft;
Button bt;
for (int i = 1; i <= 10; i++)
{
bt = new Button();
bt.ID = "Button" + i;
    bt.Text = "Button " + i;
    Panel1.Controls.Add(bt);
}
```

La proprietà Wrap consente di far continuare la disposizione dei controlli sulla linea successiva se una linea fosse più lunga della larghezza del pannello, per default essa è true, mentre nel caso in cui la si imposti a false, il pannello stesso verrebbe allargato in maniera da riuscire a contenere tutti i controlli aggiunti sulla stessa linea, a meno di non utilizzare la proprietà ScrollBars.
Quest'ultima infatti consente di visualizzare delle barre di scorrimento, sia fisse, con i valori Vertical, Horizontal o Both, oppure in caso di necessità se ScrollBars è impostata al valore ScrollBars.Auto.

Il controllo Table

Uno dei controlli o degli elementi html più comuni in una pagina web è la tabella. ASP.NET consente di creare una tabella in tre diverse maniere.
La prima è il classico tag <table> di HTML, un'altra è l'utilizzo del già

visto controllo HtmlTable, ed infine è possibile utilizzare il controllo server denominato Table.
Con il controllo Table si ha la possibilità di lavorare su diversi tipi di righe (normali, di header e di footer) e manipolare programmaticamente righe e celle.
Il codice seguente crea una tabella con due righe, di cui una di header in grassetto, e con due colonne:

```
<asp:Table ID="Table1" runat="server">
    <asp:TableHeaderRow runat="server" Font-Bold=true>
        <asp:TableCell runat="server">Nome</asp:TableCell>
        <asp:TableCell runat="server">Cognome</asp:TableCell>
    </asp:TableHeaderRow>
    <asp:TableRow runat="server">
        <asp:TableCell runat="server">Giulio</asp:TableCell>
        <asp:TableCell runat="server">Cesare</asp:TableCell>
    </asp:TableRow>
</asp:Table>
```

È possibile aggiungere una riga con due celle, da codice, creando degli oggetti TableCell, aggiungendoli poi alla collezione Cells di un oggetto TableRow, ed infine aggiungere quest'ultimo alla collezione Rows della Table.

```
TableCell cellNome = new TableCell();
cellNome.Text = txtNome.Text;
TableCell cellCognome = new TableCell();
cellCognome.Text = txtCognome.Text;
TableRow row = new TableRow();
row.Cells.AddRange(new TableCell[] { cellNome, cellCognome });
Table1.Rows.Add(row);
```

Da tenere presente che una qualsiasi aggiunta programmatica non persiste dopo un PostBack, perché le celle e le righe sono oggetti a

se stanti, e non proprietà della classe Table, dunque è necessario ricostruirli dopo ogni PostBack.

Il controllo Calendar

Un utile controllo per permettere la selezione di una data in una pagina web è il controllo Calendar, configurabile nel suo aspetto in una grande varietà di modi.
Un calendario standard è visualizzabile su una web form in maniera molto rapida, semplicemente aggiungendo il tag seguente:

```
<asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
```

Che produrrà un oggetto dall'aspetto mostrato nell'immagine seguente (figura 3.2).

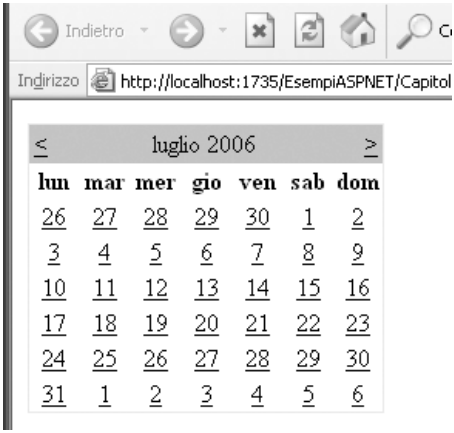

Figura 3.2: Il controllo Calendar in una pagina web

Il controllo Calendar permette di eseguire diverse operazioni basate sulle date. Naturalmente la maniera più immediata di utilizzarlo è quello di mostrare una specifica data.
In questo caso basta impostare la proprietà SelectedDate con un valore DateTime, e se non interessa dare la possibilità di selezionarne

una nuova si può bloccarne l'interazione impostando a None la proprietà SelectionMode.
La proprietà SelectionMode consente infatti di determinare con che modalità sarà possibile selezionare una data.
Se il suo valore è diverso da None, e quindi uno fra Day, DayWeek, DayWeekMonth, sarà possibile visualizzare e selezionare con un clic periodi rispettivamente di un giorno, una settimana o l'intero mese.
I possibili periodi selezionabili saranno mostrati come dei LinkButton, che al clic del mouse genereranno un evento SelectionChanged:

```
protected void Calendar1_SelectionChanged
 (object sender, EventArgs e)
{
    if (Calendar1.SelectedDates.Count == 1)
        Response.Write("Selezionato il giorno " +
 Calendar1.SelectedDate.ToShortDateString());
    else
    {
        Response.Write("Selezionato il periodo dal " +
              Calendar1.SelectedDates[0].ToShortDateString());
        int count=Calendar1.SelectedDates.Count;
        Response.Write(" al " + Calendar1.SelectedDates[count -
 1].ToShortDateString());
    }
}
```

Il codice precedente stampa sulla pagina il periodo selezionato dall'utente nel controllo Calendar.
Spostandosi inoltre da un mese all'altro verrà generato l'evento VisibileMonthChanged, all'interno del cui gestore è possibile ricavare il mese corrente e quello precedente:

```
protected void Calendar1_VisibleMonthChanged(object sender,
```

```
MonthChangedEventArgs e)
{
 Response.Write("Cambiato mese da " +
 e.PreviousDate.Month + " a " + e.NewDate.Month);
}
```

Il terzo ed ultimo evento specifico del controllo Calendar viene generato ogni volta che la cella di un giorno del controllo viene disegnata, permettendo dunque di personalizzare l'aspetto di ognuna di esse.
In questa maniera si può ad esempio visualizzare dei giorni con un aspetto particolare.
L'esempio seguente mostra come impostare un appuntamento nel giorno selezionato, facendo clic su un pulsante, e dunque visualizzare il giorno con uno sfondo di colore rosso:

```
protected void Button1_Click(object sender, EventArgs e)

{
    Appuntamento = Calendar1.SelectedDate;
}

public DateTime Appuntamento
{
    get
    {
        if (ViewState["appuntamento"] != null)
            return (DateTime)ViewState["appuntamento"];
        else return new DateTime();

    }
    set { ViewState["appuntamento"] = value; }
}
```

Al clic sul pulsante, la data selezionata viene memorizzata nel ViewState, in maniera che il metodo DayRender sappia quale giorno evidenziare.
A questo punto si può formattare la proprietà Cell del parametro DayRenderEventArgs, aggiungendo anche un controllo Literal con la stringa "Appuntamento":

```
protected void Calendar1_DayRender

 (object sender, DayRenderEventArgs e)
{
    if (e.Day.Date == Appuntamento)
    {
        e.Cell.BackColor = Color.Red;

        Literal lit = new Literal();

        lit.Text = "Appuntamento";

        e.Cell.Controls.Add(lit);
    }
}
```

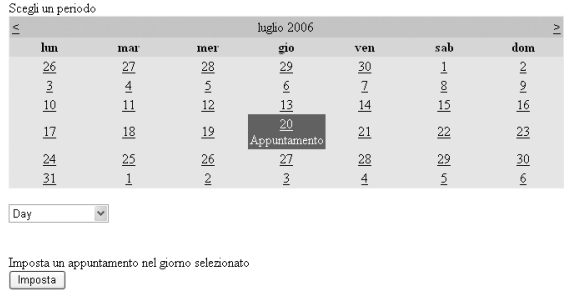

Figura 3.3: Personalizzazione di Calendar

Il controllo AdRotator

Il controllo AdRotator permette di creare il classico banner con delle immagini che variano ad ogni postback della pagina.
L'AdRotator visualizza un'immagine, ridimensionata automaticamente alla stessa dimensione del controllo, con eventualmente un'url associato.
Il comportamento del controllo viene pilotato da un file XML, con uno schema specifico.
Il seguente potrebbe essere un file XML tipico da utilizzare con un AdRotator:

```
<Advertisements>
<Ad>
  <ImageUrl>sponsor1.gif</ImageUrl>
  <NavigateUrl>http://www.ioprogrammo.it</NavigateUrl>
  <AlternateText>Il sito di IOprogrammo</AlternateText>
  <Impressions>50</Impressions>
</Ad>
<Ad>
  <ImageUrl>sponsor2.gif</ImageUrl>
  <NavigateUrl>http://www.dotnetarchitects.it</NavigateUrl>
  <AlternateText>Il sito di .NET Architects</AlternateText>
  <Impressions>50</Impressions>
</Ad>
</Advertisements>
```

L'elemento root Advertisements contiene tanti elementi figli Ad quante sono le immagini da visualizzare.
Ogni Ad deve avere poi almeno un nodo ImageUrl che indica il percorso dell'immagine da visualizzare nel controllo.
Gli altri elementi sono opzionali, ed elementi personalizzati possono essere creati per essere poi elaborati via codice.
L'elemento NavigateUrl imposta il collegamento ipertestuale del-

l'immagine, AlternateText consente di impostare un testo alternativo nel caso in cui l'immagine non venga trovata, ed Impressions è un valore numerico che indica quanto spesso l'immagine associata debba essere utilizzata in relazione alle altre presenti nel file XML.
In questo caso le due immagini hanno entrambe un valore Impressions di 50, quindi verranno visualizzate in media e alternativamente lo stesso numero di volte.
Tramite la proprietà AdvertisementFile può essere impostato il percorso del file XML da utilizzare, ad esempio:

```
<asp:AdRotator ID="AdRotator1" runat="server"
AdvertisementFile="~/Advertisements/advertisement.xml"
Height="100px" Width="200px" />
```

Il controllo Xml

Il controllo Xml consente di visualizzare il contenuto di un file XML in una web form. Il controllo permette inoltre di applicare un foglio di stile al puro XML. Il modo più semplice di caricare un file XML ed impostare il foglio di stile xsl, è quello dichiarativo, utilizzando le proprietà DocumentSource e TransformSource :

```
<asp:Xml ID="Xml1" runat="server" DocumentSource="~/advertise
ment.xml" TransformSource="~/adtransform.xsl">
</asp:Xml>
```

Dato il precedente advertisement.xml, possiamo definire il file del foglio di stile adtransform.xsl in questo modo:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/Advertisements">
  <xsl:apply-templates select="Ad" />
</xsl:template>
```

```
<xsl:template match="Ad">
  <table width="100%" border="1">
    <tr>
      <td>
        <b>
          <xsl:value-of select="ImageUrl" />
        </b>
        <td>
          <xsl:value-of select="NavigateUrl" />
        </td>
      </td>
    </tr>
  </table>
 </xsl:template>
</xsl:stylesheet>
```

Che visualizzerà in una tabella gli elementi ImageUrl e NavigateUrl ricavati dall'XML

Il controllo PlaceHolder

Come indica il nome, un PlaceHolder serve da segnaposto, cioè da contenitore di controlli che è possibile inserire dinamicamente al suo interno, e quindi visualizzare su una pagina o altro controllo custom. Il PlaceHolder è uno dei pochi controlli non derivati dalla classe WebControl, ma direttamente da Control. Si è già utilizzato il PlaceHolder in qualche esempio.

```
<asp:PlaceHolder ID="PlaceHolderLinks"
runat="server"></asp:PlaceHolder>
```

Per aggiungere un controllo, basta inserirlo nella collezione Controls dell'oggetto PlaceHolder:

```
Button bt=new Button();
PlaceHolderLinks.Controls.Add(bt);
```

Il controllo PlaceHolder non produce alcun elemento HTML, ed inoltre è da tener presente il fatto che ad ogni PostBack il suo contenuto viene svuotato, a meno di non usare altri provvedimenti per conservare lo stato della pagina, ad esempio il View State o campi hidden, dai quali quindi ricreare anche il contenuto del PlaceHolder.

3.3.3 I nuovi controlli di ASP.NET 2.0

I controlli visti finora erano già presenti in ASP.NET 1.1, anche se in qualche caso con delle funzionalità differenti o mancanti.
La versione 2.0 di ASP.NET aggiunge una serie di nuovi controlli, che aumentano le possibilità a disposizione degli sviluppatori.

Il controllo BulletedList

Il controllo BulletedList genera un elenco puntato di elementi.
Ogni singolo elemento è rappresentato da un oggetto ListItem, mentre lo stile utilizzato per ogni punto dell'elenco può essere definito mediante la proprietà BulletStyle, con uno dei valori dell'enumerazione omonima. Per esempio può essere utilizzata una immagine personalizzata, utilizzando prima il valore CustomImage, ed impostando poi il percorso del file immagine nella proprietà BulletImageUrl:

```
<asp:BulletedList ID="BulletedList1" runat="server"
BulletStyle="CustomImage"
BulletImageUrl="../Images/arrow.gif">
        <asp:ListItem>elemento1</asp:ListItem>
        <asp:ListItem>elemento2</asp:ListItem>
</asp:BulletedList>
```

Una BulletedList può essere legata ad una sorgente dati, utilizzando la proprietà DataSourceID.

Per esempio, con un database Access, contenente una tabella Clienti, si può creare un elenco dei loro cognomi:

```
<asp:AccessDataSource ID="AccessDataSource1" runat="server"
DataFile="~/App_Data/clienti.mdb"
SelectCommand="SELECT Nome, Cognome,IndirizzoPostaElettronica
FROM [Clienti]">
</asp:AccessDataSource>

<asp:BulletedList ID="BulletedList1" runat="server"
BulletStyle="CustomImage"
BulletImageUrl="../Images/arrow.gif"
  DataSourceID="AccessDataSource1" DataTextField='Cognome'>
</asp:BulletedList>
```

Ogni elemento della lista, può anche essere dotato di un comportamento attivo, trasformandolo in un collegamento ipertestuale o in un LinkButton.
La proprietà DisplayMode, infatti può essere impostata, oltre che al valore di default Text, ai valori HyperLink e LinkButton.

```
 <asp:BulletedList ID="BulletedList2" runat="server"
BulletStyle="Square"
  DataSourceID="AccessDataSource1" DataTextField='Cognome'
DataValueField="IndirizzoPostaElettronica" DisplayMode="LinkButton"
OnClick="BulletedList2_Click">

</asp:BulletedList>
```

Utilizzando dei LinkButton come nell'esempio, sarà possibile determinare il clic su uno degli elementi del controllo e leggerne il valore all'interno dell'evento Click, per esempio usando il parametro BulletedListEventArgs:

```
protected void BulletedList2_Click
(object sender, BulletedListEventArgs e)
{
 string email = BulletedList2.Items[e.Index].Value;
 //invio email;
}
```

Il controllo FileUpload

Il controllo FileUpload consente all'utente di una pagina di selezionare un file dal proprio computer e caricarlo sul server.
Esso mostra una TextBox per digitare il percorso del file, ed un pulsante di sfoglia per ricercarlo sull'hard disk.
Nel seguente esempio si hanno un FileUpload, un pulsante per avviare il caricamento di un file gif, ed un controllo Image, per mostrare il file caricato.

```
<asp:FileUpload ID="FileUpload1" runat="server" />
<asp:Button ID="btUpload" runat="server" OnClick="btUpload_Click"
Text="Carica" /><br />
<asp:Image ID="Image1" runat="server" />
```

Per gestire il file caricato, al clic del pulsante si verifica che nel controllo FileUpload sia stato impostato un file, che sarà accessibile tramite la proprietà PostedFile.
A questo punto si verifica il tipo MIME mediante la proprietà ContentType e si può accedere al file per salvarlo, tramite il metodo SaveAs.
Con il file ormai sul server, possiamo utilizzare il suo percorso per impostare la ImageUrl di un controllo Image:

```
protected void btUpload_Click(object sender, EventArgs e)
{
 if (IsPostBack)
 {
```

```
        string path = Server.MapPath("~/Images/");
        if (FileUpload1.HasFile)
        {
            if(FileUpload1.PostedFile.ContentType=="image/gif")
            {
                try
                {
        FileUpload1.PostedFile.SaveAs(path + FileUpload1.FileName);
        Image1.ImageUrl = "~/Images/" + FileUpload1.FileName;
                }
                catch (Exception ex)
                {
                    Response.Write("impossibile caricare il file.");
                }
            }
            else Response.Write("formato consentito: .gif");
        }
    }
}
```

Il contenuto del file caricato, oltre che salvato direttamente sul File System, può anche essere letto come array di byte. In tal modo si potrà per esempio salvarlo in una tabella di database:

```
System.IO.Stream myStream = FileUpload1.FileContent;
byte[] bytes = myStream.ToArray();
```

Il controllo HiddenField

HiddenField è un controllo web anomalo, perché esso non visualizzerà nulla nel browser.

Lo scopo è analogo ai campi input nascosti di html, cioè quello di creare un campo nascosto all'interno del quale conservare dei valori ma non visualizzarli.

In particolare esso conserva l'eventuale valore che contiene anche dopo un postback, ed è quindi uno dei modi di persistere lo stato di una pagina, argomento su cui si tornerà nel capitolo dedicato alla gestione dello stato.
Creare un HiddenField è semplice come aggiungere un tag con relativo id:

```
hiddenfield id="HiddenField1" value="" runat="server" />
```

Per impostare e leggere il contenuto di un HiddenField si utilizza la sua proprietà Value.
Mentre se si vuole sapere quando avviene l'eventuale cambiamento del suo valore, basta gestire l'evento ValueChanged:

```
<asp:hiddenfield id="ValueHiddenField" value="" runat="server"
        onvaluechanged="ValueHiddenField_ValueChanged"/>
…
protected void HiddenField1_ValueChanged(object sender, EventArgs e)
{
   Response.Write("Il campo nascosto contiene ora " +
ValueHiddenField.Value);
}
```

Il controllo ImageMap

Il nuovo controllo ImageMap consente di definire su una normale immagine, delle aree, di varie forme, cliccabili dall'utente, il tutto utilizzando delle coordinate bidimensionali.
Le aree attive sono dette HotSpot, e sono configurabili mediante degli elementi inseriti all'interno del tag <asp:ImageMap>.
L'esempio seguente utilizza un'immagine di un campo da calcio, definendo delle zone HotSpot per le aree di rigore e per il centrocampo.
Dunque in questo caso bisogna utilizzare rispettivamente due Rec-

tangleHotSpot ed un CircleHotSpot, impostando per il primo tipo i valori Left, Top, Right, Bottom, mentre per creare un cerchio sono necessarie le coordinate X,Y del centro ed il valore Radium del raggio. L'immagine utilizzata è quella in figura, che mostra anche il modo di definire gli HotSpot (figura 3.4).

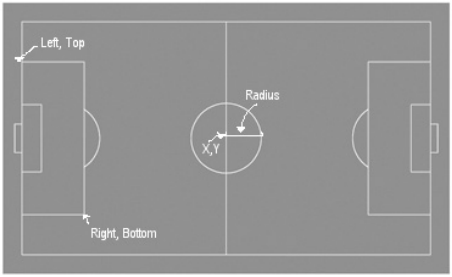

Figura 3.4: L'immagine utilizzata per creare una mappa

In maniera dichiarativa, l'ImageMap sarà creato in questa maniera:

```
<asp:ImageMap ID="ImageMap1" runat="server"

ImageUrl="../Images/campo_calcio.gif">

<asp:RectangleHotSpot Left=22 Top=57 Right=74 Bottom=185

AlternateText="Area di rigore" />

        <asp:CircleHotSpot X=190 Y=120 Radius=30
AlternateText="Centrocampo" />
</asp:ImageMap>
```

Nell'esempio viene anche utilizzato l'attributo AlternateText, che apparirà come Tooltip quando il mouse si posizionerà su una delle aree definite, come in figura 3.5

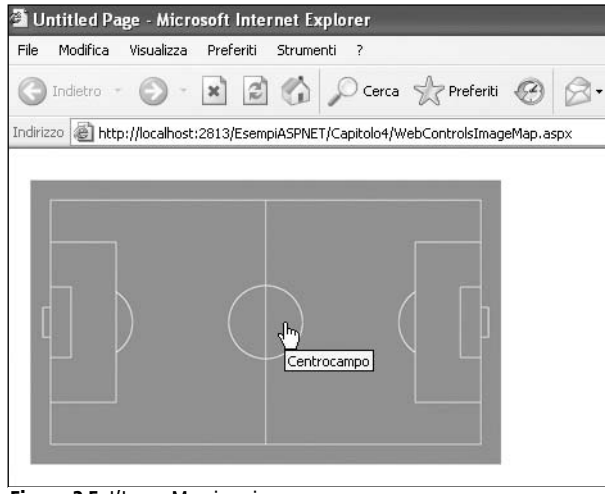

Figura 3.5: L'ImageMap in azione

Nel caso di aree non rettangolari o circolari è possibile definire dei poligoni, creando dei PolygonHotSpot, oppure derivare la propria classe personalizzata da HotSpot.
Un' altra proprietà fondamentale, per definire il comportamento al clic del mouse, è la HotSpotMode. Essa è presente sia come proprietà del controllo ImageMap, in maniera da impostare lo stesso HotSpotMode per tutte le aree HotSpot, oppure è possibile definire una HotSpotMode differente per ogni HotSpot della mappa.
Se si vuole aprire un URL al click su una particolare area, si può utilizzare il valore Navigate ed impostare l'indirizzo web da aprire in NavigateUrl:

```
<asp:RectangleHotSpot Left=306 Top=57 Right=358 Bottom=185

    AlternateText="Area di rigore destra" HotSpotMode="navigate"
NavigateUrl="area.aspx "
/>
```

La seconda possibilità è generare un PostBack, dando a HotSpotMode il valore PostBack ed impostare poi un valore in PostBackValue per distinguere l'area HotSpot selezionata, che sarà leggibile dopo il PostBack gestendo l'evento Click:

```
protected void ImageMap1_Click(object sender, ImageMapEventArgs e)
{

  if (e.PostBackValue == "centrocampo")
        Response.Write("Hai cliccato sul centrocampo");
}
```

Infine è anche possibile creare un HotSpot non attivo, con il valore HotSpotMode impostato ad Inactive. In tal modo l'area non risponderà al clic, ma verrà visualizzata in ogni caso l'AlternateText, ed è dunque utilizzabile per aggiungere delle informazioni testuali ad una immagine.

I controlli MultiView e View

MultiView e View sono due controlli che lavorano in stretta collaborazione, il primo come contenitore ed il secondo come contenuto, per creare delle pagine con sezioni visibili in maniera alternativa in una stessa pagina web.
Lo stesso effetto si potrebbe ottenere in altri modi, come per esempio agendo sulla proprietà Visible di diversi controlli Panel.
I controlli View, sono più facilmente gestibili e configurabili, soprattutto in un ambiente come Visual Studio .NET.
Infatti, anche in design mode, è possibile trascinare un controllo MultiView sulla pagina, e poi creare le varie View possibili, trascinando altrettanti controlli sul controllo contenitore MultiView.
Da codice html, comunque, non è per niente complesso, basta aggiungere un tag asp:MultiView e creare le sue View, aggiungendo come elementi figli i relativi tag asp:View:

```
<asp:MultiView ID="MultiView1" runat="server">
        <asp:View ID="View1" runat="server">
            Vista con un pulsante<br />
        <asp:Button ID="Button1" runat="server" Text="Button" /
></asp:View>
        <asp:View ID="View2" runat="server">
            <asp:Image ID="Image1" runat="server"
ImageUrl="~/Images/redgreenyellow.gif" />
         </asp:View>
        <asp:View ID="View3" runat="server">
            scegli
            <br />
    <asp:RadioButtonList ID="RadioButtonList1" runat="server">
                <asp:ListItem>scelta 1</asp:ListItem>
                <asp:ListItem>scelta 2</asp:ListItem>
            </asp:RadioButtonList>
        </asp:View>
</asp:MultiView>
```

Ogni View può contenere tutti gli elementi HTML, il testo, o i controlli web che si vogliono. Una sola vista è attiva ad ogni dato istante. Per determinare o impostare quale delle View presenti in un MultiView è quella attiva, basta utilizzare la proprietà ActiveViewIndex.
Per esempio se una DropDownList contenesse degli elementi ognuno con un valore corrispondente ad una data View, si potrebbe cambiare la vista attiva semplicemente così:

```
protected void DropDownList1_SelectedIndexChanged
(object sender, EventArgs e)
{
    MultiView1.ActiveViewIndex =
Convert.ToInt32(DropDownList1.SelectedValue);
}
```

Un'altra possibilità, forse ancora più semplice, è quella di utilizzare il metodo SetActiveView, passando come argomento proprio l'oggetto View da rendere visibile:

```
MultiView1.SetActiveView(View1);
```

Il controllo Wizard

I controlli MultiView e View apena visti potrebbero essere utilizzati per creare una procedura guidata sul web, cioè un classico Wizard, attivando una View dopo l'altra.
Fortunatamente esiste già un controllo Wizard che permette di realizzare proprio tali procedure, e, cosa non sorprendente, internamente esso utilizza proprio le classi View e MultiView.
Il modo più semplice di creare un Wizard è quello di aggiungere un tag asp:Wizard alla pagina contenente tanti elementi WizardStep quanti sono i passi che costituiscono la procedura.
Il seguente esempio crea un Wizard con tre passi, il cui ordine è definito unicamente dall'ordine con cui essi sono aggiunti al controllo Wizard:

```
<asp:Wizard ID="Wizard1" runat="server">
    <WizardSteps>
        <asp:WizardStep runat="server" Title="Step 1">
        Primo passo
        </asp:WizardStep>
        <asp:WizardStep runat="server" Title="Step 2">
        Secondo passo
        </asp:WizardStep>

        <asp:WizardStep runat="server" Title="Step 3">
        Terzo ed ultimo passo

        </asp:WizardStep>
```

```
        </WizardSteps>
</asp:Wizard>
```

La seguente immagine (figura 3.6) mostra i tre passi del Wizard precedente. Il primo passo contiene il solo pulsante Next, il secondo i pulsanti Next e Previous e l'ultimo i pulsanti Finish e Previous.

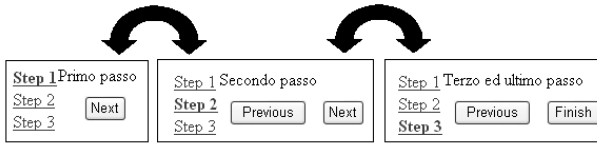

Figura 3.6: I passi di un Wizard

Ogni passo dell'esempio precedente contiene solo del testo, ma può naturalmente contenere una qualsiasi quantità e tipologia di controlli, ad esempio creando un modulo di richiesta informazioni con diverse caselle di testo, caselle combinate e tutto quanto possa servire.
Inoltre il controllo Wizard mostra per default sulla sinistra una barra di navigazione, che consente di spostarsi su un dato passo facendo clic sul titolo corrispondente.
Tale barra può essere nascosta impostando a false la proprietà DisplaySideBar.

Configurare i pulsanti

Tutti i pulsanti possono essere configurati mediante proprietà, sia da designer che da codice.
Per esempio è possibile cambiare il testo del pulsante di Start del primo passo modificando la proprietà StartNextButtonText, oppure utilizzare al posto del pulsante standard un'immagine o un collegamento, impostando la proprietà StartNextButtonType rispettivamente ai valori Image o Link.
Se si utilizza un immagine, il percorso di questa deve essere im-

postato tramite la proprietà StartNextButtonImageUrl.
Lo stesso meccanismo vale per tutti quanti i pulsanti del Wizard. In Visual Studio 2005 è possibile gestire tutti gli step in maniera completamente visuale, ed i tipi di step sono configurati automaticamente in base alla loro posizione: ciò deriva dal fatto che la proprietà StepType dell'elemento WizardStep è configurata al valore Auto.
I valori possibili, oltre ad Auto, sono quelli dell'enumerazione WizardStepType: Start, Step, Finish, Complete.
Quindi il valore Start creerà per il passo interessato solo il pulsante Next, il valore Step i due pulsanti Next e Previous, il valore Finish creerà i pulsanti Previous e Finish, ed infine Complete serve per configurare il passo finale di un Wizard, che mostra solo informazioni di riepilogo e dunque non contiene più alcun pulsante per navigare all'indietro.
Se si vuol evitare il ritorno ad un passo precedente, per esempio per impedire che l'utente da "Step 2" possa tornare a "Step 1", si può anche impostare a false la proprietà AllowReturn del passo con titolo "Step 1". In alcuni casi è necessario fornire anche un pulsante per annullare ed uscire completamente dalla procedura guidata. Dunque sarebbe utile un pulsante Cancel.
Per ottenerlo basta impostare a true la proprietà DisplayCancelButton, e poi modificare eventualmente il testo o l'immagine del pulsante che sarà visualizzato su ogni pagina del Wizard come visto per gli altri pulsanti.

Lavorare con gli eventi

Il Wizard deve consentire di salvare le informazioni inserite o selezionate dall'utente ad un certo passo, in maniera da poterle utilizzare ad un passo successivo, o per effettuare al completamento della procedura stessa determinate operazioni.
Il controllo Wizard fornisce diversi eventi per stabilire la navigazione dell'utente attraverso i vari passi e dunque ricavare ed

impostare i valori dei controlli che essi contengono. Inoltre la proprietà ActiveStepIndex consente in ogni momento di sapere quale passo è attualmente visualizzato nel controllo.
La seguente tabella mostra i diversi eventi disponibili e specifici del controllo Wizard

Evento	Descrizione
ActiveStepChanged	Si verifica ad ogni spostamento da un passo all'altro del Wizard, in ogni direzione e per ogni possibile origine e destinazione.
CancelButtonClick	Si verifica al clic su un pulsante Cancel.
FinishButtonClick	Si verifica al clic sul pulsante di Finish.
NextButtonClick	Si verifica al clic su un pulsante Next.
PreviousButtonClick	Si verifica al clic su un pulsante Previous.
SideBarButtonClick	Si verifica quando l'utente fa clic su uno dei collegamenti della barra di navigazione.

Supponendo di aver aggiunto dei controlli TextBox nel primo Step, per l'inserimento di nome, cognome e email di un utente, ed una CheckBoxList per selezionare degli interessi da un elenco, come mostrato di seguito:

```
<WizardSteps>
        <asp:WizardStep runat="server" Title="Step 1">
            inserisci le tue informazioni<br />
            nome
  <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <br />
            cognome
  <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
            <br />
```

```
            email
    <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>
        </asp:WizardStep>
        <asp:WizardStep runat="server" Title="Step 2">
            <asp:Label ID="LabelStep2" runat="server"
Text="Label"></asp:Label>
            <br />
            Seleziona i tuoi interessi<br />
        <asp:CheckBoxList ID="CheckBoxList1" runat="server">
            <asp:ListItem>Computer</asp:ListItem>
            <asp:ListItem>Sport</asp:ListItem>
            <asp:ListItem>Musica</asp:ListItem>
            <asp:ListItem>Libri</asp:ListItem>
            <asp:ListItem>Cinema</asp:ListItem>
        </asp:CheckBoxList>
        </asp:WizardStep>
        <asp:WizardStep runat="server" Title="Step 3"
StepType="Finish">
            Riepilogo delle informazioni<br />
            <br />
            <asp:Label ID="LabelFinish" runat="server"
Text="Label"></asp:Label>
            <br />
            premi Finish per salvare</asp:WizardStep>
        <asp:WizardStep runat="server" StepType="Complete"
Title="Completato">
            I tuoi dati e i tuoi interessi sono stati
memorizzati</asp:WizardStep>
</WizardSteps>
```

Si può gestire l'evento ActiveStepChanged per ricavare le informazioni inserite ad ogni passo e mostrarne un riepilogo al passo finale, e dunque prima di un salvataggio delle stesse informazioni:

```
protected void Wizard1_ActiveStepChanged
(object sender, EventArgs e)
{ if (Wizard1.ActiveStepIndex == 1)
    {
        LabelStep2.Text = "Ciao " + TextBox1.Text + ",";
    }
    else if (Wizard1.ActiveStepIndex == 2)
    {
        StringBuilder sb=new StringBuilder();
        sb.AppendFormat("Nome: {0}<br>",TextBox1.Text);
        sb.AppendFormat("Cognome: {0}<br>",TextBox2.Text);
        sb.AppendFormat("Email: {0}<br>",TextBox3.Text);
        sb.Append("Interessi: <br>");
        foreach(ListItem item in CheckBoxList1.Items)
        {
            if(item.Selected)
            {
                sb.AppendFormat("- {0} <br>",item.Text);
            }
        }
        LabelFinish.Text=sb.ToString();
    }
}
```

Si noti che i controlli presenti in una qualunque schermata della procedura sono accessibili in qualunque istante e quindi ad ogni cambiamento del passo attivo, che viene ricavato nel gestore dell'evento ActiveStepChanged.

Il controllo TreeView

Il controllo TreeView è un altro dei controlli introdotti in ASP.NET 2.0, e serve naturalmente a visualizzare dei dati mediante una struttura ad albero.

Per creare un albero su una pagina ASP.NET basta utilizzare il tag asp:TreeView e definire la struttura di nodi all'interno dell'elemento Nodes. Ogni nodo dell'albero viene rappresentato da un oggetto TreeNode::

```
<asp:TreeView ID="TreeView1" runat="server">
        <Nodes>
            <asp:TreeNode Text="New Node1" Value="New
 Node"></asp:TreeNode>
            <asp:TreeNode Text="New Node2" Value="New Node">
                <asp:TreeNode Text="New Node3" Value="New
 Node"></asp:TreeNode>
            </asp:TreeNode>
        </Nodes>
</asp:TreeView>
```

L'albero così creato non sarà certamente visivamente complesso ed allettante, e magari non si adatterà all'aspetto di un sito esistente. Se si utilizza Visual Studio è possibile configurare l'aspetto di un TreeView in maniera semplice e senza scrivere manualmente nessuna riga di codice.
Facendo clic con il tasto destro del mouse sull'albero, nella pagina di design, e selezionando dal menù la voce Auto Format, è possibile scegliere uno degli stili predefiniti.
Per esempio, lo stile Contacts darà al TreeView uno stile simile a quello che mostra i contatti di MSN Messenger, e che in codice HTML assomiglierà al seguente:

```
<asp:TreeView ID="TreeView1" runat="server" ImageSet="Contacts"
NodeIndent="10">
  <Nodes>
    <asp:TreeNode Text="New Node" Value="New
Node"></asp:TreeNode>
```

```
        <asp:TreeNode Text="New Node" Value="New Node">
            <asp:TreeNode Text="New Node" Value="New
  Node"></asp:TreeNode>
        </asp:TreeNode>
    </Nodes>
    <ParentNodeStyle Font-Bold="True" ForeColor="#5555DD" />
    <HoverNodeStyle Font-Underline="False" />
    <SelectedNodeStyle Font-Underline="True"
                HorizontalPadding="0px" VerticalPadding="0px" />
    <NodeStyle Font-Names="Verdana" Font-Size="8pt"
   ForeColor="Black"
                HorizontalPadding="5px"  NodeSpacing="0px"
  VerticalPadding="0px" />
</asp:TreeView>
```

Come si può notare vengono definiti caratteri, colori, dimensioni di ogni tipologia di nodo, per esempio per un nodo selezionato, per uno radice, per uno normale e così via, in questo caso mediante le proprietà ParentNodeStyle, HoverNodeStyle, SelectedNodeStyle e NodeStyle.

Naturalmente un albero su una pagina web non è utile solo per visualizzare informazioni, ma può servire come menù di navigazione o di comando. In questo caso potrebbe essere gestito l'evento SelectedNodeChanged che si verifica quando l'utente seleziona un nuovo nodo con il mouse:

```
protected void TreeView1_SelectedNodeChanged
(object sender, EventArgs e)
{
  Response.Write("Hai selezionato il nodo
"+TreeView1.SelectedNode.Text);

}
```

Una proprietà interessante del controllo è ShowCheckBoxes che consente di visualizzare delle CheckBox a diversi livelli dei nodi dell'albero. La proprietà può assumere infatti uno dei valori dell'enumerazione TreeNodeTypes: All, None, Leaf, Parent, Root.
Un nodo Leaf è un nodo foglia, che cioè non ha nodi figli. Un nodo Root è una radice, cioè con uno o più figli, ma che non ha alcun nodo al di sopra di esso.
Un nodo Parent invece è un nodo che può avere figli ma anche un padre.
Se dunque si volessero creare delle CheckBox solo a livello dei nodi terminali, o foglie, si utilizzerà il valore Leaf.

```
TreeView1.ShowCheckBoxes=TreeNodeTypes.Leaf;
```

Un'altra possibilità per aggiungere una CheckBox ai singoli nodi, è quella di impostare la proprietà ShowCheckBox di ogni singolo TreeNode, che è un valore booleano: true se si vuole la casella di selezione, false altrimenti.

```
<asp:TreeNode Text="New Node" Value="New Node"
ShowCheckBox="True">
```

I nodi selezionati mediante un segno di spunta potranno poi essere ricavati leggendo la proprietà CheckedNodes:

```
foreach(TreeNode checkedNode in TreeView1.CheckedNodes)
{
…
}
```

La classe TreeView mette a disposizione dello sviluppatore diverse proprietà e metodi per controllarne il comportamento da codice.
Per esempio è possibile espandere tutti i nodi dell'albero oppure al

contrario collassarlo, mediante i metodi ExpandAll o CollapseAll rispettivamente.

3.4 CONTROLLI PERSONALIZZATI

Se nel catalogo di controlli messo a disposizione da ASP.NET 2.0 mancasse una particolare funzionalità, è sempre possibile implementare un proprio controllo personalizzato da zero oppure estendendo quelli predefiniti.

3.4.1 User Control

Uno User Control è il modo più semplice per creare un controllo personalizzato, in quanto si tratta semplicemente dell'incapsulamento di altri controlli in uno solo, in maniera da sfruttarne le caratteristiche congiunte, e poterle utilizzare facilmente e comodamente in qualunque parte di un sito web. Si supponga ad esempio di volere creare un controllo per permettere all'utente di recuperare la propria password dimenticata (esiste già, ma si faccia finta di niente).
Un controllo simile avrà bisogno di una Label, di una TextBox all'interno della quale inserire il proprio indirizzo e-mail, ed un Button per effettuare la richiesta.
I controlli suddetti possono essere raggruppati in un singolo User Control, chiamato magari RetrievePasswordControl.
In Visual Studio creare il controllo è abbastanza immediato, come al solito basta scegliere la voce Add New Item dal menù contestuale della soluzione, e selezionare poi la tipologia Web User Control, quindi non resta che impostare il nome del file, che avrà estensione ascx. Dando un'occhiata al codice HTML generato per il file, si leggerà un contenuto predefinito simile al seguente:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="RetrievePasswordControl.ascx.cs"
Inherits="RetrievePasswordControl" %>
```

Rispetto ad una pagina aspx, si noterà la direttiva Control al posto della Page, e naturalmente nessun tag body o form, dato che essi saranno forniti dalla pagina web stessa che conterrà il nuovo controllo. Anche in questo caso, basta trascinare i controlli che costituiscono il nuovo User Control dalla casella degli strumenti, oppure scrivere i tag HTML per la relativa creazione. Il controllo per il recupero della password potrebbe essere fatto così:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="RetrievePasswordControl.ascx.cs"
Inherits="Controls_RetrievePasswordControl" %>
<asp:Label ID="Label1" runat="server" Text="Inserisci la tua
Email"></asp:Label>
<asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
<asp:Button ID="btRetrieve" runat="server" Text="Invia"
OnClick="btRetrieve_Click" />
```

Si noti la presenza anche del nome del metodo che gestirà il clic sul pulsante:

```
protected void btRetrieve_Click(object sender, EventArgs e)
{…}
```

Una volta implementata la logica di funzionamento del controllo, per utilizzarlo in una pagina dell'applicazione web, basterà trascinare il file sulla pagina di Design, se si lavora con Visual Studio, oppure aggiungere manualmente il codice. Innanzitutto è necessario registrare il controllo mediante la direttiva Register:

```
<%@ Register Src="RetrievePasswordControl.ascx"
 TagName="RetrievePasswordControl"
  TagPrefix="uc1" %>
```

Dopodiché sarà possibile posizionare il controllo in maniera analoga a quelli predefiniti, utilizzando il prefisso indicato nell'attributo TagPrefix ed il suo TagName:

```
<body>

 <form id="form1" runat="server">
    <uc1:RetrievePasswordControl ID="RetrievePasswordControl1"
runat="server" />

 </form>
</body>
```

3.5 I CUSTOM CONTROL

La seconda possibilità per la creazione di controlli personalizzati, è quella di scrivere una propria classe, derivandola da una delle classi progenitrici di tutti i web control ASP.NET: System.Web.UI.Control oppure System.Web.UI.WebControls.WebControl.
Se si vuole realizzare una libreria di controlli custom da riutilizzare in diverse applicazioni web, è il caso di creare un progetto apposito.
In Visual Studio è possibile creare il progetto selezionando il tipo di progetto Web Control Library che si trova nella sottocategoria Windows, sia che si preferisca Visual Basic, sia con Visual C#, come mostrato nella figura 3.7
Confermata la scelta, Visual Studio creerà automaticamente uno scheletro del controllo custom, nel linguaggio scelto:

```
[DefaultProperty("Text")]
[ToolboxData("<{0}:WebCustomControl1
runat=server></{0}:WebCustomControl1>")]
public class WebCustomControl1 : WebControl
{
```

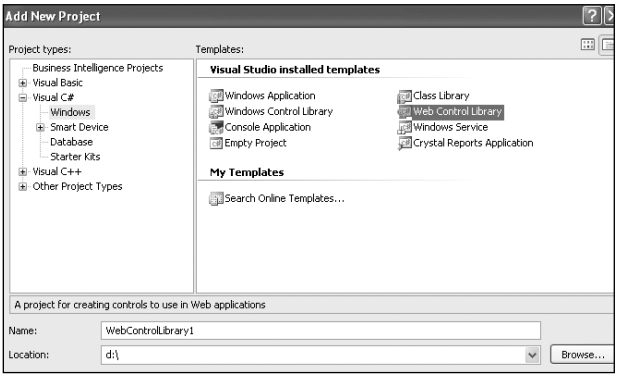

Figura 3.7: Creazione di una libreria di WebControl

```
    [Bindable(true)]

    [Category("Appearance")]
    [DefaultValue("")]

    [Localizable(true)]
    public string Text

    {

        get
        {
            String s = (String)ViewState["Text"];
            return ((s == null) ? String.Empty : s);
        }
        set
        {
            ViewState["Text"] = value;
        }
    }
```

```
protected override void RenderContents(HtmlTextWriter output)
    {
        output.Write(Text);
    }
}
```

Tale classe, derivata da WebControl, definisce un controllo denominato WebCustomControl1, dotato di una proprietà Text, e che visualizza il contenuto di tale proprietà sullo schermo, mediante l'override del metodo RenderContents. In pratica non fa niente di più di una Label.
L'utilizzo dei vari attributi, permette di facilitare l'utilizzo del controllo in VisualStudio. Infatti una volta compilato il progetto, sarà disponibile fra gli altri controlli della casella strumenti, in una categoria con lo stesso nome del progetto, il controllo WebCustomControl1.
Trascinandolo su una pagina web, né verrà creata un'istanza, mediante l'aggiunta prima di una direttiva Register:

```
<%@ Register Assembly="WebControlLibrary"
Namespace="WebControlLibrary" TagPrefix="cc1" %>
```

E quindi con la creazione del tag del controllo:

```
<cc1:WebCustomControl1 ID="WebCustomControl1_1" runat="server"
/>
```

Selezionando il controllo nella pagina di design è inoltre possibile impostare le proprietà dall'apposita finestra, come un controllo standard.
Il fatto di creare uno o più controlli custom in un progetto separato, e dunque compilati in un nuovo assembly non è una scelta obbligata.

Nel caso in cui sia necessario implementare per esempio un solo controllo custom, senza doverlo condividere fra diverse applicazioni web, in ASP.NET 2.0 è possibile creare una normale classe, C# o VB che sia, posizionandola naturalmente nella cartella speciale App_Code ed utilizzandola poi come già visto in precedenza.

3.6 VALIDAZIONE DELL'INPUT

Quando una pagina web consente all'utente di immettere dei dati, ad esempio per riempire i campi di un modulo, è importante verificare che i dati inseriti siano corretti e validi.
ASP.NET mette a disposizione dei meccanismi incorporati per validare i dati forniti dall'utente, sotto forma dei cosiddetti controlli validatori.
I controlli validatori derivano dalla classe BaseValidator, che a sua volta deriva dalla classe Label, ed infatti il fine ultimo di tali controlli è quello di mostrare un'etichetta di testo, che illustri se e quale errore si sia verificato nella pagina.
Ogni controllo validatore si riferisce ad un controllo di input, ad esempio una TextBox, ed ogni controllo di input può anche essere validato da più di un controllo, per esempio nel caso in cui la TextBox debba contenere un indirizzo e-mail obbligatorio, si potrebbe avere un controllo che verifichi la validità dell'indirizzo inserito, ed uno che verifichi che un valore sia stato effettivamente inserito nella casella.

3.6.1 Associare un validatore ad un controllo

La classe BaseValidator possiede la proprietà ControlToValidate, che consente di impostare il controllo di input da validare.
La proprietà deve essere impostata utilizzando il valore della proprietà ID del controllo destinatario della validazione.
Non tutti i controlli di input possono essere associati ad un controllo validatore.
Le classi che è possibile validare sono identificati dall'attributo Vali-

dateProperty, e fra questi quelli tipicamente utilizzati sono TextBox, ListBox, DropDownList, FileUpload, RadioButtonList, mentre ad esempio non sono associabili ad un controllo validatore i controlli CheckBox, RadioButton, CheckBoxList.

3.6.2 Campi obbligatori: il controllo RequiredFieldValidator

Quando è necessario garantire che un dato controllo venga riempito, il controllo validatore RequiredFieldValidator compie questa operazione in maniera immediata.
Basta impostare la proprietà ControlValidate con l'ID del campo di input obbligatorio:

```
<asp:TextBox ID="txtUsername" runat="server"></asp:TextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
runat="server"
    ControlToValidate="txtUsername"
 ErrorMessage="Username obbligatoria">
</asp:RequiredFieldValidator>
```

Se al posto della proprietà ErrorMessage, o in concomitanza ad essa, si imposta la proprietà Text, il valore di quest'ultimo verrà visualizzato in ogni caso all'esecuzione della pagina, ad esempio si può impostare un valore che indichi all'utente di inserire lo username, e verrà invece nascosta se l'utente inserisce un valore nel campo corrispondente al ControlToValidate.
Si vedrà parlando del controllo ValidationSummary come e perché utilizzare contemporaneamente Text ed ErrorMessage.

3.6.3 Confronto di valori: il controllo CompareValidator

Il controllo CompareValidator consente di eseguire un confronto fra il valore inserito in un controllo, ed un altro valore, secondo diverse

operazioni, ad esempio quella di uguaglianza, ma anche per verificare che il valore sia minore o maggiore di un altro valore.
Il valore con cui eseguire il confronto può essere una costante, o anche un valore letto da un altro controllo nello stesso contenitore.
Il seguente codice mostra come aggiungere un CompareValidator per verificare che una TextBox contenga un valore numerico minore dell'anno attuale:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:CompareValidator ID="CompareValidator1" runat="server"
ControlToValidate="TextBox1"
ErrorMessage="valore non valido per l'anno di nascita"

Operator="LessThanEqual" ValueToCompare=2006 Type="Integer">
</asp:CompareValidator>
```

Innanzitutto è necessario indicare il controllo da validare, mediante il suo ID, in questo caso TextBox1, quindi la proprietà ErrorMessage contiene la stringa da mostrare nel caso di valore inserito non valido. Il confronto da eseguire viene determinato dalla proprietà Operator, che può essere impostata ad uno dei valori dell'enumerazione ValidationCompareOperator, che definisce i classici operatori di confronto, in questo caso si è scelta la possibilità che il valore inserito sia minore o uguale del valore impostato nella proprietà ValueToCompare, la costante 2006. Trattandosi di un numero, è necessario impostare l'attributo Type al valore Integer.
Questo farà altresì in modo che eventuali valori non numerici vengano automaticamente trattati come valori non validi e quindi verrà mostrato anche in questo caso il messaggio di errore.
Se invece di un valore si vuol eseguire il confronto sul contenuto di un altro controllo, invece della proprietà ValueToCompare basterà utilizzare la proprietà ControlToCompare.
Un classico caso di utilizzo di tale proprietà è quello che prevede il

confronto fra una password inserita in una casella di testo, e quella di conferma:

```
<asp:Label ID="Label1" runat="server"
Text="Password"></asp:Label>
<asp:TextBox ID="TextBoxPassword" runat="server"
TextMode="Password">
</asp:TextBox>
<br />
<asp:Label ID="Label2" runat="server" Text="Conferma
Password"></asp:Label>
<asp:TextBox ID="TextBoxConfermaPassword" runat="server"

TextMode="Password">
</asp:TextBox>
<br />
<asp:CompareValidator ID="CompareValidator2" runat="server"

ControlToCompare="TextBoxPassword"
ControlToValidate="TextBoxConfermaPassword"
ErrorMessage="Le password devono
coincidere"></asp:CompareValidator>
```

Il contenuto del controllo TextBoxConfermaPassword viene confrontato con quello del controllo TextBoxPassword, secondo l'operatore Equal.

3.6.4 Intervalli di valori: il controllo RangeValidator

Il controllo RangeValidator consente di verificare che il valore inserito in un controllo sia compreso in un intervallo di valori.

```
<asp:TextBox ID="txtStipendio" runat="server"></asp:TextBox>
```

```
<asp:RangeValidator ID="RangeValidatorStipendio" runat="server"
ControlToValidate="txtStipendio" ErrorMessage="Inserire un valore fra
100 e 10000" MaximumValue="10000" MinimumValue="100"

Type="Integer">
</asp:RangeValidator>
```

Anche in questo caso viene impostato il tipo Integer mediante la proprietà Type, mentre l'intervallo di valori consentito è determinato dai valori dati a MinimumValue e MaximumValue.

3.6.5 Espressioni regolari: il controllo RegularExpressionValidator

In alcuni casi è necessario verificare che l'input dell'utente rispetti un certo formato, basti pensare all'indirizzo email da inserire in ogni modulo di registrazione, e che naturalmente ha un formato ben definito. Le espressioni regolari in tali occasioni sono la soluzione ideale, ed il controllo RegularExpressionValidator, automatizza la verifica del formato inserito in un campo, utilizzando un'espressione regolare impostata tramite la proprietà ValidationExpression.


```
<asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>

<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
runat="server"
    ControlToValidate="txtEmail" ErrorMessage="email non valida"

    ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)
 *\.\w+([-   .]\w+)*">
</asp:RegularExpressionValidator>
```

Visual Studio permette di impostare alcune delle espressioni regolari più comuni scegliendole da un elenco.

L'espressione regolare per la validazione dell'email appena utilizzata ne è un esempio.

3.6.6 Validazione personalizzata: il controllo CustomValidator

Se nessuno dei precedenti controlli validatori fornisse le funzionalità ricercate, o semplicemente se si vuol validare un controllo che non è marcato con l'attributo ValidationProperty, come una CheckBox, è possibile utilizzare un controllo CustomValidator.
Tale controllo consente di utilizzare sia un metodo lato client, scritto per esempio in Javascript, sia un metodo di validazione eseguito sul server.
Nel primo caso è necessario impostare il nome della funzione da eseguire nella proprietà Client

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
ControlToValidate="txtCodice"

    ErrorMessage="Codice non valido" ValidateEmptyText="True"

ClientValidationFunction="ControllaCodice">
</asp:CustomValidator>
```

In questo caso verrà eseguita la funzione javascript ControllaCodice, mentre la proprietà ValidateEmptyText indica che il contenuto deve essere verificato anche se il campo fosse vuoto.
La funzione indicata deve rispettare una particolare firma, di cui il seguente è un esempio valido:

```
<script type="text/javascript" language="javascript">
    function ControllaCodice(source, args)
    {
        if(args.Value=='123')
```

```
            args.IsValid=true;
        else args.IsValid=false;
    }
</script>
```

Deve dunque essere presente un parametro source che conterrà il ControlToValidate, ed un parametro args da utilizzare per ricavare il valore e per impostare la validità. Dal lato del server, si può utilizzare un'analoga controparte, anche contemporaneamente alla funzione lato client. Basta gestire l'evento ServerValidate:

```
protected void CustomValidator1_ServerValidate(object source,
ServerValidateEventArgs args)
{
    if (args.Value == "123")
        args.IsValid = true;
    else
        args.IsValid = false;
}
```


È però possibile tralasciare di impostare la proprietà ControlToValidate e quindi ricavare lato server i valori contenuti in tutti i controlli che si vuole, dato che in tal caso la proprietà Value del parametro args sarà uguale ad una stringa vuota.

3.6.7 Riepilogare gli errori: il controllo ValidationSummary

Il controllo ValidationSummary permette di riepilogare in una Label tutti gli eventuali errori trovati dai validatori di una pagina.
È possibile mostrare la lista di errori in differenti modi, definiti dall'enumerazione ValidationSummaryDisplayMode, impostando la proprietà DisplayMode del controllo. Per default essa è pari a BulletList, cioè viene utilizzato un elenco puntato. È possibile anche fornire

un testo di intestazione, mediante la proprietà HeaderText. Oltre alla Label sulla pagina, in corrispondenza del ValidationSummary, è possibile visualizzare una MessageBox con lo stesso riepilogo.

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
ShowMessageBox="true" HeaderText="Errori trovati" />
```

Il riepilogo degli errori, cioè il controllo ValidationSummary, viene visualizzato solo se c'è almeno un errore nella pagina. Combinando le proprietà ErrorMessage e Text dei controlli validatori, ed un controllo ValidationSummary, è possibile così costruire diversi scenari. Per esempio si può impostare contemporaneamente la proprietà Text e la ErrorMessage, ed in tal modo, al verificarsi di un errore, in corrispondenza del controllo Validator relativo ad un campo verrà visualizzato il contenuto di Text, mentre il contenuto di ErrorMessage sarà visualizzato come elemento della lista del ValidationSummary.
È questo forse lo scenario più diffuso, utilizzando ad esempio un asterisco o un'abbreviazione come Text, ed un messaggio di errore esplicito per la ErrorMessage. La figura seguente mostra una pagina web con una serie di errori mostrati sia con singoli controlli Validator, sia con un ValidationSummary, e anche con una MessageBox.

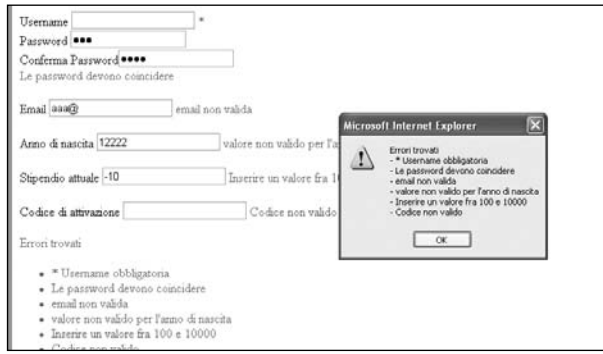

Figura 3.8: Riepilogo degli errori di validazione

LAYOUT DELLE PAGINE

Una delle funzionalità più desiderate dagli sviluppatori ASP.NET 1.1 era la possibilità di definire un formato comune per le pagine, ed utilizzarlo in maniera semplice ed immediata.
Per esempio si supponga di voler creare un sito web con una barra in alto contenente il menù principale, una barra in basso come piè di pagina contenente informazioni di copyright o altri link, ed una barra di navigazione laterale.
Nella versione 1.1, tale obiettivo era raggiungibile inserendo dei controlli utente personalizzati, ognuno dei quali implementasse ad esempio il menù, la barra del piè di pagina, o la barra di navigazione laterale.
Tali controlli dovevano essere ripetuti in ogni pagina, ad esempio inserendoli in tabelle per facilitare il loro posizionamento.
Con applicazioni di una certa dimensione tale copia e incolla poteva portare ad errori, ed inoltre costringeva ad un faticoso lavoro di manutenzione nel caso di modifiche al layout stesso, ad esempio per spostare o ridimensionare la tabella che fungeva da contenitore. ASP.NET 2.0 introduce un nuovo concetto di template, basato sulle master page

4.1 LE MASTER PAGE

Le master page sono un semplice ed efficace mezzo per fornire alle pagine di un'applicazione Web, un formato comune. I concetti fondamentali sono l'esistenza di un file che definisce tale formato e che avrà l'estensione .master, e di diverse pagine aspx che invece definiscono il contenuto, e che dunque sono chiamate content page.
Basta includere nel file .master qualunque elemento, controllo utente, formattazione, che è necessario ripetere su ogni pagina del sito, poi aggiungere i contenuti alle content page, ed il motore ASP.NET combinerà a runtime i due elementi per creare l'aspetto condiviso da ogni pagina dell'applicazione.

4.2 CREARE UNA MASTER PAGE

Per aggiungere una master page ad un'applicazione esistente, basta selezionare da Visual Studio il progetto, fare clic con il tasto destro, e poi sulla voce Add New Item.
A questo punto è possibile scegliere il tipo Master Page e dare un nome alla pagina. Nella parte html verrà creato un codice simile al seguente:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="Capitolo5_MasterPage" %>
  <html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
  <title>Untitled Page</title>
 </head>
<body>
  <form id="form1" runat="server">
  <div>
     <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">

     </asp:contentplaceholder>
  </div>
  </form>
 </body>
</html>
```

Il file è molto simile a quello di una pagina aspx, ma naturalmente inizia con una direttiva Master, seguita da una serie di attributi, ad esempio Language per indicare il linguaggio da utilizzare per il codice.
Il resto è identico a quello che si trova in una pagina aspx.
Quindi possono essere inseriti tag html, web control, controlli personalizzati, per definire l'aspetto che avrà la pagina master, e dunque tutte le content page che la utilizzeranno come template.

Nella pagina master precedente è possibile notare il tag asp:contentplaceholder, che definisce l'area in cui una content page può inserire il suo contenuto personale.
È possibile inserire in una pagina master tutti i ContentPlaceHolder desiderati, ad esempio per creare aree differenti in cui inserire barre di navigazione, menù, o altre sezioni.
Al di fuori dei ContentPlaceHolder è ora possibile creare l'aspetto che le pagine erediteranno dalla loro Master.
Supponendo di voler dare un titolo comune ed un piè di pagina a tutte le pagine, con un placeholder per il contenuto centrale ed uno per una barra laterale, è possibile creare una Master Page in questa maniera:

```
<form id="form1" runat="server">
  <div>
     <h1>Esempio di Master Page</h1>
     <table>
     <tr>
     <td style="width:70%">
  <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
          </asp:contentplaceholder>
     </td>
     <td>
<asp:contentplaceholder id="ContentPlaceHolderNav" runat="server">
          </asp:contentplaceholder>
     </td>
     </tr>
     </table>
     <h2>
        Piè di pagina
     </h2>
  </div>
 </form>
```

4.3 CREARE LE CONTENT PAGE

Dopo aver definito una Master Page, è ora possibile utilizzare questo template ed applicarlo a tutte le pagine che si vuole.

In Visual Studio è possibile aggiungere visivamente una Content Page, per una data Master Page, in due differenti modi.

Il primo è il classico Add New Item sul progetto, e selezionare poi dalla finestra il tipo Web Form, abilitando la casella di controllo Select Master page (figura 4.1).

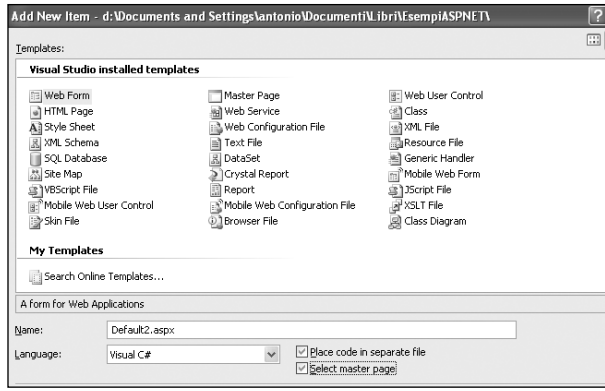

Figura 4.1: Creazione di una pagina di contenuto.

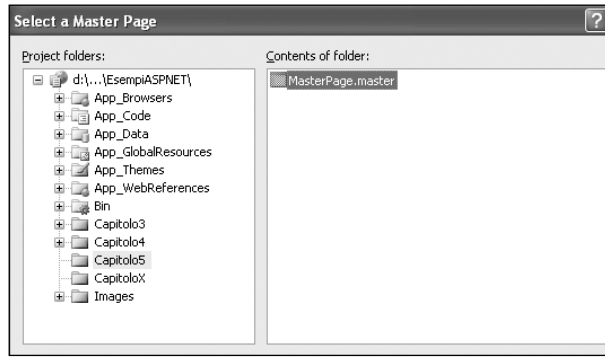

Figura 4.2: Selezionare la pagina Master.

Dopo il clic sul pulsante Add sarà possibile selezionare una Master Page esistente, definita cioè in precedenza nella stessa applicazione (figura 4.2).
La seconda possibilità è invece fare clic con il tasto destro su una Master Page nel Solution Explorer, e quindi selezionare la voce Add Content Page dal menù contestuale. In entrambi i casi verrà creata una Web Form con il codice seguente:

```
<%@ Page Language="C#"
MasterPageFile="~/Capitolo5/MasterPage.master"

AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="Capitolo5_Default" Title="Untitled Page" %>
<%@ MasterType VirtualPath= "~/Capitolo5/MasterPage.master"%>
<asp:Content ID="Content1"

ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp:Content>
<asp:Content ID="Content2"
ContentPlaceHolderID=ContentPlaceHolderNav runat="server">
</asp:Content>
```

La pagina avrà automaticamente l'attributo MasterPageFile impostato al percorso della Master Page scelta.
Ed inoltre è possibile notare la presenza del tag asp:Content mentre non c'è questa volta alcun tag form, nè alcun altro tag html.
In ciascuna Content Page possono essere inseriti tanti tag Content quanti sono i ContentPlaceHolder della Master Page associata, ed ognuno di essi dunque avrà un attributo ContentPlaceHolderID il cui valore è uguale all'ID del ContentPlaceHolder corrispondente.
Il grande aiuto fornito da Visual Studio è ora la possibilità di visualizzare in modalità design, la Content Page, con la parte derivata dalla sua Master page in grigio sullo sfondo (figura 4.3).

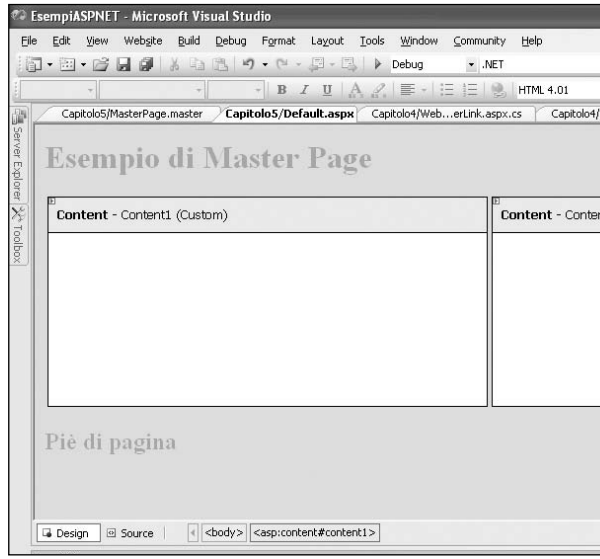

Figura 4.3: ContentPage in modalità di design.

Ogni ContentPage inoltre può utilizzare un linguaggio di programmazione diverso da quello della propria Master Page, così è possibile integrare un team che sviluppa Content Page in C# con quello che invece ha creato le Master Page in VB.
Analogamente è possibile mischiare i modelli code inline e code behind.

4.3.1 Impostare diverse Master Page

Ogni ContentPage dell'applicazione Web può usare una Master Page differente. Oltre che tramite l'attributo MasterPageFile della direttiva Page, è possibile impostare la pagina Master tramite il file di configurazione web.config, utilizzando l'elemento pages:

```
<configuration>
    <system.web>
```

```
        <pages masterPageFile="~/MasterPage.master" />
    </system.web>
</configuration>
```

In questa maniera tutte le pagine dell'applicazione utilizzeranno la MasterPage specificata.
Se si vuol usare una diversa Master Page per delle pagine contenuta in una data directory, è possibile specificare quest'ultima tramite l'elemento location.
In questo modo si potranno avere diverse sezioni del sito con aspetto differente.

```
<configuration>
    <location path="AreaRiservata">
        <system.web>
            <pages masterPageFile="~/MasterReserved.master" />
        </system.web>
    </location>
</configuration>
```

Lo stesso elemento location può anche specificare una singola pagina aspx:

```
<location path="pagina.aspx">
        <system.web>
        <pages masterPageFile="~/MasterPagina.master" />
        </system.web>
</location>
```

4.3.2 Accedere alla Master Page

Da una qualunque Content Page è possibile accedere alle proprietà della propria Master Page, in maniera da personalizzare l'aspetto. Ad esempio, supponiamo che la Master Page abbia una Label destina-

ta a contenere il nome della sezione in cui ci si trova nel sito, che potrebbe essere quello della Content Page.
Per accedere ad un controllo della pagina Master è possibile utilizzare il metodo FindControl tramite la proprietà Master:

```
protected void Page_Load(object sender, EventArgs e)
{
    Label labTitle = Master.FindControl("LabelTitle") as Label;
    if (labTitle != null)
        labTitle.Text = "Titolo della content page";
}
```

Il procedimento visto è necessario perché la proprietà Master restituisce un oggetto della generica classe MaserPage.
Un'altra possibilità è dunque quella di tipizzare la MasterPage utilizzando nella ContentPage la direttiva MasterType:

```
<%@ MasterType VirtualPath= "~/Capitolo5/MasterPage.master"%>
```

In tal modo si potranno utilizzare le proprietà ed i metodi specifici della Master Page.
Se in quest'ultima fosse presente la proprietà seguente:

```
public Label TitleLabel
{
    get
{
    return LabelTitle;
  }
  set{
    LabelTitle = value;
  }
}
```

Essa si potrà utilizzare dalla Content Page come una qualunque altra proprietà.
Aggiungendo ad esempio una ListBox con i nomi di tre diversi font, e la proprietà AutoPostBack impostata a true:

```
<asp:Content ID="Content1"
ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
  <asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
 OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
      <asp:ListItem>Arial</asp:ListItem>
      <asp:ListItem>Courier New</asp:ListItem>
      <asp:ListItem>Garamond</asp:ListItem>
  </asp:ListBox>
</asp:Content>
```

Si può selezionare un nuovo font per la Label ed impostarlo semplicemente nel seguente modo:

```
protected void ListBox1_SelectedIndexChanged
(object sender, EventArgs e)
{  Master.TitleLabel.Font.Name = ListBox1.SelectedItem.Text;   }
```


4.4 LAVORARE CON I TEMI

ASP.NET 2.0 ha introdotto nelle applicazioni web il concetto di tema, familiare agli utenti di Windows XP, che possono cambiare l'aspetto dell'intero sistema operativo scegliendolo da un elenco ed in maniera completamente visuale. Un tema è dunque un insieme di skin, css, immagini, ed altri file che servono a dare in maniera uniforme un bell'aspetto ad un controllo, una pagina o un intero sito. Inoltre tale aspetto potrà essere facilmente modificato da ogni singolo utente in base alle sue preferenze personali scegliendo uno dei temi che si possono mettere a sua disposizione.

4.4.1 I file di un tema

Un tema è costituito da diverse tipologie di file posizionati in tante sottodirectory della cartella speciale App_Themes, una per ogni tema che si vuol rendere disponibile.
Ogni tema può contenere le seguenti tipologie di risorse:

- CSS: fogli di stile che contengono la definizione degli stili applicabili a qualunque elemento HTML di una pagina o di un controllo.
- Skin: uno skin è un insieme di proprietà e template che possono essere applicate ad un dato tipo di controllo, ed ogni controllo può avere diversi skin.
- Immagini: molti controlli possono essere personalizzati mediante l'utilizzo di immagini, ad esempio un pulsante può visualizzare sulla sua superficie un'immagine invece del classico testo.
- Template: oltre all'aspetto grafico, alcuni controlli possono essere personalizzati nel layout, cioè nel modo in cui viene definita la sua struttura, in maniera indipendente dal comportamento.
- Altre risorse: i temi possono essere personalizzati con tutte le risorse necessarie al funzionamento di un dato controllo.

4.4.2 Livelli di tema

Un tema può essere applicato a diversi livelli.
Se si vuol dare lo stesso aspetto, definito in un tema creato per l'applicazione, ad ogni pagina aspx e dunque a tutti i controlli in essa contenuti, si può agire a livello di file di configurazione web.config, utilizzando l'attributo theme nell'elemento pages:

```
<pages theme="Theme1"/>
```

L'esempio precedente presuppone l'esistenza di un tema denominato Theme1 e dunque di una sottocartella omonima posizionata nella cartella speciale App_Themes che contiene i file utilizzati dal te-

ma stesso. Se si vuol applicare un tema ad una pagina specifica è possibile utilizzare l'attributo Theme o StylesheetTheme della direttiva Page:

```
<%@ Page Language="C#" Theme="Theme1">
<%@ Page Language="C#" StylesheetTheme="Theme1">
```

Nel primo caso si agisce su tutte le proprietà definite nel tema, sovrascrivendo in pratica le eventuali proprietà già impostate nella pagina o nel controllo, il secondo invece consente una personalizzazione per differenza, nel senso che solo gli attributi non impostati ma personalizzati ad esempio da uno skin, saranno impostati secondo quanto definito dal tema. L'ultimo possibile livello di applicazione di un tema, come già anticipato, agisce a livello di un singolo controllo. In un file di skin è possibile personalizzare l'aspetto di un Button aggiungendo un contenuto come il seguente:

```
<asp:Button runat="server" BackColor="Black" ForeColor="Yellow">
</asp:Button>
```

A tutti i controlli Button, posizionati in una pagina a cui è applicato un tema, sarà applicato Black come colore di sfondo e Yellow come colore del testo.
È inoltre possibile creare uno skin con un identificatore SkinID che ne indica il nome univoco.

```
<asp:Button SkinID="SkinButton1" runat="server"
BackColor="Black"
ForeColor="Yellow">
</asp:Button>
```

Ed applicarlo ad un particolare controllo Button utilizzando l'attributo SkinID del controllo stesso:

```
<asp:Button SkinID=buttonSkin ID="Button1" runat="server"
Text="Button" />
```

In questo caso i temi devono essere attivati o da file di configurazione o a livello di pagina.
Se non si ricade in nessuno dei due casi, il tema verrà applicato al pulsante solo se esso venisse poi aggiunto programmaticamente, nel gestore dell'evento PreInit o in uno precedente:

```
protected void Page_PreInit(object sender, EventArgs e)
{
    this.Theme = "Theme1";
}
```

4.4.3 Creare un tema

Per creare un proprio tema è innanzitutto necessario realizzare l'appropriata struttura delle cartelle nell'applicazione web, a partire dalla cartella speciale App_Themes. In Visual Studio basta fare clic con il tasto destro sulla soluzione e quindi selezionare dal sottomenu Add ASP.NET Folder la voce Theme. Ciò creerà una sottocartella Theme1, che potrà essere rinominata con il nome scelto per il tema.
All'interno della cartella speciale App_Themes può essere creato un numero qualsiasi di sottocartelle, una per ogni tema dell'applicazione.

Creazione degli Skin

Per creare uno skin, che sarà contenuto in un file con estensione .skin, basta fare clic con il tasto destro sulla cartella del tema in cui inserire il file, e scegliere la voce Add New Item.
A questo punto bisogna selezionare il tipo Skin File e dare un nome al file. Non resta che aggiungere il contenuto al file di skin, aggiungendo per ogni controllo a cui si vuole applicare lo skin una definizione simile a quella che si darebbe per aggiungere il controllo ad una

pagina aspx.
Per esempio, quello sotto potrebbe essere un file skin che definisce l'aspetto per i controlli Calendar, per le Label, e per i Button:

```
<asp:Calendar runat="server" BackColor="Red" ForeColor="Yellow">
  <DayStyle BackColor="Blue" BorderColor="White" Wrap="True" />
  <DayHeaderStyle BackColor="Green" />
</asp:Calendar>
<asp:Label runat="server" BackColor="White" ForeColor="Red" Font-
Names="Arial" Font-Size="10pt">
</asp:Label>
<asp:Button SkinId="buttonSkin" runat="server" BackColor=Black

ForeColor="Yellow">

</asp:Button>
```

La definizione per i controlli Button utilizza anche l'attributo SkinID, e quindi sarà utilizzata da un controllo Button su una data pagina aspx solo se verrà impostata la proprietà SkinID, come visto precedentemente.

Creazione dei CSS

Una pagina aspx non è fatta solo di controlli server, ed in molti casi, ad esempio per formattare il classico codice HTML, sarà di estrema utilità l'utilizzo di file CSS, cioè dei fogli di stile.
Per utilizzare un CSS in un dato tema, basta ancora una volta aggiungere un nuovo elemento alla cartella del tema stesso, e selezionare il tipo Style Sheet.
La sintassi dei fogli di stile va al di là degli scopi del testo, e si rimanda per un approfondimento al sito ufficiale *http://www.w3.org/Style/CSS/*. Si ricorda però che Visual Studio .NET consente una definizione visuale degli stili, che si rifletterà naturalmente sul contenuto

del file .css, per esempio

```
A:link
{
    font-size: 12px;
    color: blue;
    font-family: Arial;
    text-decoration: none;
}
 A:hover
{
    font-size: 12px;
    color: blue;
    font-family: Arial;
    text-decoration: underline overline;
}
H1
{
    font-weight: bold;
    font-size: 14px;
    font-family: 'Arial Black';
}
```

Il precedente esempio mostra la definizione degli stili da applicare agli elementi H1, ai collegamenti ipertestuali, nello stato normale (link) e come devono apparire nel momento in cui su di essi si posiziona il mouse (hover).

Utilizzo delle immagini

Il primo passo per aggiungere delle immagini ad un tema, è quello della creazione di una cartella Images all'interno della cartella dello specifico tema.

Le immagini aggiunte alla cartella Images possono essere utilizzate

in differenti modi. Per esempio, all'interno di un file Skin, è possibile referenziare direttamente un'immagine nelle definizioni dei controlli:

```
<asp:TreeView runat="server" CollapseImageURL="images\minus.gif"
ExpandImageURL="images\plus.gif">
  <ParentNodeStyle ImageUrl="images\iconroot.gif"/>
  <RootNodeStyle ImageUrl="images\iconroot.gif"/>
  <LeafNodeStyle ImageUrl="images\iconleaf.gif"/>
</asp:TreeView>
```

Ed un controllo TreeView in una pagina aspx potrebbe apparire come nella figura seguente senza nessun'altra configurazione o impostazione di proprietà:

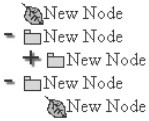

Figura 4.4: Applicare uno skin ad un albero.

ACCESSO AI DATABASE E DATA BINDING

Una delle tecnologie più frequentemente utilizzate in un'applicazione web di livello medio-alto, soprattutto a livello aziendale, è l'accesso ad un database. Così come .NET fornisce delle librerie per costruire un'interfaccia grafica, le Windows Forms, o una tecnologia con relative librerie per sviluppare applicazioni web (quella di cui parla il libro, ASP.NET!), esiste una tecnlogia che permette l'accesso a praticamente ogni tipo di database conosciuto, il cui nome è ADO.NET.

5.1 ADO.NET

ADO.NET è l'erede della vecchia tecnologia ADO, il cui acronimo stava per Active Data Objects. Essa è la tecnologia .NET di accesso ai database, ed esattamente fornisce un insieme di cosiddetti data provider, che permettono le fondamentali operazioni eseguite su una sorgente dati: connettersi, eseguire dei comandi, ricavare ed utilizzare i risultati. ASP.NET 2.0 introduce un nuovo strato al di sopra dei Data Provider, per facilitare il compito dello sviluppatore, e per ottenere una maggiore astrazione sciogliendosi in pratica dalla dipendenza verso il database fisico. Tale nuovo strato è costituito dai controlli DataSource e dai controlli DataBound, e permette di lavorare con i dati in maniera dichiarativa senza praticamente scrivere una sola riga di codice dedicata all'accesso al database.

5.2 DATA BINDING IN ASP.NET 2.0

Una delle più potenti funzionalità dei controlli ASP.NET, già dalla versione 1.1, è la possibilità di "legare" delle collezioni di oggetti o sorgenti di dati ad alcuni controlli web, che supportano appunto il data binding, senza scrivere quantità di codice esagerate, anzi in maniera praticamente immediata ed a tempo di esecuzione. ASP.NET

2.0 ha potenziato ed esteso il concetto di data binding, introducendo nuovi controlli DataSource, che si occupano della gestione dell'accesso alla sorgente dati, e nuovi controlli data-bound, cioè che supportano il data binding verso una sorgente per mezzo di un DataSource.

5.2.1 I controlli DataSource

I cinque controlli DataSource predefiniti inclusi in ASP.NET 2.0 sono dedicati ognuno ad uno specifico tipo di supporto dati. La tabella seguente ne fornisce una rapida descrizione

Controllo	Descrizione
SqlDataSource	Fornisce l'accesso ad una qualunque sorgente dati per la quale esista un ADO.NET Provider ad esempio SqlServer, Oracle, o anche database accedibili via ODBC o OleDB.
AccessDataSource	È un controllo DataSource dedicato ai database Microsoft Access.
ObjectDataSource	Permette di accedere a classi che implementano business object e le loro collezioni.
XmlDataSource	Fornisce l'accesso a dati in formato XML, sia leggendoli da file che dalla memoria.
SiteMapDataSource	I dati della mappa di un sito web possono iessere utilizzati come una altra sorgente dati utilizzando questo controllo.

Tabella 5.1: i cinque controlli DataSource

Tutte e cinque le classi derivano dalla DataSourceControl, e dunque condividono molte comuni funzionalità, inoltre è sempre possibile implementare la propria classe DataSource derivandola dalla stessa classe madre. Tutto ciò permette di poter utilizzare un controllo DataSource sia in maniera programmatica, sia in maniera dichiarativa,

quindi senza dover implementare una sola linea di codice.

5.2.2 I controlli Data Bound

ASP.NET include diversi controlli che supportano il Data Binding, e sono chiamati dunque controlli Data Bound. Tutti espongono due proprietà fondamentali, DataSource e DataSourceID, in quanto tutti derivano dalla nuova classe base BaseDataBoundControl. Altre proprietà sono poi specifiche di ogni controllo. Fra gli altri, i più utilizzati sono i controlli che derivano da ListControl: CheckBoxList, RadioButtonList, BulletedList, DropDownList, e ListBox, mentre altri controlli più avanzati includono TreeView, Menu, GridView, DataGrid, FormView, DetailsView.
La procedura per connettersi ad un database, e visualizzare dei dati ricavati da esso in un controllo DataBound è praticamente standard, quindi si vedranno un paio di esempi basati su qualcuno dei controlli appena elencati.

5.2.3 Il controllo SqlDataSource

Per mostrare il funzionamento di un SqlDataSource, si utilizzerà un database SqlServer, con una tabella Clienti, su cui effettuare una query per ricavare dei record. I risultati saranno mostrati in un controllo GridView. È dunque necessario prima di tutto creare un nuovo database oppure utilizzarne uno esistente, e naturalmente conoscerne le credenziali di accesso. In questo caso si assume l'utilizzo di un database Sql Server 2005, aggiunto direttamente all'applicazione web, nella directory App_Data, ma ognuno potrà utilizzare il proprio, semplicemente aggiustando la stringa di connessione nei passi successivi. La tabella Clienti del database è formata dalle seguenti colonne: IDCliente, Nome, Cognome, Indirizzo, Città, Email, Telefono. Il campo IDCliente è inoltre la chiave primaria della tabella.
Creando una connessione al database da Visual Studio, il file di configurazione verrà modificato, in maniera da riflettere la presenza della nuova informazione, per esempio, utilizzando un database man-

tenuto nel file database.mdf, si avrà il seguente nuovo elemento con nome ConnectionString:

```
<configuration>
  …
    <connectionStrings>
        <add name="ConnectionString"
        connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=
|DataDirectory|\Database.mdf;Integrated Security=True;User
Instance=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
<configuration>
```

È possibile adesso creare un controllo SqlDataSource, aggiungendolo in maniera dichiarativa ad una pagina aspx, referenziando la stringa di connessione precedente, e definendo una query sql di SELECT per ricavare tutte le righe della tabella Clienti ordinate per Cognome:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
 ConnectionString="<%$ ConnectionStrings:ConnectionString
%>"
SelectCommand="SELECT [IDCliente], [Nome], [Cognome], [Indirizzo],
[Città],
    [Email], [Telefono] FROM [Clienti] ORDER BY [Cognome]">
</asp:SqlDataSource>
```

5.2.4 Visualizzare i dati in una GridView

Il controllo GridView è uno dei nuovi controlli di ASP.NET 2.0, che permette di mostrare dei dati, non solo ricavati da un database, in una efficace e classica grglia. Utilizzare il controllo SqlDataSource per

mostrare le righe della tabella in un controllo DataBound come GridView è semplice come assegnare la proprietà DataSourceID.
Nella stessa pagina contenente il controllo SqlDataSource1 si aggiunga allora il codice seguente:

```
<asp:GridView ID="GridView1" runat="server"
    AutoGenerateColumns="True" DataSourceID="SqlDataSource1">
</asp:GridView>
```

Eseguendo la pagina, verrà riempita la griglia con tutte le colonne ricavate dalla query, e ciò grazie all'impostazione AutoGenerateColumns impostata a true, che indica di generare automaticamente tutte le colonne. Naturalmente è possibile utilizzare solo determinate colonne, creando un elemento BoundField all'interno dell'elemento padre Columns per ogni colonna da visualizzare:

```
 <asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="False"
DataSourceID="SqlDataSource1">
    <Columns>
  <asp:BoundField DataField="IDCliente" HeaderText="IDCliente"
InsertVisible="False"
            ReadOnly="True" />
    <asp:BoundField DataField="Nome" HeaderText="Nome" />
 sp:BoundField DataField="Cognome" HeaderText="Cognome" />
  <asp:BoundField DataField="Indirizzo" HeaderText="Indirizzo" />
    </Columns>
</asp:GridView>
```

5.2.5 Aggiornare ed eliminare i dati

Visualizzare i dati in molte applicazioni reali, non è sufficiente, e si vuole dare la possibilità di modificare le informazioni presenti nel database o di cancellare uno o più record.

La GridView permette questa possibilità, ma prima di tutto è necessario modificare anche il controllo SqlDataSource in maniera da consentire una query UPDATE di uno o più record ed analogamente una DELETE.

Innanzitutto è necessario definire i due nuovi UpdateCommand e DeleteCommand, contenenti le relative istruzioni SQL:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"

ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
SelectCommand="SELECT [IDCliente], [Nome], [Cognome],
[Indirizzo], [Città], [Email], [Telefono] FROM [Clienti] ORDER BY

[Cognome]"
UpdateCommand="UPDATE Clienti SET Nome = @Nome, Cognome
= @Cognome, Indirizzo = @Indirizzo WHERE IDCliente=@IDCliente"

        DeleteCommand="DELETE FROM Clienti WHERE
IDCliente=@IDCliente">
        <UpdateParameters>

            <asp:Parameter Name="Nome" />
            <asp:Parameter Name="Cognome" />
            <asp:Parameter Name="Indirizzo" />
            <asp:Parameter Name="IDCliente" />
        </UpdateParameters>
        <DeleteParameters>
            <asp:Parameter Name="IDCliente" />
        </DeleteParameters>
    </asp:SqlDataSource>
```

Per ogni parametro delle query di aggiornamento ed eliminazione, deve essere definito un corrispondente parametro, all'interno degli

elementi UpdateParameters e DeleteParameters, facendo corrispondere l'attributo Name al nome del parametro.
Perché le query funzionino correttamente è inoltre necessario informare la GridView sulla colonna da utilizzare come chiave.
Ciò è semplicemente fattibile impostando l'attributo DataKeyNames con il nome della colonna che funge chiave primaria, cioè IDCliente.
Deve essere poi possibile modificare lo stato della GridView, utilizzando dei pulsanti di Edit e di Delete..
In questo caso, si aggiungeranno, analogamente a quanto fatto per le colonne di dati, due colonne CommandField, come mostrato di seguito:

```
<asp:GridView ID="GridView1" runat="server"

AutoGenerateColumns="False"
    DataKeyNames="IDCliente" DataSourceID="SqlDataSource1">
        <Columns>
   <asp:BoundField DataField="IDCliente" HeaderText="IDCliente"
InsertVisible="False"
            ReadOnly="True" />
      <asp:BoundField DataField="Nome" HeaderText="Nome" />
 <asp:BoundField DataField="Cognome" HeaderText="Cognome"
/>
    <asp:BoundField DataField="Indirizzo" HeaderText="Indirizzo"
/>
          <asp:CommandField ShowEditButton="True" />

          <asp:CommandField ShowDeleteButton="True" />
        </Columns>
</asp:GridView>
```

Eseguendo la pagina, verranno adesso visualizzate due nuove colonne Edit e Delete.

Figura 5.1: Una GridView in modalità di aggiornamento.

Facendo clic sulla seconda, il record mostrato nella corrispondente riga sarà eliminato, mentre facendo clic su Edit la GridView passerà in modalità di Update, permettendo la modifica dei dati di una riga, come mostrato nella figura seguente:

NOTE

NOTE

NOTE

NOTE

NOTE

NOTE

NOTE

i libri di

ioPROGRAMMO

IMPARARE ASP.NET

Autore: Antonio Pelleriti

EDITORE
Edizioni Master S.p.A.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona ind. - 87036 Rende (CS)

Realizzazione grafica:
Cromatika Srl
C.da Lecco, zona ind. - 87036 Rende (CS)

Art Director: Paolo Cristiano
Responsabile di progetto: Salvatore Vuono
Coordinatore tecnico: Giancarlo Sicilia
Illustrazioni: Tonino Intieri
Impaginazione elettronica: Francesco Cospite

Servizio Clienti
☎ **Tel. 02 831212 - Fax 02 83121206**
@ **e-mail: customercare@edmaster.it**

Stampa: Grafica Editoriale Printing - Bologna

Finito di stampare nel mese di Settembre 2006