IL PRIMO MAGAZINE ITALIANO DEDICATO A VISUAL BASIC, C#, .NET, ACCESS E SQL SERVER

Poste Italiane S.p.A. - Spedizione in Abbonamento Postale - D.L. 353/2003 conv. in L. 27/02/2004 n°46 art.1,comma 1, DCB PERUGIA

VISUAL BASIC & .NET JOURNAL

n.61

Gennaio/Febbraio 2005 - bimestrale - Anno 11 - Euro 7,75

Tecniche per il WEB

Sviluppare con XHTML, CSS e ASP.
Scrivere applicazioni Web con il code inline

Beginner
Introduzione al multithreading

Tecniche
Implementare l'UNDO in un'applicazione Windows Forms

Strumenti
Utilizzare Access con Visual Source Safe

Database
Azioni referenziali con SQL Server 2000

Architect's Corner
Definizione esplicita dei [WebMethod]

Software Engineering
La programmazione generica

Architetture
Service Oriented Architecture

GRUPPO EDITORIALE INFOMEDIA

EDITORIALE

Dal patrimonio informativo alla conoscenza

Ora che le beta di Whidbey - oops, volevo dire del nuovo Visual Studio 2005 - sono disponibili pubblicamente, diventa sempre più evidente che Microsoft vuole continuare a spingere sull'approccio RAD (Rapid Application Development) nello sviluppo di applicazioni con il .NET Framework. Con la prossima versione di Visual Studio, ad esempio, è possibile costruire form anche complessi per il data-entry, sia per le applicazioni Win32 che per le applicazioni ASP.NET. Le applicazioni generate in questo modo supportano il master-detail e le ricerche, e altro ancora.

Com'è nella tradizione di Visual Studio .NET, questi form sono ottenuti generando codice - a differenza di quanto avveniva con VB6, in cui i form erano una specie di "scatole nere" su cui non era possibile intervenire - e questo è ovviamente un bene, in quanto permette di intervenire sul risultato per adattarlo alle proprie esigenze. Addirittura, per agevolare il lavoro del generatore di codice di VS.NET, i linguaggi VB.NET e C# sono stati espansi per supportare le cosiddette *partial class*, ossia la possibilità di suddividere il codice di una classe (e quindi di un form) in due file sorgenti separati, in modo che le personalizzazioni aggiunte al codice generato da VS.NET non siano perse quando il codice del form viene rigenerato al passaggio successivo. Lavoro in questo settore da troppi anni per non essere un po' sospettoso ogni volta che vedo una mirabolante demo durante la quale, in pochi minuti e davanti ai miei occhi, viene creata una piccola applicazione di data-entry e reporting. Il sospetto è che nelle applicazioni reali tutte queste feature non siano usabili realmente. Per capire cosa intendo, basta chiedersi quanti programmi commerciali VB3 hanno mai realmente usato il binding e il data control. In effetti, la mia impressione è che in Visual Studio 2005 vi siano alcune "forzature". Ad esempio, la toolbar di ASP.NET 2.0 è affollata di controlli - per esempio, per il login e la navigazione del sito - che potrebbero essere realizzati in pochi minuti e senza alcun problema, ma soprattutto con maggiore libertà grafica e funzionale. Però disporre di questi controlli nella toolbar permette di affermare che è possibile scrivere dei piccoli siti funzionali senza scrivere neanche una riga di codice, in modalità completamente RAD. E potrei fare esempi simili per le funzioni di data binding.

La domanda è: quanti di questi controlli e di queste feature sono realmente in grado di agevolare il lavoro di chi sviluppa applicazioni professionali? Visual Studio 2005 (come pure la versione 2003) sono dei sistemi di sviluppo molto più complessi del vecchio VB6, che continua ad essere una opzione praticabile per scrivere applicazioni desktop non troppo sofisticate. Nella scrittura e nel test di una applicazioni client-server e multi-tier, la creazione di un form di data entry rappresenta una frazione minima del tempo complessivo, quindi perché accanirsi su questi aspetti relativamente secondari? Sono queste le feature che gli sviluppatori chiedono davvero? Per il momento posso solo notare che, se vi piace il RAD, Visual Studio 2005 ha davvero tantissimo da offrire. E ha molto da offrire anche a chi piace sviluppare "alla vecchia maniera", in quanto C# e (soprattutto) VB sono stati espansi e migliorati notevolmente, e anche il .NET Framework stesso contiene molte novità interessantissime. Quanto al dubbio amletico citato nel titolo, la risposta la darà il mercato e non ci vorrà molto per sapere se Microsoft ha scelto la strada giusta.

Francesco Balena
fbalena@codearchitects.com

VISUALBASIC & .NET JOURNAL
online.infomedia.it
n. 61 - gennaio/febbraio 2005
bimestrale - anno undicesimo

Direttore Responsabile
Marialetizia Mari *(mmari@infomedia.it)*

Direttore Esecutivo
Francesco Balena *(fbalena@infomedia.it)*

Technical Editor
Alberto Falossi *(afalossi@infomedia.it)*

Managing Editor
Renzo Boni *(rboni@infomedia.it)*

Collaboratori
Daniele Bochicchio
Gionata Aladino Canova
Dino Esposito, Jim Lee
Davide Mauri, Paolo Pialorsi
Francesco Quaratino
Ingo Rammer, Lorenzo Vandoni

GRUPPO EDITORIALE INFOMEDIA

Direzione
Natale Fino *(nfino@infomedia.it)*

Marketing & Advertising
Segreteria: 0587/736460
marketing@infomedia.it

Amministrazione
Sara Mattei
(amministrazione@infomedia.it)

Grafica
Manola Greco *(mgreco@infomedia.it)*

Technical Book
Lisa Vanni (*book@infomedia.it)*

Segreteria
Enrica Nassi
(info@infomedia.it)

Stampa
TIPOLITOGRAFIA PETRUZZI
Citta' di Castello (PG)

Ufficio Abbonamenti
Tel. 0587/736460 - Fax 0587/732232
e-mail: abbonamenti@infomedia.it
www.infomedia.it

Gruppo Editoriale Infomedia srl
Via Valdera P., 116 - 56038 Ponsacco (PI) Italia
Tel. 0587/736460 - Fax 0587/732232
red_vbj@infomedia.it
Sito Web www.infomedia.it

Visual Basic Journal *è una rivista di Gruppo Editoriale Infomedia S.r.l. Via Valdera P, 116 Ponsacco - Pisa.*
Registrazione presso il Tribunale di Pisa n. 20/1999

SOMMARIO

GENNAIO/FEBBRAIO

N.61

SPECIALE

BEGINNER

TECNICHE

STRUMENTI

DATABASE

RUBRICHE

ARCHITECT'S CORNER

SOFTWARE ENGINEERING

ENTERPRISE

OPINIONI

I MITI

Codice allegato

All'indirizzo ftp.infomedia.it/pub/VBJ *sono liberamente scaricabili tutti i listati relativi agli articoli pubblicati. La presenza di questa immagine indica l'ulteriore disponibilità, allo stesso indirizzo, di un progetto software relativo all'articolo in cui l'immagine è inserita. Il nome identifica la cartella sul sito ftp.*

Sviluppare siti con XHTML, CSS e ASP.NET

XHTML e CSS riportano la pagina Web alle origini, alla vera essenza di documento ipertestuale.
Basta un po' di buona volontà, un pizzico di lavoro e la lettura degli standard per rendersi conto che XHTML e CSS sono utili e ci aiutano ad essere più produttivi. Anche con ASP.NET.

di Daniele Bochicchio

La prima domanda che vi sarete sicuramente fatti dopo aver sentito parlare di XHTML suona più o meno così: "ma a cosa serve un altro HTML?". La risposta in casi come questo è molto semplice: *XHTML non è HTML* e, per questo motivo, mira a risolvere problemi di natura differente. Gran parte dei problemi del Web moderno, in particolare per quanto concerne accessibilità ed usabilità, sono dovuti alla poca rigidità di HTML, che ha consentito il proliferare di browser in grado di interpretare codice scritto in maniera poco ortodossa.

XHTML, come il nome stesso suggerisce, è completamente basato su XML. Sia XHTML che HTML mirano alla definizione della struttura di un documento ipertestuale, anche se con presupposti e metodologie di approccio al problema di natura completamente differente.

Differenze tra HTML e XHTML

Alla base di HTML, come di XML e quindi di XHTML, c'è l'ormai famoso *SGML* (Standard Generalized Markup Language), per cui è facile trovare molte somiglianze tra i due linguaggi. In realtà HTML non è assimilabile ad XML, perché viene meno alle due regole che ogni documento XML deve rispettare:

- essere *well formed*: sottostare ad alcune semplici regole, che vogliono che il documento sia formattato in un certo modo, senza tag annidati e con questi ultimi terminati in caso di tag "isolato";
- essere *valido* secondo la DTD/Scheda del caso.

Tanto per fare un esempio, nelle specifiche di HTML [1] non è vietato scrivere documenti che non siano well formed, tanto è vero che è perfettamente lecito annidare in maniera errata i tag:

```
<p><b><i>Testo</b></i></p>
```

Questo codice non è valido in XHTML, perché XML prevede come regola principale la buona formattazione dello stesso. Con XHTML il codice diventerà:

Daniele Bochicchio *è il content manager di ASPItalia.com, community che si occupa di ASP.NET, Classic ASP e Windows Server System. Si occupa di consulenza e formazione, specie su ASP.NET, e scrive per diverse riviste e siti. È Microsoft .NET MVP, un riconoscimento per il suo impegno a supporto delle community e per l'esperienza maturata negli anni. Il suo blog è all'indirizzo http://blogs.aspitalia.com/daniele/*

```
<p><b><i>Testo</i></b></p>
```

Altro punto di differenza è il case dei tag, che in HTML è indifferente, mentre in XHTML è minuscolo. Questo vuol dire che <P> è differente da <p> ed il primo di questo due in XHTML non è valido.

Non sono poi previsti tag che non siano chiusi:

```
<p>Paragrafo<P>Paragrafo
```

deve diventare:

```
<p>Paragrafo</p><p>Paragrafo</p>
```

Non ci possono essere tag "orfani" e quindi vanno terminati con il carattere / finale.
 diventa così
, <img…> diventa <img… /> e così via.

Tutti i valori degli attributi vanno racchiusi in una coppia di " o ' e non sono previsti attributi "semplici". Ad esempio l'attributo checked di una checkbox in XHTML diventa:

```
<input type="checkbox" id="chk1" checked="checked" />
```

Va cioè specificato sempre e comunque un valore, caratteristica ereditata da XML.

Infine l'attributo *name* di HTML è stato soppiantato dall'attributo *id*, che deve essere univoco nella pagina stessa.

Un discorso a parte merita il carattere &, anche noto come *ampersand*. Se presente nel testo (anche nei link) va usata l'entity corrispondente, che è &. Questo perché il carattere di ampersand è considerato riservato e viene utilizzato per inserire entity (ovvero caratteri particolari) all'interno di un documento, come à é, etc.

Una regola empirica è quella, in ogni caso, di scrivere tutto il codice in minuscolo, valore degli attributi incluso.

Le DTD di XHTML

XHTML allo stato attuale è disponibile in 5 principali DTD differenti, divise in 2 famiglie di versioni, la 1.0 e la 1.1:

- XHTML-1.0-Strict: rappresenta la versione che taglia con XHTML, deprecando molti attributi e tag di HTML 4.0
- XHTML-1.0-Transitional: è la versione più permissiva, che consente di mantenere una certa compatibilità con il passato, a discapito di un taglio più netto con le cattive abitudini.
- XHTML-1.0-Frameset / XHTML-1.1-Frameset: sono le versioni che prevedono l'utilizzo di frame
- XHTML-1.1: è l'unica versione di XHTML 1.1, che a differenza della 1.0 non prevede più una versione transitional.

La scelta di una o dell'altra dipende dalla reali necessità che abbiamo. Allo stato attuale, se si deve scrivere un documento ex-novo, XHTML 1.1 ha più senso dato che consente di mantenere una buona compatibilità con i browser che supportano solo HTML ed al tempo stesso di trarre il massimo vantaggio da quelli che supportano XHTML.

Figura 1 Un tipico layout a 3 colonne con Internet Explorer 6

Dove HTML ha fallito e XHTML è riuscito

HTML ha fallito per colpa dei browser, che hanno evidenziato al massimo il limite più grande, ovvero la mancanza di regole "rigide" nella definizione di un documento, e per colpa degli sviluppatori, che hanno cominciato ad usare HTML ed i suoi tag per la definizione della grafica anziché del contenuto. La maggior parte dei siti Web utilizza infatti le tabelle per creare la struttura del documento, venendo meno ad uno dei principi più utili nella definizione delle interfacce: la separazione tra struttura e sua rappresentazione grafica.

XHTML è basato su XML e garantisce la netta separazione tra struttura e rappresentazione, dato che nel mondo XML questo concetto è espresso in

maniera chiara ed è quasi alla base di tutto il resto.

Resta inteso che questa separazione è possibile anche con HTML, a patto di scrivere le pagine in una certa maniera, cosa che favorisce, nella quasi totalità dei casi, l'utilizzo di XHTML 1.1.

L'ultimo vero vantaggio di XHTML è che prevede una semantica precisa e rigorosa, cosa che rende il documento trasformabile in altri formati attraverso l'uso di XSLT[3] e XSL-FO[4]. È così possibile creare, da una pagina, un documento PDF semplicemente variando il "foglio di stile" associato, fare interrogazioni sui nodi (ad esempio, prendere tutti i collegamenti di un documento, o tutti i paragrafi) con una precisione garantita al 100%.

CSS e posizionamento degli elementi

Una parte centrale nel ruolo di XHTML è giocata dai CSS (Cascading Style Sheet), una tecnologia che ha come scopo quello di formattare i documenti delle famiglie (X)HTML.

La struttura di un documento CSS è estremamente semplice, dato che vengono definiti gli stili in base a tre diverse tipologie:

- per tag: si associa ad un tag, ad esempio p, uno stile;
- per id: si associa ad un oggetto del documento con un particolare id uno stile;
- per classe: si associa ad una classe un insieme di caratteristiche.

Ecco ad esempio un tipico foglio di stile:

```
body {background-color: gray; } /* associato ad un tag */
#rosso {color: red;} /* per id */
.giallo {color:yellow;} /* per classe */
```

Ed il relativo documento XHTML che ne fa uso:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http:
                        //www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="it">
<head>
<style type="text/css" media="screen">
/*<![CDATA[*/
@import url(test1.css );
/*]]>*/
</style>
<title>Esempio di CSS</title>
</head>
<body>

<p id="rosso">Testo rosso</p>
<p class="giallo">Testo blu</p>

</body>
</html>
```

L'unica vera novità rispetto ai documenti HTML che avete scritto è probabilmente il *doctype*, che è chiaramente differente. Per il resto, si nota subito, a prima vista il grande vantaggio di avere un layout pulito, costruito attraverso il foglio di stile e non con i tag, che servono solo per descrivere le informazioni.

Un layout a tre colonne con i CSS

L'unico modo per convincervi delle bontà della coppia XHTML-CSS è quello di mostrarvi un tipico esempio, ovvero un layout a tre colonne costruito con i CSS anziché con le tabelle. Questo genere di layout, che spesso è semplificato nell'equivalente a 2 o reso più complesso nella variante a 4 colonne, è molto diffuso ed è praticamente lo stile di base di quasi tutti i siti web. I CSS ci permettono di variare molto facilmente il layout del sito, tanto che spesso si trovano in giro siti multi-layout, dove la differenza di stile serve per dare un diverso posizionamento ai vari elementi della pagina. Esiste anche un progetto, CSS Zen Garden **[5]**, che rende questo concetto facile da apprezzare. Basta cambiare il CSS per dare un significato completamente diverso alle informazioni.

Non è così remota, infatti, l'eventualità di tenere un solo motore, inteso come gestore e generatore del codice XHTML, e poter personalizzare l'interfaccia semplicemente agendo sui CSS, senza per questo perdere tutte la flessibilità e le caratteristiche del layout. Una cosa del genere significa, anche e soprattutto, un tempo minore necessario alla manutenzione delle applicazioni.

A questo punto è necessario comunque sottolineare che ci possono essere problemi con alcuni browser, soprat-

Figura 2 Variando il CSS si può cambiare l'intera struttura e l'apparenza del documento XHTML

Listato 1 La struttura XHTML del layout a 3 colonne

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http:
//www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:
lang="it">
<head>
<style type="text/css" media="screen">
/*<![CDATA[*/
@import url(3colonne.css );
/*]]>*/
</style>
<title>Esempio di CSS con layout a 3 colonne</title>
</head>
<body>

<!-- colonna di destra -->
<div id="right">
<h2>Colonna di destra</h2>
<p>Lorem ipsum dolor sit amet, ...</p>
</div>

<!-- colonna di sinistra -->
<div id="left">
<h2>Colonna di sinistra</h2>
<p>Lorem ipsum dolor sit amet, ...</p>
</div>

<!-- colonna centrale -->
<div id="main">
<h2>Colonna centrale</h2>

<p>Lorem ipsum dolor sit amet, ...</p>

</div>

</body>
</html>
```

tutto i vecchi, e con Internet Explorer, che in certi casi interpreta alcune istruzioni in maniera fantasiosa. Si tratta comunque di piccoli difetti che possono essere compensati dal vantaggio di rendere possibile, a diversi dispositivi, l'accesso alle informazioni in maniera molto semplice. Tra un attimo ad esempio vedremo come trasformare il layout a 3 colonne nell'equivalente ad una colonna per dispositivi dalle ridotte capacità, come i palmari.

Perché il layout sia "fluido" ed i vari elementi interagiscano tra di loro, si utilizza una proprietà particolare, *float*, che come il termine suggerisce consente di far "galleggiare" un elemento rispetto a quelli che lo circondano.

Notate come, per questo motivo, il layout segua uno schema insolito:

```
<!-- colonna di destra -->
<div id="right">
</div>
```

```
<!-- colonna di sinistra -->
<div id="left">
</div>
```

```
<!-- colonna centrale -->
<div id="main">
</div>
```

Procediamo prima con la definizione dell'elemento di destra, che è impostato per galleggiare a destra dei contenuti che lo seguono, per poi inserire quello di sinistra, che galleggia dal verso opposto al primo elemento, finendo con l'elemento centrale, che andrà ad occupare lo spazio lasciato libero dai due elementi galleggianti (che occupano ognuno il 25% dello spazio disponibile). Il risultato è (**Listato 1** e **2**) un layout che si adatta anche alle dimensioni dello schermo, con pochissime istruzioni ed ancora meno sforzo.

Semplicemente giocando un po' con il CSS, si possono avere tanti risultati diversi, come quello mostrato nelle **Figure 1**, **2**, **3**, **4** che parte dalla stessa struttura, per definire una colonna di destra più piccola con un effetto grafico differente.

Dove i CSS non hanno eguali: un formato per ogni device

Qualcuno più attento avrà sicuramente notato la proprietà *media* del tag style.

Nel nostro esempio fa sempre riferimento a "screen", che è ovviamente quello utilizzato di default, perché associa lo stile corrente alla visualizzazione attraverso un browser normale.

È possibile specificare fogli di stile differenti per device differenti, ad esempio tv (per Media Center) o handheld (per palmari).

In quest'ultimo caso ci basta aggiunge una definizione come segue:

```
<link rel="stylesheet" type="text/css" media="handheld"
                                        href="handheld.css" />
```

In questo modo i browser per mobile device che riconoscono l'istruzione sono in grado automaticamente di utilizzare il foglio di stile pensato per questi dispositivi. Nel nostro esempio non facciamo altro che nascondere la colonna di sinistra, che non reputiamo utile, ma ovviamente si può anche cambiare il carattere, la sua dimensione o i colori utilizzati, per migliorare il contrasto.

Una variante molto interessante di questa tecnica consiste nel definire un foglio di stile apposito per la stampa, semplicemente impostando su "print" il valore dell'attributo media.

Gli altri possibili valori, con relative funzionalità, sono disponibili in **[6]**.

Figura 3 Un layout a 3 colonne con impostazione di un CSS specifico per mobile device

Consigli nell'uso di XHTML

Per poter sfruttare al meglio XHTML è necessario abbandonare le vecchie abitudini che con HTML sono la normalità. Tutta la formattazione va ovviamente fatta con i CSS ed andrebbe evitata, per quanto possibile, la formattazione in-line, ovvero l'utilizzo del tag style per definire le proprietà di un oggetto della pagina, perché questo rende meno il vantaggio di separare layout e struttura del documento.

È sempre meglio evitare l'utilizzo del posizionamento assoluto, in favore di quello relativo, come abbiamo visto negli esempi precedenti.

Questa scelta permette a browser dalle limitate capacità di mostrare comunque il contenuto, che è quello che conta, senza per questo sacrificare la leggibilità dello stesso. Inoltre è ovviamente opportuno fare attenzione a non specificare lo stesso ID per due elementi della pagina. Per essere sicuri che il proprio codice sia XHTML, è sufficiente utilizzare il validator del W3C **[7]**.

Un documento XHTML andrebbe servito con un content type particolare, application/xhtml+xml, che però IE non è in grado di interpretare a dovere.

Per sfruttare al meglio i browser che sono in grado di farlo viene spesso sfruttato il cosiddetto meccanismo di content negotiation **[8]**, per servire text/html o text/xml a browser non compatibili e questo formato a browser in grado di leggere nativamente XHTML, come Mozilla.

Nel caso si scelga di utilizzare text/xml o application/xhtml+xml, è necessario cambiare la definizione dello stile, che in questo caso viene definito come un vero e proprio XSL di un documento XML:

```
<?xml-stylesheet href="stile.css" type="text/css"
                                          media="screen "?>
```

Ovviamente deve essere anche presente la dichiarazione XML all'inizio del documento. Questa scelta ha ripercussioni anche sulle funzionalità della pagina, dato che il DOM in questo caso smette di funzionare e non si può utilizzare l'attuale implementazione di Javascript, che è pensato per l'albero di un documento HTML.

ASP.NET e XHTML

Purtroppo ASP.NET non va molto d'accordo con XHTML. Nell'ultimo Service Pack 1 della versione 1.1 alcune cose sono state sistemate, ma rimane comunque il problema che la classe HtmlTextWriter del namespace System.Web.UI, che è richiamata dall'adaptive rendering su molti browser (ma non tutti) non produce affatto codice XHTML, né tanto meno lo fanno i Web control. Sebbene il problema sarà risolto entro l'anno con il rilascio di ASP.NET 2.0, per i progetti già funzionanti (e nell'attesa) questo vuol dire rinunciare ad utilizzare gli standard, che invece rappresentano uno dei modi migliori di costruire applicazioni per il Web, nel pieno rispetto di qualsiasi tipo di browser. Gli interventi che si possono fare sono di natura differente ed hanno impatti e conseguenze su punti diversi del ciclo di esecuzione della pagina. Il migliore dal punto di vista delle performance, ma anche il più invasivo, è senza dubbio quello di ricostruire i control in modo che producano output XHTML **[9]**. Questo però vuol dire riscrivere tutta l'applicazione perché utilizzi control non standard, con tutti gli svantaggi del caso.

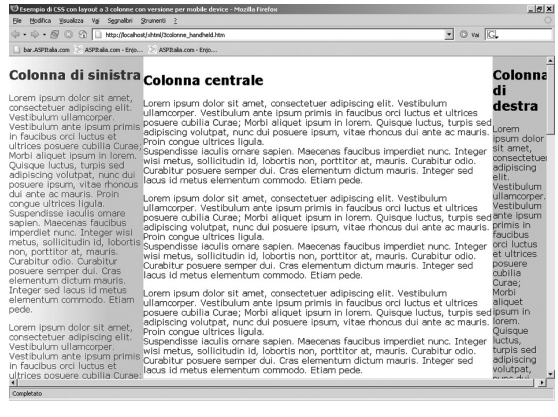


Figura 4 Il layout a 3 colonne visto all'interno di Mozilla FireFox. Come si può notare, il risultato è lo stesso

Listato 2 Il file CSS

```
body {background-color: #fff; font-family:Verdana;}
#left {width:25%; float:left; background-color:
#c0c0c0;}
#right {width:25%; float:right; background-color:
#f0f0f0;}
#main {padding: 10px;}
```

La meno performante, ma sicuramente più pratica perché ha un impatto prossimo allo zero su quanto già creato, consiste nell'utilizzare un HttpModule che si mette in ascolto di un determinato evento, per utilizzare un Filter **[11]** sull'oggetto Response in grado di manipolare l'output prima che venga inviato ad ASP.NET, con un approccio molto simile a quanto si farebbe per singola pagina o control **[10]**. Infine l'ultimo consiste nel creare una classe di base per la pagina **[12]** che sovrascriva l'output. Anche qui è necessario però cambiare manualmente tutte le pagine per fare in modo che la classe dalla quale ereditano sia quella in grado di cambiare l'output di ASP.NET.

> XHTML non è HTML. È XML e per questo ha una semantica rigorosa e tutti i vantaggi che un documento well formed si porta dietro.

In ogni caso, comunque, è necessario aggiornare il web.config **[13]** perché di default ASP.NET non riconosce né Mozilla né le ultime versioni di Opera, dato che l'uscita del prodotto è precedente al loro rilascio, e li tratta, per come funziona l'adaptive rendering, come browser sconosciuti, generando quindi HTML 3.2 e non HTML 4.0 (sul quale si devono fare le opportune modifiche). Ci sono dei control, come il calendar, che per loro natura sono molto difficili da "sistemare" ed il cui utilizzo andrebbe evitato, in sostituzione di custom control. Stesso discorso può essere fatto per i validator control, che obbligano ancora una volta a fare a meno di un'interessante ma poco standard funzionalità dal punto di vista del codice XHTML prodotto.

Conclusioni

Costruire applicazioni ASP.NET (o anche con altre tecnologie) che utilizzano XHTML e CSS consente di costruire siti Web "moderni", e soprattutto di prestare maggiore attenzione ad utenti che hanno difficoltà ad accedere ai contenuti, perché XHTML favorisce intrinsecamente una maggiore accessibilità; ad esempio riduce il rumore di fondo che un browser vocale deve "subire" rispetto ad una pagina che contiene anche la formattazione.

Infine, ma non ultimo, XHTML aiuta a ridurre il consumo di banda, dato che è frequente una diminuzione anche sostanziosa del peso della singola pagina. Da qualche mese sul portale nazionale che gestisco siamo passati ad un sistema completamente XHTML e table-less (senza tabelle), perdendo, a parità di informazioni mostrate sulle pagina, circa 50 kb.

Da oltre 85 kb di peso della vecchia versione, siamo passati ai 35 scarsi dell'attuale. Pensate al risparmio che ogni singolo utente vi porta in termini di costo della banda: se state cercando un motivo economico per passare a XHTML e CSS, l'avete trovato!

Riferimenti

[1] Specifiche HTML 4 in italiano
http://www.liberliber.it/biblioteca/w/world_wide_web_consortium/specifiche_html40/html/cover.html

[2] Specifiche XHTML
http://www.w3.org/TR/html/

[3] Specifiche XSL Transformations (XSLT)
http://www.w3.org/TR/xslt

[4] FOP
http://xml.apache.org/fop/

[5] CSSZenGarden.com
http://www.csszengarden.com/

[6] CSS 2.1 Candidate Recommendation - Media types
http://www.w3.org/TR/CSS21/media.html

[7] W3C Validator
http://validator.w3.org

[8] XHTML Content negotiation
http://www.w3.org/TR/xhtml-media-types/xhtml-media-types.html#application-xhtml-xml

[9] Vallid XHTML within .NET
http://www.liquid-internet.co.uk/content/dynamic/pages/series1article1.aspx

[10] A C# class to make your ASP.NET pages XHTML valid
http://www.charon.co.uk/content.aspx?CategoryID=28&ArticleID=53

[11] Modificare l'output di ASP.NET utilizzando Response.Filter
http://www.aspitalia.com/articoli/aspplus/responsefilter.aspx

[12] A C# class to make your ASP.NET pages XHTML valid
http://www.codeproject.com/aspnet/ASPNET2XHTML.asp

[13] Browser Tester 0.90
http://lab.aspitalia.com/lab.aspx?ID=5

Scrivere applicazioni ASP.NET con il code inline

Molto spesso si confonde la velocità di sviluppo con la pulizia e la precisione che un'applicazione Web riesce a raggiungere. La modalità code behind di Visual Studio può suscitare qualche perplessità in sviluppatori che vogliono tenere sotto controllo tutti i dettagli. L'approccio alternativo del code inline permette di sfruttare alcune caratteristiche di ASP.NET che VS non supporta, garantendo gli stessi vantaggi del modello "predefinito".

di Daniele Bochicchio

Chi ha aperto almeno una volta nella propria vita VS.NET sa che ogni pagina Web è organizzata attraverso una semplice struttura che prevede l'utilizzo di due entità distinte: la pagina stessa (o meglio, il codice HTML) ed il code behind. Quest'ultimo è il codice che "programma" la pagina e non è altro che una semplice classe che eredita da System.Web.UI.Page. La classe del code behind è poi a sua volta ereditata dalla pagina ASPX. Questo modello è considerato da molti come il migliore attualmente disponibile, perché consente la totale separazione del modello dal controller e dalla sua visualizzazione, proprio come suggerito dal pattern Model-View-Controller [1].

Il code behind

Analizzando in maniera approfondita questo pattern, ciò che ne consegue è che il modello, la visualizzazione ed il controller devono essere *tre entità logiche separate*.

La ragione per cui il code behind nasce è dunque semplice: permette di separare la parte di visualizzazione (la pagina vera e propria) dal model-controller (il code behind e le classi della business logic). Tra gli obiettivi c'è quello di rendere più semplice la modifica di uno degli ambiti di applicazione. Infatti l'interfaccia grafica di solito cambia più spesso della logica e questo modello dovrebbe favorire proprio situazioni simili. Il code behind sfrutta semplicemente una caratteristica intrinseca di ASP.NET. Ogni pagina ASPX di default eredita da Page, se non diversamente indicato. Questa classe si occupa di agganciare alla pagina alcune funzionalità, come la possibilità di utilizzare le classi HttpResponse o HttpRequest, in maniera diretta, senza passare ogni volta per HttpRuntime e HttpContext.

Dunque cambiare la classe da cui eredita la pagina permette di cambiarne le caratteristiche. Il code behind si incastra proprio qui, tanto è vero che non è detto che una pagina debba avere code behind, così come non è detto che ogni pagina ne abbia uno differente. È perfettamente lecito, infatti, avere pagine senza codice o due pagine identiche tra di loro che sfruttino lo stesso "motore".

Daniele Bochicchio *è il content manager di ASPItalia.com, community che si occupa di ASP.NET, Classic ASP e Windows Server System. Si occupa di consulenza e formazione, specie su ASP.NET, e scrive per diverse riviste e siti. È Microsoft .NET MVP, un riconoscimento per il suo impegno a supporto delle community e per l'esperienza maturata negli anni. Il suo blog è all'indirizzo http://blogs.aspitalia.com/daniele/*

Listato 1 I due file generati da VS .NET per supportare il code behind

```
<%@ Page trace=false language="c#" Codebehind="webform1
.aspx.cs" AutoEventWireup="false" Inherits="WebApplicati-
on.webform1"%>

<form runat="server">
<asp:button id="button1" runat="server" />
</form>
```

L'alternativa del code inline

Alcuni pensano che il code behind rappresenti una forzatura del concetto di pagina Web, che è composta da un unico elemento, creato a partire dalle tre entità logiche del pattern MVC. Suddividere la pagina che è atomica in due oggetti legati da una ereditarietà equivale a cambiarne il significato. Attenzione: non stiamo dicendo che separare la presentazione dal codice sia sbagliato, ma che la separazione è possibile anche senza il modello di sviluppo imposto da Visual Studio. L'alternativa si chiama *code inline,* e consente di avere gli stessi vantaggi del code behind:

- separazione totale tra visualizzazione e codice;
- riutilizzo del codice.

In più il code inline ha diversi vantaggi che cercheremo di inquadrare subito. Ovviamente gli ambiti di applicazione sono ben determinati e non è detto che vadano bene in qualsiasi scenario.

Addio al code behind con ASP.NET 2.0

Il fatto che il code behind con ASP.NET 2.0 sparirà, sostituito dal *code beside*, è la conferma che l'attuale modello, nonostante le premesse, è limitante in alcune situazioni. La buona notizia è che Visual Studio 2005 supporterà nativamente sia il code inline che il code beside, che sarà il sostituto del code behind. Col code beside la pagina torna ad essere un unico oggetto, indivisibile, ma formato da due entità logiche, la sua visualizzazione ed il suo codice, tenuti in file separati e compilati a runtime in una sola entità, sfruttando una nuova caratteristica del .NET Framework 2.0 che prende il nome di *partial classes*. In pratica una classe potrà essere definita su più file separati, che sono legati in fase di compilazione in un'unica classe. In questo modo si continua ad avere la separazione dei due livelli, ma viene meno la forzatura di utilizzare due oggetti distinti.

Attenzione al designer di Visual Studio

Visual Studio .NET 2003 è tutto tranne che un buon editor HTML. Non si discute la velocità, ma il risultato prodotto in automatico. Purtroppo troppo spesso si vedono in giro applicazioni Web fatte senza gli opportuni criteri. Il Web ha regole ben precise e standard da anni diffusi ed utilizzati, oltre che un approccio alla progettazione profondamente differente, specie nella GUI, rispetto alle applicazioni per Windows. Il designer Visual Studio .NET rende difficile creare applicazioni Web accessibili, perché spesso l'editor riscrive il codice. Inoltre di default Visual Studio prevede il posizionamento assoluto degli oggetti sulla pagina, cosa che va contro i principi di sviluppo delle applicazioni Web. I limiti che il code inline ha attualmente sono dovuti al fatto che Visual Studio .NET non supporta questa modalità e che l'editor integrato produce codice discutibile. Questo non significa che Visual Studio sia da buttare! Visual Studio è molto comodo, per tutte le feature che offre, nello sviluppo delle classi necessarie alla business logic ed al data layer. La parte Web, che si riduce ad un po' di HTML e qualche Web control, può essere gestita con un editor HTML fatto per questo compito. Nel mio caso utilizzo un semplicissimo editor di testo che si chiama NotepadEx [2] e che è in pratica una versione evoluta del Blocco Note di Windows, con supporto per replace avanzati, indicazioni righe/colonne, ecc.

I vantaggi del code inline

Il vantaggio principale del code inline è dovuto alla rapidità di intervento che l'utilizzo di questa tecnica permette. Se è necessario cambiare qualcosa all'interno di un file, la modifica può essere fatta facilmente, attraverso FTP, e soprattutto è immediata. Utilizzando il code behind, invece, ogni modifica richiede la ricompilazione dell'assembly che contiene le classi della pagina, con conseguente tempo di intervento maggiore perché la modifica si propaghi, e successivo riavvio forzato dell'applicazione. L'effetto collaterale del riavvio dell'applicazione è purtroppo la perdita di tutto ciò che è presente in Cache, Session e Application. Un effetto indesiderato che in alcuni scenari crea non pochi problemi. Se infatti in una Intranet, o comunque in ambienti dove il codice cambia relativamente poco, un riavvio programmato può non essere un problema, in un sito Web pubblico, specie con un traffico medio o alto, rappresenta un problema di non poco conto. La possibilità di garantire un intervento rapido e poco invasivo, alla fine, allunga notevolmente l'uptime e la velocità dell'applicazione. Ad ogni riavvio dell'appdomain, infatti, vengono ricaricati tutti gli assembly, mentre per quanto riguarda una pagina con code inline l'assembly stesso è relativo alla sola pagina e può essere sostituito al volo senza toccare le altre. Ulteriore vantaggio del code inline è una velocità maggiore in fase di startup dell'applicazione. La parte di codice compilata è infatti notevolmente minore, dato che con buone probabilità gli assembly in /bin/ non contengono molto codice, che è invece inserito direttamente nella pagina stessa. E soprattutto, se una pagina non viene richiesta non sarà mai compilata,

con risparmio di risorse. Se questo dovesse essere visto come uno svantaggio, ci sono tecniche che sfruttando un HttpModule sono in grado di compilare, in fase di startup dell'applicazione, tutte le pagine, velocizzandone l'accesso. In realtà, come molti di voi sapranno, la fase di compilazione (che si verifica solo alla prima richiesta) è talmente veloce che l'utente nemmeno se ne accorge. Col code inline ogni pagina può essere scritta in un linguaggio differente, dato che la compilazione avviene in maniera singola, a differenza di VS.NET che compila tutto in una sola DLL e necessita che il linguaggio utilizzato sia sempre lo stesso. Per lo stesso motivo, possiamo combinare user control scritti in linguaggi differenti. Non è proprio uno dei metodi migliori di agire, dato che il linguaggio dovrebbe essere unico per un solo progetto, ma in certi casi può rivelarsi utile (ed aumentare la produttività) specie se stiamo utilizzando funzionalità prese da altri progetti. Ultimo vantaggio è la garanzia che all'interno della pagina c'è tutto quanto serve per farla funzionare.

Listato 2 Una pagina nel modello code–behind

```
using System;
using System.Collections;
using System.Data;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using FoglioMatricolare.Core;

namespace WebApplication
{
  /// <summary>
  /// Pagina contenitore di tutti gli altri quadri.
  /// </summary>
  public class WebForm1 : Page
  {
    protected System.Web.UI.WebControls.Button button1;

    private void Page_Load(object sender,
System.EventArgs e)
    {
      // codice per programmare la pagina
    }

    #region Web Form Designer generated code
    override protected void OnInit(EventArgs e)
    {
      //
      // CODEGEN: This call is required by the ASP.NET
Web Form Designer.
      //
      InitializeComponent();
      base.OnInit(e);
    }

    /// <summary>
    /// Required method for Designer support - do not
modify
    /// the contents of this method with the code edi-
tor.
    /// </summary>
    private void InitializeComponent()
    {
      this.Load += new System.EventHandler(this.Page_
Load);

    }
    #endregion
  }
}
```

È un mito infatti che il solo code behind permetta di separare presentazione da codice, in quanto è lo stesso modello di ASP.NET a garantire questa caratteristica ed il code behind è solo la modalità che chi ha progettato VS.NET ha ideato e messo a disposizione. Ecco la separazione presentazione/logica con il code inline, con la stessa pulizia di codice del code behind:

C#

```
<SCRIPT RUNAT="SERVER" LANGUAGE="C#">
  void Page_Init()
{
  // codice
}
</SCRIPT>

<form runat="server">
<asp:calendar id="cal" runat="server" />
</form>
```

Come si può notare, il codice è racchiuso tra <SCRIPT> e </SCRIPT> ed è distinto dal codice HTML: così come nel code behind, anche in questo caso le zone sono perfettamente delimitate. Visual Studio 2005 sarà in grado di separare le due zone anche se presenti all'interno di un solo file.

Un caso applicato: ASPItalia.com

Mi sono scontrato con queste problematiche diverse volte ed in diversi scenari. Il caso che voglio portare all'attenzione è ASPItalia.com (http://www.aspitalia.com/), sito che gestisco quotidianamente. Il sito ha circa 940 mila page views mensili, con 38 mila unique users al mese. La quantità di pagine presenti (o che sono aggiunte di volta in volta) è considerevole. C'è un lavoro giornaliero di tuning, modifica o semplicemente di correzione del tiro che coinvolge varie parti del sito, dalla business logic al contenuto delle pagine stesse. La business logic è contenuta in assembly specifici e quindi risente dei problemi prima menzionati in caso di aggiunta di funzionalità; ma la modifica ad una pagina, ad esempio quella che mostra le ultime notizie, non impatta sulle altre e non intacca la Cache, di cui viene fatto ovviamente un grande uso, per ottimizzare il carico sul database e rende-

Listato 3 L'unico file necessario ad ASP .NET per funzionare con il code inline

```
<%@ Page language="c#"%>

<form runat="server">
<asp:button id="button1" runat="server" />
</form>

<SCRIPT RUNAT="SERVER">
void Page_Load
{
  // codice per programmare la pagina
}
</SCRIPT>
```

Listato 5 Un esempio di partial class in ASP .NET 2

```
using System;

namespace ASP {

  public partial class Default2_aspx
  {
    void Page_Load()
    {
      // codice per programmare la pagina
    }
  }
}
```

re più veloce la risposta. La rapidità di intervento su un sito Web con un certo traffico, poi, è un requisito indispensabile, perché permette di essere proattivi rispetto a qualsiasi problema, cosa che un intervento programmato non consente. Pensate ad esempio a scenari ancora più grandi, come un sito e-commerce o di notizie molto frequentato, che hanno bisogno di continuo lavoro in ambiente di produzione e pensate ai benefici che un approccio che in pratica non impatta sull'uptime del servizio può garantire.

Gli svantaggi del code inline

Sembrebbero tutte rose e fiori, invece qualche svantaggio, come in tutte le cose, c'è. La mancanza più grande del code inline è ovviamente il supporto di un editor che attualmente sia in grado di permetterne un pieno sfruttamento, mancanza che sarà sopperita con Visual Studio 2005. Potrebbe essere uno svantaggio il fatto che il codice sia in chiaro, rispetto ad un assembly, ma chiunque abbia preso in mano un reflector sa benissimo che non c'è differenza alcuna: il tempo necessario ad avere, da un assembly, il codice C# corrispondente è pari al tempo necessario a sorseggiare un caffè al bar. Il vantaggio più grande del code behind, invece, è la possibilità di creare una DLL che contenga tutto, in maniera automatica o sfruttando un tool come il Precompiler [3] per inglobare tutto, codice HTML incluso. Sicuramente diventa più semplice da ridistribuire, ma è un approccio che, come detto, non è applicabile in caso di siti che cambiano frequentemente. Di sicuro, proprio perché VS.NET non lo supporta nativamente, non c'è una "compilazione" dei file (va quindi fatto un controllo manuale, o attraverso tool automatizzati come Microsoft Application Center [4]). Per quanto riguarda il debug, invece, basta utilizzare la funzionalità di attach del processo per avere le stesse funzionalità. Per quanto riguarda la fase di unit testing, in genere questi vengono fatti, sfruttando sistemi come NUnit [5], sulle classi del business layer piuttosto che sul codice della pagina, dunque la scelta non toglie né aggiunge nulla.

Listato 4 Il code beside di ASP.NET 2.0 in azione: si noti la maggiore puliza della classe utilizzata, dovuta alle partial classes

```
<%@ page language="C#" compilewith="~/default2.aspx.cs" %>

<form runat="server"
<asp:button id="button1" runat="server" />
</form>
```

Conclusioni

Code behind o code inline, alla fine, è la solita scelta che in informatica accende come sempre appassionanti diatribe tra le diversi fazioni. Se l'ambito è un'applicazione monolitica che non subisce cambiamenti nel tempo (come nel caso di una Intranet o per pacchetti) allora il code behind ha il vantaggio di essere supportato da Visual Studio .NET. Se invece un'applicazione ha bisogno di interventi giornalieri, come siti web dall'elevato traffico o soggetti a frequenti cambiamenti o fine-tuning, allora il code inline può essere una valida alternativa. Alla fine è solo un approccio diverso allo sviluppo, influenzato dallo scenario in cui si andrà ad utilizzarlo e dai gusti personali.

Riferimenti

[1] Model-View-Controller – MSDN Patterns & Practices
http://msdn.microsoft.com/library/en-us/dnpatterns/html/DesMVC.asp
[2] NotepadEx
http://notepadex.cjb.net/
[3] ASP.NET 1.x Precompiler – Cristian Civera
http://precompiler.aspitalia.com/
[4] Microsoft Application Center - Microsoft
http://www.microsoft.com/applicationcenter/
[5] NUnit
http://www.nunit.org/

Introduzione al multithreading

La programmazione di thread multipli permette di risolvere molti problemi concreti, soprattutto durante la realizzazione di applicazioni server

di Lorenzo Vandoni

La programmazione multithreading era considerata alcuni anni or sono come un ottimo meccanismo per sfruttare al meglio la potenza degli elaboratori. Più recentemente si è affermata una corrente di pensiero secondo la quale il gioco, tutto sommato, non valga la candela. In particolare, si ritiene che la complessità aggiuntiva legata alla gestione della sincronizzazione tra i thread pesi maggiormente, in termini di costi di sviluppo e di manutenzione, rispetto ai vantaggi che si potrebbero ottenere. Molto spesso, gli stessi vantaggi si possono ottenere più facilmente e a costi più contenuti, semplicemente aumentando la potenza dell'hardware disponibile. Al di là di queste considerazioni, però, continuano ad esistere molti casi in cui la programmazione multithreading si rivela necessaria. Tra questi, quelli più frequenti riguardano lo sviluppo di funzionalità che debbano essere terminate in background senza interrompere la possibilità di utilizzare il sistema da parte dell'utente, e la realizzazione di applicazioni server, che debbano rispondere contemporaneamente alle richieste di client multipli. In questo articolo verrà mostrato come realizzare un sistema multithreaded utilizzando Visual Basic.NET. Nel **Riquadro 1** viene invece descritto il supporto offerto da Visual Basic 6 per realizzare un sistema di questo tipo.

Cos'è la programmazione multithreading

Cominciamo con un breve cenno introduttivo, per ricordare le caratteristiche principali di un sistema multithreaded, senza la pretesa di dare una descrizione esaustiva.

Un sistema si dice *multitasking* se è in grado di eseguire più processi contemporaneamente. Un'applicazione si dice *multithreaded* se è in grado di sfruttare le caratteristiche di un sistema multitasking per fare eseguire contemporaneamente diverse sezioni di codice, dette *thread*. Il thread è l'unità di base a cui viene assegnato uno slot di tempo del processore. Naturalmente, in un sistema multiprocessore i vari thread possono essere contemporaneamente in esecuzione su processori diversi, ottenendo in questo caso un parallelismo reale e non simulato. Il principale problema, nella programmazione di un'applicazione multithreaded, è la condivisione delle risorse, come variabili globali o file. Poiché più parti dell'applicazione sono in esecuzione allo stesso tempo, è possibile che la stessa variabile sia utilizzata contemporaneamente da thread differenti, con possibili conseguenze negative. Esistono algoritmi consolidati, come semafori e monitor, che permettono di garantire l'accesso condiviso alle risorse mantenendole in uno stato consistente. Questi algoritmi sono variamente implementati nelle librerie di sistema disponibili per vari linguaggi di programmazione oggi più in voga. Nei paragrafi seguenti, vedremo come sono implementati nella Base Class Library (BCL) di .NET.

Programmazione multithreading con .NET

Nella BCL ogni thread viene rappresentato come un'istanza della classe *System.Threading.Thread*. Ogni applicazione ha almeno un thread, ovvero il thread principale dell'ap-

Lorenzo Vandoni *è laureato in Informatica, ed è uno specialista di progettazione e sviluppo con tecniche e linguaggi object–oriented. Ha collaborato alla realizzazione di framework, strumenti di sviluppo e software commerciali in C++, Java e Visual Basic. Può essere contattato tramite e–mail all'indirizzovandoni@infomedia.it.*

plicazione, e il thread in esecuzione può essere in ogni istante recuperato utilizzando la variabile membro statica *CurrentThread* della classe *Thread*. Per creare un nuovo thread, si può procedere in questo modo:

- anzitutto, occorre definire una classe, che possiamo chiamare *myClass*, includendo il codice che dovrà essere eseguito nel thread separato all'interno di un metodo *myMethod*, e istanziare la classe creando un oggetto *myObject*;
- un riferimento al metodo *myMethod* dovrà poi essere passato a un oggetto *delegate* di tipo *ThreadStart*, in questo modo:

```
Dim myStart As New ThreadStart (AddressOf myObject.myMethod)
```

- infine, occorre creare un'istanza della classe *Thread*, passandogli un riferimento al *delegate* appena creato, e lanciare in esecuzione lo stesso con il metodo *Start*, nel modo seguente:

```
Dim myThread As New Thread(myStart)
MyThread.Start()
```

Un oggetto *delegate*, nei linguaggi .NET, rappresenta un generico riferimento ad una funzione. Come i programmatori C++ ben sanno, i riferimenti a funzione possono essere utilizzati per vari utili scopi, come la definizione di funzioni callback o di algoritmi parametrici, come ad esempio una generica funzione di ordinamento che prenda come parametro una funzione di confronto. In questo caso, il riferimento viene utilizzato per istanziare la classe *Thread*, informandola di quale funzione dovrà essere eseguita nel thread separato. In particolare, nella BCL di .NET, ogni *delegate* è istanza di una specifica classe. È possibile dichiarare nuove classi *delegate* in questo modo …

```
Public Delegate Sub myDelegateClass(...)
```

… e quindi istanziarle scrivendo:

```
Dim myDelegateFunc As MyDelegateClass( AddressOf
myObject.myMethod )
```

Perché l'assegnamento sia corretto, è necessario che la *signature* della funzione *myMethod* corrisponda a quella indicata nella definizione di *myDelegateClass*. Ogni classe *delegate*, cioè, specifica il formato delle funzioni che potranno essere passate come riferimenti alle sue istanze. La classe *ThreadStart* è una classe predefinita all'interno della BCL, e serve appunto ad identificare un oggetto *delegate* utilizzato per incapsulare una funzione che debba essere eseguita in un thread separato. La classe è definita in questo modo:

```
Public Delegate Sub ThreadStart()
```

Come si può notare, la funzione non accetta nessun parametro. Ho notato che questo particolare, a volte, mette in difficoltà i programmatori, perché sembra indicare che le funzioni eseguite in thread separati non possano accettare parametri in input. Non bisogna dimenticare, però, che si tratta sempre di metodi eseguiti da oggetti, che possono tranquillamente ricevere parametri in ingresso tramite il costruttore o altre funzioni pubbliche. Si potrà cioè scrivere:

```
Dim myObject as New myClass(myVar1, myVar2)
Dim myStart As New ThreadStart (AddressOf myObject.myMethod)
Dim myThread As New Thread(myStart)
MyThread.Start()
```

Una volta in esecuzione, il metodo *myMethod* potrà tranquillamente accedere a tutte le variabili dichiarate all'interno di *myClass*, incluse quelle al cui interno sono stati salvati i valori dei parametri *myVar1* e *myVar2*. La gestione di eventuali valori di ritorno del thread può essere effettuata in modo simile. Il thread, cioè, potrebbe semplicemente salvare i valori di ritorno all'interno di variabili accessibili anche da altri punti dell'applicazione. Come già precedentemente accennato, però, la possibilità di condividere variabili fra thread distinti, oltre a costituire una opportunità, può anche creare qualche problema.

Riquadro 1 Visual Basic 6 e il multithreading

Visual Basic consente di creare applicazioni multithreaded solo in un caso particolare, e con delle limitazioni ben precise. Il caso è quello degli *Exe ActiveX*, ovvero applicazioni che, come Word ed Excel, oltre a poter essere utilizzate in modo autonomo, offrono agli sviluppatori un modello ad oggetti COM per accedere alle loro funzionalità. Un'applicazione di questo tipo deve poter servire contemporaneamente le richieste provenienti da più fonti diverse, e quindi deve necessariamente essere multithreaded. Nel caso in cui si decida di creare un *Exe ActiveX*, Visual Basic 6 mette a disposizione diversi tipi di opzioni che possono essere impostate all'interno della finestra delle proprietà del progetto. L'opzione *one thread per object*, ad esempio, significa che ogni oggetto creato da altre applicazioni vivrà in un thread separato, mentre l'opzione *thread pool* significa che verrà fissato un numero massimo di thread, e che ogni oggetto vivrà all'interno di uno di questi.

Il modello di threading adottato, denominato *Apartment Model*, permette di garantire che ogni thread utilizzi una copia separata di tutte le variabili ad esso accessibili. L'unico modo per far comunicare due oggetti in thread separati è far sì che l'applicazione che li usa passi ad uno il riferimento all'altro. La comunicazione si chiama *cross thread marshaling* e risulta piuttosto lenta. Il supporto offerto da Visual Basic 6 alla programmazione multithreading è quindi molto limitato, ed è stato introdotto solamente per la necessità di supportare lo sviluppo di *Exe ActiveX*.

Gestione dei thread

Il thread precedentemente creato può essere successivamente gestito utilizzando i vari metodi messi a disposizione dalla classe *Thread*. Tra questi vale la pena di citare:

- le variabili *Thread.ThreadState* e *Thread.IsAlive*, che permettono di determinare lo stato del thread;
- il metodo *Thread.Sleep(n)*, che permette di mettere in pausa un thread per *n* millisecondi; questo metodo, tra l'altro, può essere utilizzato anche nel caso di applicazioni non multithreaded, per interrompere temporaneamente l'esecuzione del thread principale;
- i metodi *Thread.Suspend* e *Thread.Abort*, che permettono rispettivamente di mettere un thread in pausa e di bloccarlo;
- il metodo *Thread.Resume*, che consente di ripristinare un thread precedentemente sospeso;
- la variabile *Thread.Priority*, che permette di determinare la priorità assegnata al thread;
- il metodo *Thread.Join*, infine, che può far si che un thread attenda il termine dell'esecuzione di un altro thread prima di procedere.

Il metodo *Thread.Join* costituisce uno dei meccanismi messi a disposizione da parte della BCL per effettuare la sincronizzazione tra i thread. Vediamo di che si tratta.

Sincronizzare i thread

Ho già accennato al fatto che il principale problema, nella programmazione mulithreading, sia la condivisione delle risorse. Per affrontare il problema, la BCL mette a disposizione vari tipi di meccanismi di sincronizzazione. In particolare:

- l'istruzione *SyncLock* permette di impostare un blocco su un oggetto, e fare in modo che nessun altro thread possa accedervi fino a che il blocco non viene esplicitamente rilasciato. Può essere utilizzata in questo modo:

```
SyncLock myObject
  MyObject.myMethod()
  'altre istruzioni
End SyncLock
```

Le istruzioni incluse all'interno del blocco compreso tra *SyncLock* e *End SyncLock* verranno eseguite solo se il thread riesce ad ottenere un accesso esclusivo all'oggetto *myObject*. Il tentativo di accesso esclusivo è rappresentato dall'istruzione *SyncLock myObject*.

Qualora un altro thread abbia precedentemente ottenuto un blocco sullo stesso oggetto, l'esecuzione del thread verrò bloccata sulla riga *SyncLock myObject*, fino a che l'altro thread non avrà terminato l'esecuzione del blocco

- il metodo *Thread.Join*, già precedentemente citato, permette di bloccare esplicitamente l'esecuzione del thread chiamante, fino a che il thread chiamato non ha terminato la propria esecuzione. In pratica, eseguendo, dall'interno di un metodo *myMethod*, la seguente istruzione:

```
myThread.Join()
```

si otterrà un blocco sull'esecuzione di *myMethod*, che verrà rilasciato solamente nel momento in cui il thread *myThread* avrà terminato la propria esecuzione.

La funzione *Join* accetta anche, come parametro opzionale, un intervallo di tempo alla scadenza del quale l'esecuzione del metodo potrà comunque essere ripresa, indipendentemente dalla effettiva terminazione di *myThread*

- la classe *System.Threading.Monitor* fornisce un insieme di metodi statici (o *Shared*, per Visual Basic) che permettono di impostare dei blocchi su oggetti. Di fatto, l'istruzione *SyncLock* viene implementata sfruttando i servizi offerti da questa classe, e ne costituisce un caso particolare di utilizzo. I metodi *TryEnter* ed *Enter* permettono di acquisire un blocco, mentre il metodo *Exit* consente di rilasciarlo. I metodi *Wait*, *Pulse* e *PulseAll* consentono una sorta di comunicazione tra i thread interessati all'utilizzo di uno stesso oggetto.

La classe *Monitor* costituisce in un certo senso un'implementazione impropria del classico concetto di monitor della programmazione multithreading, dove con questo termine viene denotato un oggetto che garantisca mutua esclusione nell'esecuzione dei propri metodi, ma all'atto pratico garantisca lo stesso tipo di funzionalità.

Conclusioni

I concetti che stanno alla base della programmazione multithreading sono tutto sommato semplici, e grazie al supporto offerto dal namespace *System.Threading*, l'introduzione di questo tipo di funzionalità all'interno di un'applicazione non presenta particolari problemi.

La necessità di sincronizzare l'accesso alle risorse condivise, però, obbliga lo sviluppatore ad adottare uno stile di lavoro ben preciso, in cui ogni thread deve chiedere il permesso di accedere a tali risorse. Un errore di programmazione in questa fase può portare a comportamenti del tutto imprevedibili o a blocchi critici, ovvero situazioni in cui un thread viene bloccato all'infinito in attesa dell'accesso a una risorsa che non verrà mai liberata. La programmazione multithreading va quindi utilizzata quando si rende realmente necessaria, come nei casi citati all'inizio.

Programmazione delle Smart Card

Standard, specifiche e piattaforme di sviluppo per Java, C e Visual Basic

Internet non è in grado di proporre un mezzo di identificazione certificata degli interlocutori né un livello adeguato di protezione delle trasmissioni: le tecnologie basate su smart card permettono di sviluppare applicazioni sicure per i settori delle telecomunicazioni mobili, PayTV, commercio elettronico, sistemi bancari e di accesso sicuro ai sistemi informativi.

Tra gli argomenti affrontati nel testo:

lo standard ISO 7816 - la piattaforma Javacard - Architettura PC/SC, le specifiche PKCS#11 - il Framework OpenCard - utilizzo di smart card con Windows, Outlook e Internet Explorer - Crittografia.

Viene inoltre descritto un emulatore di smart card e di dispositivo di lettura utilizzabile per esercitarsi con le istruzioni APDU ISO 7816.

€ 18.00 - pagg. 160
ISBN 8881500140

GRUPPO EDITORIALE
INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

WEB: http://www.infomedia.it - e-mail: book@infomedia.it - tel: 0587/736460

Implementare l'UNDO in un'applicazione Windows Form

In diversi tipi di programmi la presenza di una funzione di "undo" (annulla) risulta fondamentale; vediamo come implementarla in un'applicazione Windows Forms

di Lorenzo Vandoni

Una funzione di "undo" è un meccanismo che permette all'utente di annullare l'ultima operazione effettuata, qualunque essa sia. Tutti i programmi di uso professionale che permettono di eseguire una serie di operazioni (Word, Excel, Photoshop, Autocad sono solo alcuni nomi) possiedono questa possibilità.

La funzione di undo deve essere progettata in modo da permettere all'utente di annullare più operazioni, in ordine inverso rispetto a come sono state eseguite. Questa caratteristica è ormai diffusa e credo che ormai solo Notepad preveda una funzione di undo ad un solo livello. In questo articolo mostreremo come implementare una funzione di undo multiplo in un'applicazione Windows Forms.

Un'infrastruttura riutilizzabile

Il codice presentato è costituito da un piccolo framework, che costituisce l'infrastruttura di un sistema di undo del tutto generico. È fornita anche un'applicazione che lo utilizza. L'applicazione demo è molto semplice, essendo costituita da un'unica finestra con un immagine al suo interno che può essere spostata dall'utente in una nuova posizione via drag-and-drop. Lo spostamento dell'immagine costituisce l'azione che può essere annullata. Il codice relativo alla creazione della finestra e alla gestione dello spostamento dell'immagine è semplice e utilizza in modo appropriato i vari eventi di gestione del mouse. Per i dettagli rimando al progetto allegato, nel quale ho in ogni caso inserito alcuni commenti nei punti più significativi.

Il termine "framework" è probabilmente un po' eccessivo, ma credo sia quello che meglio identifichi questo gruppo di classi ed interfacce, il cui scopo è quello di fornire delle generiche funzionalità di undo, che poi dovranno essere completate in ogni specifica applicazione, implementando una classe derivata. Vediamo quali sono gli ingredienti fondamentali:

- serve un meccanismo che permetta al sistema di mantenere traccia delle azioni effettuate dall'utente, per poterle annullare su richiesta. L'ordine di annullamento è inverso rispetto all'ordine di esecuzione, cioè l'ultima azione effettuata dovrà essere annullata per prima. Il meccanismo richiesto è quindi molto simile a una struttura LIFO (Last In First Out), cioè a uno stack;
- è importante non imporre nessun vincolo rispetto ai possibili tipi di azioni che dovranno poter essere annullate, in modo che il framework possa essere riutilizzato in più ap-

Lorenzo Vandoni *è laureato in Informatica, ed è uno specialista di progettazione e sviluppo con tecniche e linguaggi object–oriented. Ha collaborato alla realizzazione di framework, strumenti di sviluppo e software commerciali in C++, Java e Visual Basic. Può essere contattato tramite e–mail all'indirizzovandoni@infomedia.it.*

plicazioni. D'altra parte, è necessario che tutte queste azioni siano implementate tramite moduli software simili tra loro, in modo che si possa scrivere una generica implementazione del meccanismo di undo, che selezioni l'azione da annullare, ed esegua l'annullamento scrivendo *oAction.Undo()*, senza preoccuparsi di quale azione effettivamente si tratti. Tutto questo si tradurrà nella necessità di definire un'interfaccia *IUndoableAction*, che dovrà essere implementata in modo diverso per ogni diverso tipo di azione;

- occorre infine prevedere la possibilità di impostare ed annullare azioni complesse, cioè composte da più azioni elementari. Supponiamo per esempio di implementare una piccola estensione all'applicazione di esempio, in cui sia possibile selezionare più di un'immagine e spostare con un'unica operazione di drag-and-drop tutte le immagini selezionate. In questo caso, una singola operazione di undo dovrà essere in grado di ripristinare contemporaneamente la posizione di tutte le immagini. Questo requisito può essere soddisfatto introducendo nel nostro sistema il concetto di transazione, permettendo cioè di raggruppare una serie di operazioni elementari tramite due comandi *begin* e *commit*.

Partendo da queste considerazioni, passiamo ora a definire l'architettura object-oriented del framework.

Progettazione del framework

Il primo requisito da soddisfare è quello di creare un generico sistema in grado di mantenere traccia delle operazioni effettuate, in modo simile ad uno stack. Questo requisito può essere soddisfatto tramite la creazione di una classe che fornisca le classiche operazioni *Push* e *Pop*, nonché alcune operazioni aggiuntive, come ad esempio *Begin* e *Commit*, necessarie per implementare il concetto di transazione. Possiamo chiamare questa classe *CompoundActionStack*, per evidenziare sia il tipo di funzionamento sia la capacità di gestire azioni complesse. Dovendo integrare la classe nella .NET FCL (Framework Class Library), potremmo evitare di implementare le funzionalità di gestione dello stack, grazie alla disponibilità della classe *System.Collections.Stack*. Come mettere in relazione però le due classi? La classe *CompoundActionStack* "è uno" stack, quindi la relazione di ereditarietà sembrerebbe la più corretta. D'altra parte, la classe *Stack* fornisce una serie di metodi non necessari ai nostri scopi, che anzi se utilizzati ne comprometterebbero il corretto funzionamento (ad esempio Clear e Pop manipolano lo stack senza annullare effettivamente l'azione). L'ideale sarebbe poter ereditare il funzionamento della classe *Stack*, senza però esporne tutti i metodi. Questa è l'*ereditarietà privata*, ma sfortunatamente è disponibile solo in C++. Dovremo quindi ricorrere ad un compromesso, cioè quello di utilizzare una relazione di aggregazione e definire nella nostra classe dei metodi *Push* e *Pop* che sfrutteranno gli omonimi metodi della classe *Stack*. Ognuno degli elementi contenuti all'interno del *CompoundActionStack* è un'azione complessa. Una classe *CompoundAction* servirà a mantenere una lista ordinata di azioni elementari. Anche in questo caso potremo sfruttare la disponibilità all'interno della BCL di una classe come *ArrayList*, che fornirà tutte le funzionalità richieste. La creazione di una nuova classe, rispetto al semplice utilizzo di *ArrayList*, si rende necessaria sia per dotare la classe *CompoundAction* di un'interfaccia più semplice ed adatta allo scopo sia per introdurre il controllo sul tipo degli oggetti mantenuti nella lista. Infatti, tutte le classi del namespace *System.Collections* sono definite come collezioni di *Object* e possono quindi contenere al loro interno oggetti di qualsiasi tipo. Nel nostro framework, invece, è necessario creare una lista che possa contenere solo azioni elementari. Anche in questo caso ecco una limitazione all'interno dell'ambiente di programmazione utilizzato, in quanto la disponibilità della programmazione generica (che verrà introdotta nella versione 2.0 di .NET) ci avrebbe permesso di creare una versione specifica della classe *ArrayList* che ammettesse solo elementi di un determinato tipo. La definizione di ognuna delle azioni elementari costituisce il punto di contatto tra il framework e l'applicazione che dovrà utilizzarlo. La definizione di cosa sia effettivamente una singola azione elementare che possa essere annullata dovrà essere infatti demandata all'applicazione, mentre il framework si limiterà a definire un'interfaccia che ogni classe che rappresenti un'azione elementare dovrà implementare. Chiameremo questa interfaccia *IUndoableAction*. La **Figura 1** contiene il diagramma delle classi descritte.

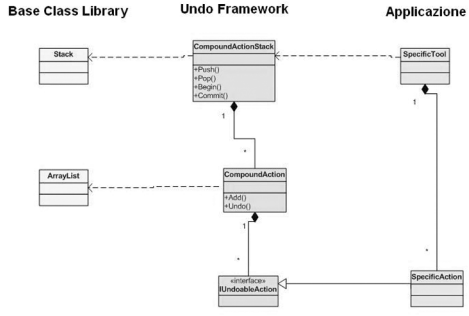


Figura 1 Il diagramma delle classi del nostro piccolo framework

Implementazione in VB.NET

L'implementazione del framework in linguaggio VB.NET, allegata all'articolo e contenuta nel file *UndoFramework.vb*, segue fedelmente l'architettura tratteggiata nel paragrafo precedente. Vale in ogni caso la pena di sottolineare alcuni dettagli:

- i metodi della classe *CompoundActionStack*, per gli scopi della nostra applicazione di esempio, potrebbero essere definiti come statici, o *shared* secondo la sintassi VB.NET. Volendo però permettere l'utilizzo del framework anche all'interno di un'applicazione più complessa, composta da più editor che potrebbero essere aperti contemporaneamente in finestre differenti, è preferibile la scelta di utilizzare metodi e variabili di istanza;
- la variabile booleana *bAtomicTransaction* nel metodo *Push* nella classe *CompoundActionStack* viene utilizzata per gestire transazioni implicite composte da una sola azione elementare. L'obiettivo è anche quello di evitare errori nel caso in cui il metodo *Push* venga invocato senza prima avere eseguito il metodo *Begin*;
- il metodo *Undo* della classe *CompoundAction* esegue l'annullamento delle singole azioni elementari nell'ordine in cui queste sono state inserite. Questo perché si suppone che un'azione composta sia costituita da un insieme di azioni elementari il cui ordine di esecuzione sia ininfluente. Nel caso in cui tale ordine sia invece rilevante, le azioni elementari dovranno essere suddivise a livello applicativo (cioè nel momento in cui il framework viene utilizzato) in più azioni composte.

La classe DragAction

Per potere utilizzare il nostro framework è anzitutto necessario definire una classe che implementi l'interfaccia *IUndoableAction*. All'interno di questa classe, che chiameremo *DragAction*, occorre predisporre delle variabili di istanza che permettano di registrare lo stato attuale del sistema, in modo da poterlo ripristinare.

Ci occorreranno, in particolare, un riferimento al control che dovrà essere ripristinato e le coordinate originali dello stesso. Il riferimento potrà essere passato come argomento nel costruttore della nostra classe, mentre le coordinate potranno essere automaticamente dedotte da tale argomento (notare che questo implica che l'istanza di *DragAction* dovrà essere creata prima di effettuare lo spostamento del control). La classe, infine, dovrà implementare il metodo *Undo* dichiarato nell'interfaccia e ripristinare, al suo interno, la situazione corrispondente al momento in cui l'istanza è stata creata. Il codice corrispondente alla classe *DragAction* è mostrato nel **Listato 1**.

Utilizzare il framework nelle applicazioni

Per utilizzare il framework è necessario procedere nel modo seguente:

1. dichiarare e istanziare un'istanza della classe *CompoundActionStack*;
2. registrare ognuna delle azioni effettuate dall'utente in modo che possano essere annullate; questa registrazione va impostata nel momento in cui il mouse viene rilasciato, scrivendo:

Listato 1 Implementazione della classe DragAction

```
'questa classe mantiene una lista ordinata di azioni
elementari
Public Class DragAction
    Implements IUndoableAction
    'variabili che mantengono lo stato da ripristinare
    Dim mX As Integer
    Dim mY As Integer
    Dim moCtrl As Control

    'il costruttore accetta un control
    Public Sub New(ByVal oCtrl As Control)
        moCtrl = oCtrl
        mX = moCtrl.Left
        mY = moCtrl.Top
    End Sub

    'ripristina lo stato precedente
    Public Sub Undo() Implements IUndoableAction.Undo
        moCtrl.Left = mX
        moCtrl.Top = mY
    End Sub
End Class
```

```
moCompoundActionStack.Push(New DragAction(PictureBox1))
```

3. permettere all'utente di effettuare l'undo scrivendo

```
moCompoundActionStack.Pop.Undo
```

Nel codice di esempio questa azione è associata alla classica combinazione di tasti *Ctrl+Z*. Come si può notare, l'esempio non fa uso di azioni complesse e utilizza la possibilità di eseguire transazioni implicite.

Conclusioni

Il piccolo framework proposto in questo articolo è ovviamente elementare e migliorabile.

Non sono state incluse operazioni semplici come *Clear*, per svuotare lo stack, e non è stato affrontato l'aspetto del consumo di memoria.

L'utilizzo di un framework di questo tipo, infatti, implica la creazione di un gran numero di oggetti, in corrispondenza di ognuna delle azioni effettuate dall'utente. Una tecnica per limitare l'uso della memoria consiste nell'impostazione di un limite massimo al numero di azioni che possono essere registrate nello stack, lascio questo aspetto come esercizio per chi volesse provare ad estendere le classi qui proposte.

Durante la trattazione sono state anche sottolineate alcune limitazioni del framework .NET, come la mancanza di ereditarietà privata e della programmazione generica, che obbligano, in alcuni casi, ad adottare soluzioni di ripiego.

I QUADERNI

I QUADERNI

pratici economici e di approfondimento

ABC della Programmazione

Troppe sigle incomprensibili?
Scopri il loro significato!
In un unico volume tutte le sigle spiegate da Programmo Subito

Il mondo dell'informatica è un fertile terreno in cui la "contaminazione" delle sigle e degli acronimi attecchisce in maniera vigorosa ed esuberante.
Se da un lato và riconosciuto che si tratta di un fenomeno inevitabile, visto anche il continuo fiorire di nuove tecnologie, dall'altro è frequente ritrovarsi disorientati di fronte a contesti (libri, lezioni, presentazioni, interviste, ...) in cui l'uso di sigle è estremizzato e, quindi, portato oltre il limite della sopportazione e della comprensione. Direi quindi che, una volta accantonato il desiderio di venire a capo sempre e comunque di una qualsivoglia abbreviazione, è indispensabile inserire nel proprio bagaglio di conoscenze, quel sottinsieme minimo e "sempreverde" di termini che sono ormai entrati a pieno titolo nel gergo informatico. ABC della programmazione nasce proprio con lo scopo d'individuare lo stretto indispensabile, col desiderio di "dare il La": siamo convinti che, grazie ad esso, ognuno di voi troverà più facile muovere i primi passi nel proprio cammino di formazione.

I quaderni di Programmo Subito
ABC della Programmazione
pagg. 30 - ISBN 8881500051
€ 5.16

ABC di Java

Entra nel mondo della Programmazione ad oggetti con un linguaggio completo, flessibile e multipiattaforma

Java ha sicuramente suscitato un grande fermento nella comunità di sviluppatori al momento della sua comparsa. il concetto "rivoluzionario" che stava alla base della sua implementazione era quello di realizzare un "sogno" comune a molti programmatori: poter creare un'applicazione che girasse su più sistemi operativi senza dover ogni volta modificare il codice, ricompilare, testare, ecc. Scoprirete nei dettagli come muovere i primi passi, come creare la vostra prima applicazione e aumentarne progressivamente le potenzialità; ma, soprattutto, sarete introdotti al moderno approccio alla programmazione, quello basato sul paradigma della Object Oriented Programming o OOP.
Nessun argomento fondamentale è stato tralasciato: apprenderete come gestire i file, come interrogare un database, come realizzare un'applicazione di networking, come gestire gli errori e come sfruttarli a vostro vantaggio.

I quaderni di Programmo Subito
ABC di Java
pagg. 102 - ISBN 8881500094
€ 7.75

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

WEB: http://book.infomedia.it - e-mail: book@infomedia.it - Tel: 0587/736460

Alla scoperta di Visual Basic e VB .NET

L'obiettivo del libro è di permettere a chiunque di iniziare a sviluppare i primi programmi in Visual Basic. La trattazione prevede un linguaggio molto semplice ed è supportata da esempi ed esercizi, che hanno lo scopo di dare un riscontro pratico agli argomenti esposti. Il lettore è guidato in modo graduale durante l'apprendimento; inizia dapprima ad acquisire confidenza con i concetti di base della programmazione e prosegue imparando a sviluppare delle applicazioni sempre più complesse, aumentando la propria conoscenza fino a raggiungere la capacità di gestire, fra l'altro, dei contenuti multimediali e degli archivi organizzati in basi di dati.
Gli ultimi capitoli sono incentrati sulla nuova versione del linguaggio, Visual Basic .NET, puntando l'attenzione in particolar modo sulle diversità che caratterizzano questa versione rispetto alla precedente. Sono mostrati, attraverso una serie di esempi pratici, i punti chiave del nuovo linguaggio oltre a come implementare una semplice soluzione basata su Visual Basic .NET

Alla scoperta di Visual Basic e VB .NET
Maurizio Crespi - Andrea Frosini
pagg. 385
ISBN 8881500108
€ 34.00

dal SOMMARIO

Controllo If e varianti
Controllo Select Case
La struttura For
Strutture regolate da una condizione booleana
La gestione dei file di testo
I file ad accesso casuale
Applicazioni in grado di far uso dei databse
Oggetto Recordset: ricerche all'interno degli archivi Interrogare i database
Produzione di stampe di elevata qualità
La visualizzazione di immagini grafiche
Disegnare immagini con Picture
La creazione dei menu
Gestire i menu a run time
La gestione degli errori
Sviluppare codice migliore
Applicazioni SDI e MDI
I file help di Windows
La programmazione ad oggetti
VB .NET
VB .NET e le WebForm
ADO .NET e "Data Components"
DataSet, DataTable e ReportViewer

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

technical BOOK

Clicca su www.infomedia.it per leggere il capitolo d'esempio

WEB: http://book.infomedia.it - e-mail: book@infomedia.it - Tel: 0587/736460

Utilizzare Access con Visual Source Safe

L'introduzione di Visual Source Safe nello sviluppo con Microsoft Access porta ad una benefica rivoluzione nel modo di lavorare, sia dello sviluppatore solitario sia in un gruppo di programmatori.

di Gionata Aladino Canova

Gli sviluppatori Microsoft si dividono essenzialmente in due grandi famiglie: chi sviluppa con Microsoft Access e chi sviluppa con Microsoft Visual Studio. I primi sono in un limbo per cui non sono considerati utenti (provate a chiedere, ad un utente esperto di Office, la differenza tra CurrentProject.Connection e CodeProject.Connection!) ma non sono neanche considerati sviluppatori a pieno titolo. Quindi accade che soluzioni adatte ad entrambe le famiglie come Visual Source Safe, siano ben documentate per chi le utilizza con Visual Studio, ma lo siano pochissimo per chi le utilizza con Microsoft Access. Il problema è amplificato se si cerca documentazione in lingua italiana. Nei precedenti numeri (VBJ 42-43-44) sono comparsi tre ottimi articoli su Visual Source Safe, che illustrano le caratteristiche dei software di Version Control System e danno un'ampia panoramica su Visual Source Safe. Riagganciandomi ad essi, tenterò di colmare le lacune che si trovano provando ad utilizzare questo strumento con Microsoft Access. Considereremo il caso tipico di un'azienda con più sviluppatori; quello che però diremo è valido in massima parte anche nel caso di un singolo sviluppatore.

L'installazione di Visual Source Safe

Visual Source Safe è uno strumento che richiede poche risorse di sistema. Si consideri il caso tipico di un server dove memorizzare le varie versioni dei programmi e dei client sui quali esse vengono sviluppate. L'installazione inizia dal server.

Inserito il CD-ROM, si selezioni *Installazione standard*. Con pochi passi, la procedura di installazione sarà completata.

A questo punto si dovrà condividere manualmente una cartella che, di default, è quella in cui è stato installato Visual Source Safe. (Nota personale: poiché mi dà una certa orticaria condividere una cartella che sta dentro la cartella Programmi, ho preferito creare e condividere una cartella chiamata VSS sotto la root del disco del server.)

Perché tutto funzioni, chi deve utilizzare il sistema deve avere i diritti di scrittura sulla cartella. Le istruzioni per effettuare le impostazioni complete si trovano sull'articolo [1] in MSDN. Si avvii ora l'interfaccia di amministrazione di Visual Source Safe e si selezioni la cartella appena condivisa. Visual Source Safe provvederà a crearvi un nuovo database.

Per i client, l'installazione può essere fatta in due modi diversi: si può utilizzare il NETSETUP.EXE, presente nella cartella di installazione di Visual Source Safe, oppure si può utilizzare il CD-ROM e installare la parte client di Visual Source Safe. Se si opta per il NETSETUP.EXE, c'è un piccolo vantaggio: se sul server sono state applicate le patch a Visual Source Safe, l'installazione risulterà già aggiornata (al momento della stesura dell'articolo, la versione è la 6.0d).

A questo punto, sul client, si ha a disposizione l'interfaccia normale di Visual Source Safe. Access ha del-

Gionata Aladino Canova *programma dai bei tempi del Sinclair Spectrum. Laureato in Informatica, è titolare della Aladino Informatica e socio di TDE Informatica srl. Sviluppa con Microsoft Access, VB.NET e realizza siti in ASP/ASP.NET.*
Può essere contattato a info@aladinoinformatica.com

le proprie estensioni (add-in) per utilizzare Visual Source Safe, che vanno installate separatamente. Se si ha Office XP Developer, inserendo il CD di Office XP Developer, si selezioni *Strumenti per la produttività di Access* (**Figura 1**). Chi usa Access 2003, come prerequisito, deve avere il SP1 installato, come si legge in [2]. Dalle prove che ho fatto, esistono poi due alternative. Se dispone anche di Access 2002 ed ha installato gli *Strumenti per la produttività di Access*, è a posto. Infatti, l'add-in funziona correttamente anche su Access 2003. Oppure può andare sul sito Microsoft [3] e scaricare l'*Access 2003 Plug-in for Visual SourceSafe*. Personalmente, non ho rilevato problemi di compatibilità, ma, se avete un ambiente misto, consiglio prima una ricerca su Google.

Una volta completata l'installazione, l'ambiente di Access si presenta come in **Figura 2**. Le opzioni sono molto intuitive, a parte che alcune sono in italiano ed altre in inglese! Di seguito le traduzioni da sapere:

- Check out = Estrai
- Check in = Archivia
- Undo Check out = Annulla Estrazione

Visual Source Safe come coltellino svizzero

Un sistema di versionamento può essere utile in vari contesti. Lo si può impiegare ovviamente per memorizzare diverse versioni dei programmi. Ad esempio, avendo creato un file Aggiorna.vbs che automatizza l'aggiornamento dei diversi client, se ne possono archiviare le diverse versioni in Visual Source Safe. Un sistema di versionamento è utile non solo per archiviare i soli sorgenti della procedura principale, ma anche per memorizzare gli oggetti più disparati, magari le diverse versioni della bmp che costituisce la schermata di avvio della procedura, oppure i manuali d'uso o i file di progetto. Ci sono aziende che utilizzano sistemi come Visual Source Safe per garantire la storicità dei documenti, richiesta da certificazioni come l'ISO9001.

Access e Visual Source Safe

Chi sviluppa con Access è abituato ad avere pochissimi file da gestire, talvolta anche uno solo, in quanto un singolo file contiene struttura dati, dati, maschere, report, macro e codice. Avere un solo file che contiene tutto è un approccio che risolve molti problemi ma ne crea altri. Volendo salvare diverse versioni di una maschera, si è generalmente costretti a duplicare molte volte il database. Per esempio, se si sta sviluppando il programma Fatture, in assenza di Visual Source Safe, si potrebbero generare dei database chiamati fatture01.mdb, fatture02.mdb etc. Passando a Visual Source Safe, probabilmente quindi, il primo approccio sarebbe quello di utilizzarlo per salvare ogni versione del file .mdb/.adp modificata. Visual Source Safe insieme all'add-in per Access, fornisce invece un sistema molto più efficiente per memorizzare le diverse versioni prodotte, perché, entro certi limiti, consente il salvataggio e, quindi, il versionamento, di un singolo oggetto.

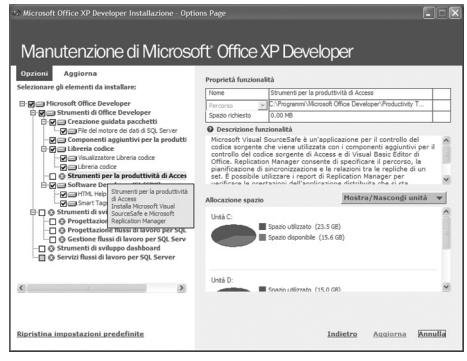


Figura 1 Installazione delle estensioni per Microsoft Access

Il solo grande limite che si incontra è che Visual Source Safe salva in un solo blocco la struttura dei dati ed i dati stessi. Questa limitazione abbastanza forte è dovuta al fatto di preservare l'integrità della base dati. Se, infatti, si potessero avere diverse versioni di una singola tabella, sia come struttura che come dati, ad ogni "prelievo" di una versione, potrebbero verificarsi incongruenze praticamente impossibili da gestire. Si pensi, ad esempio, alla classica situazione Clienti-Ordini: nella versione 2 della tabella clienti, è stato aggiunto il cliente 5, insieme ad alcuni suoi ordini; nel momento in cui si ripristina la versione 1 della stessa tabella che non ha il cliente 5, nella tabella ordini si avrebbero dei record orfani! Attenzione quindi ad eventuali query di creazione tabella. Se le tabelle vengono create quando non è stato fatto il check-out di *Data and Misc. Objects*, esse verranno cancellate al successivo check-out.

Visual Source Safe salva in un solo blocco in formato binario, la struttura dati con le relative relazioni, i dati, le CommandBar e le impostazioni di avvio. In particolare:

- Tabelle (dati e struttura dati)
- Informazioni di connessione per i progetti di Access (.adp)
- Relazioni
- Command Bar definite dall'utente
- Proprietà del database e proprietà di avvio
- Specifiche di importazione/esportazione
- Riferimenti impostati in VBA
- Nome del progetto in VBA
- Argomenti di compilazione condizionale
- Informazioni per la stringa di connessione
- Collegamenti per le pagine di accesso ai dati

Quindi, ogni volta che si devono effettuare modifiche ad una di queste cose, è necessario fare il check-out di *Data and Misc. Objects* - Purtroppo vengono perse eventuali proprietà aggiunte manualmente alla collection CurrentProject.Properties.

Mi ero fatto un'aggiunta al VBE la quale, ad ogni compilazione incrementava un numero di build memorizzato in CurrentProject.Properties("AppBuild"); però, ricreando da zero il database, la proprietà viene persa.

Visual Source Safe permette inoltre, di creare da zero un database.

Questa possibilità risulta molto utile oltre che, ovviamente, per recuperare un database perduto, anche per risolvere tutti quei problemi che ogni tanto si presentano e che i più scafati risolvevano con un bel */decompile*. Ricreando il database da zero la dimensione è la minima possibile e si ottiene un file che non è mai stato sporcato da crash di Access.

Il primo approccio

La prima volta che si lancia Visual Source Safe, se si ha un server, si dovrà fornire nome utente e password, ma soprattutto, la cartella che è stata condivisa sul server. Infatti, Visual Source Safe, se non installato con il NETSETUP.EXE sui client, propone quella creata localmente. Si avvii Access e si crei un nuovo database in una cartella locale. Dal menu *Strumenti/SourceSafe* si selezioni *AddDatabase to SourceSafe...* Se si vuole creare un progetto con più oggetti, nella schermata che si presenta, si crei una cartella, con il pulsante *crea*. Poi, assegnato un nome al database, si prema *Ok.* Nella **Figura 3** si vede un progetto di esempio dell'applicazione *Vendere* che ha le seguenti caratteristiche:

- La cartella $ è la radice di tutti i progetti.
- La cartella *Vendere* è la radice del progetto Vendere e contiene solo i sottoprogetti.

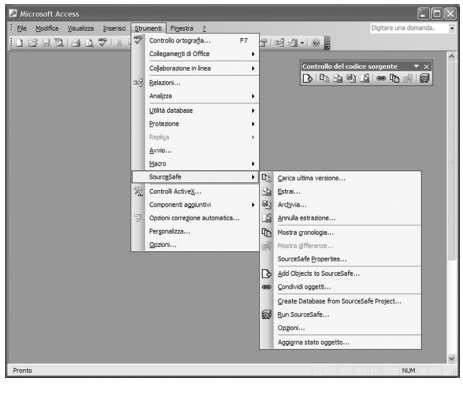


Figura 2 Il menu di Access con le nuove opzioni disponibili

- La cartella Backup contiene le diverse versioni del file Backup.vbs, uno script che effettua il backup dei dati. Come si vede, la cartella non risulta in grigetto, poiché contiene dei file che possono essere estratti anche tramite l'interfaccia di Visual Source Safe.
- La cartella Vendere App contiene il database applicazione Vendere.mdb, memorizzato tramite l'add-in di Access. Per evitare corruzioni, i file non sono manipolabili direttamente dall'interfaccia di Visual Source Safe, quindi la cartella risulta in grigetto.
- La cartella Vendere Dati contiene il database dati VendereDati.mdb, memorizzato tramite l'add-in di Access. Anche in questo caso, essa è manipolabile solo tramite l'apposito add-in.

Adesso, verrà chiesto quali sono gli oggetti da mettere sotto il controllo di Visual Source Safe. Ovviamente, nel caso di un database appena creato, ci sarà solo la struttura dati e quindi, l'unica opzione selezionabile sarà *Data and Misc. Objects*. Se si fosse aggiunto a Visual Source Safe un database con diversi oggetti, si sarebbero potuti specificare gli oggetti da far tenere sotto controllo a Visual Source Safe. Il messaggio che si ottiene alla fine dell'operazione, se tutto è andato bene, è molto insistente e ricorre spesso, perché è importante. Ricorda che, per fare modifiche ai dati, quindi per aggiungere un record, ad esempio, *si deve* sempre prima fare un check-out di *Data and Misc. Objects*, poi modificare i dati ed infine rifare un check-in. Eventuali modifiche effettuate senza questa procedura, verranno sovrascritte la prima volta che si preleverà l'ultima versione o si effettuerà il check-out di *Data and Misc. Objects*.

Lavorare con Visual Source Safe

Un database sotto controllo di Visual Source Safe e in ambiente multiutente si presenta come in **Figura 4**. Il significato delle icone è:

- Il lucchetto della Maschera1 indica che essa è sotto il controllo di Visual Source Safe e che nessuno ci sta lavorando.
- Il segno di spunta della Maschera2 indica che di essa è stato fatto il check-out dall'utente del database. Quindi, è possibile effettuare modifiche sulla maschera.
- L'icona dell'omino della Maschera3 indica che qualcun altro ha fatto il check-out della maschera e, quindi, attualmente, non è possibile modificarla.
- La Maschera4 non ha icone, ossia non è sotto il controllo di Visual Source Safe. Se il database andasse perso, non sarebbe possibile recuperarla.

Per modificare un oggetto, si deve effettuare prima il check-out, in italiano *estrai*. Eseguita l'operazione, si modifica l'oggetto, lo si salva e si effettua il check-in, in italia-

no *archivia*. Per annullare l'operazione di check-out senza modificare il numero di versione dell'oggetto, si utilizza l'opzione *Annulla estrazione*.

Tramite l'opzione *Mostra cronologia*, si ha accesso alle diverse versioni dell'oggetto.

Nell'uso standard di Visual Source Safe, se si è abilitata l'apposita opzione, è possibile effettuare un check-out multiplo di un oggetto.

Lavorando in accoppiata con Access, questa opzione viene limitata ai soli moduli. Il motivo è semplice: se due persone modificano lo stesso sorgente, al momento dell'ultimo check-out, Visual Source Safe tenta il merge automatico dei sorgenti. Se non riesce, visualizza una finestra che permette di vedere le differenze tra file e di riconciliare i conflitti.

Ma tutto questo, se l'oggetto modificato è una form, non è possibile e lo è ancor meno se si tratta di *Data and Misc. Objects*, che è un oggetto binario. In ambiente multiutente, si devono limitare i check-out allo stretto indispensabile, per non rischiare di bloccare inutilmente qualche altro sviluppatore.

Feature e problemi

- *Processo di autenticazione*
  Accedendo ad un database di Visual Source Safe, viene richiesta l'autenticazione. Questo processo può essere evitato se si ha cura di chiamare gli utenti di Visual Source Safe con lo stesso nome che utilizzano per l'accesso in rete. Si presenta però il problema inverso, volendo accedere a Visual Source Safe come diverso utente (magari come Admin). La soluzione più veloce sperimentata, se si dispone dei tool di amministrazione, è stata di cambiare al volo il proprio nome su Visual Source Safe. Al successivo tentativo di accesso, verrà richiesta l'autenticazione.
- *Marcatura del database con il nome utente*
  L'add-in di Access, quando si crea un database da Visual Source Safe, lo marca con il nome dell'utente, per ragioni di coerenza. Se lo stesso database viene aperto da un altro utente, si ottiene il messaggio *Another user (<nome utente>) has placed this database under source code control and is the only person who should work with it. Source code control features will be disabled*. In conseguenza vengono disabilitati i tool di Visual Source Safe. Il problema si può presentare perché un altro utente, da un altro pc, apre il database, oppure perché con i tool di amministrazione è stato cambiato il nome dell'utente su Visual Source Safe. Le soluzioni praticabili sono di aprire il database con il nome utente giusto oppure cancellarlo e ricrearlo.
- *La cartella <nomedatabase>.scc*
  Quando un database viene posto sotto il controllo di Visual Source Safe, nella cartella che lo contiene, viene creata una sottocartella che ha lo stesso nome del database ed estensione .scc. Visual Source Safe non conosce né deve conoscere come Access memorizzi i propri dati, perciò l'architettura prevede che l'add-in di Access esporti ed importi dei file che poi vengono memorizzati e gestiti da Visual Source Safe. In pratica, quando si crea un nuovo progetto tramite l'add-in, i vari oggetti di Access vengono esportati in altrettanti file; in una seconda fase, questi file vengono esportati su Visual Source Safe. Viceversa succede quando importiamo un qualche oggetto da Visual Source Safe. In passato, la cartella veniva creata all'inizio della sessione di lavoro e distrutta alla fine. Oggi, viene mantenuta per ragioni di prestazioni. Potete però cancellarla tranquillamente: verrà ricreata alla prima sessione. Ricordarsi di questa cosa potrebbe essere importante. Per essere sicuri di ripartire con un database pulito, è sufficiente cancellare il database stesso e la sua cartella .scc e, infine, ricrearlo con l'opzione *Strumenti/SourceSafe/Create Database from SourceSafe Project*.

Riquadro 1 Curiosità: il formato interno di Access

Chi non ha iniziato a lavorare con Access, forse si è sempre chiesto come vengano memorizzati gli oggetti internamente. La documentazione è praticamente inesistente, si ha conoscenza solo che esistono delle tabelle di sistema che, di solito sono nascoste.

Bene, è giunto il momento di dare uno sguardo alla cartella <nomefile>.scc, che viene creata ogni qualvolta si lavori con un database memorizzato in Visual Source Safe. Ogni file, come si vede in Tabella 1, contiene uno o più oggetti di Access. Aprendo un file di una maschera, si trova una piacevole sorpresa, ossia una sintassi piuttosto simile a quella usata dal Visual Basic per le form.

Risolvere i problemi

Un problema molto strano (cercando su Google, c'è un tizio che ci ha perso 40 ore!) riguarda il ripristino di un database salvato su Visual Source Safe; se si ottiene un messaggio tipo *Failed to import file '<nomefile>' into Microsoft Access*, il problema potrebbe essere dovuto al formato predefinito di file di Access. Ossia, se il database inserito in Visual Source Safe è in formato Access 2002/2003 e il formato predefinito dell'Access che effettua il ripristino è il 2000 si verifica il problema. Passi per riprodurlo:

- Creare un database in formato Access 2002/2003
- Creare una maschera vuota
- Aggiungere un pulsante di comando
- Aggiungere il seguente codice sull'evento Click del pulsante:

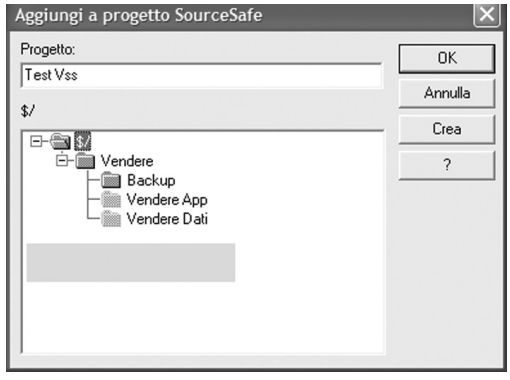


Figura 3 Fase di creazione di un progetto in Visual Source Safe

```
Private Sub Comando0_Click()
    MsgBox "Access Build: " & Application.Build
End Sub
```

- Salvare la maschera
- Aggiungere il database a Visual Source Safe tramite *Strumenti/SourceSafe/ AddDatabase to SourceSafe...*
- Andare in *Strumenti/Opzioni/Avanzate* ed impostare il *Formato di file predefinito* a Access 2000
- Cancellare il database in locale e la cartella <nome database.scc>
- Tramite *Strumenti/SourceSafe/Create Database from SourceSafe Project* ricreare il database.

All'ultimo passo dovrebbe verificarsi il problema precedentemente esposto.

Può succedere inoltre, che un programmatore abbia vinto alla lotteria e parta la notte stessa per una vacanza in un posto romantico. E, che nella fretta della partenza, si sia scordato che aveva lasciato dei file bloccati, ossia, ne aveva fatto il check-out ma mai il check-in. La stessa cosa può avvenire se il programmatore perde il database sul quale stava lavorando, magari per un guasto dell'hard disk. In questi casi, si presenta il problema di sbloccare quei file. Provando a farlo manualmente, ci si accorge che l'operazione non è consentita da Visual Source Safe, che ci segnala *This is an integration-only project; getting files is not allowed from the SourceSafe Explorer*.. Solo l'utente che ha fatto il check-out di un oggetto può sbloccarlo. La soluzione consiste nell'intervenire su un file di configurazione che modifica questo comportamento.

Nella cartella che contiene il database di Visual Source Safe (di standard: C:\Programmi\Microsoft Visual Studio\VSS) si trova il file *srcsafe.ini* il quale, per ogni progetto creato dall'add-in di Access, contiene una sezione del tipo

```
[$/TestVSS]
Disable_UI = Yes
```

È sufficiente, con un editor di testo, cancellare temporaneamente la riga in questione affinché Visual Souce Safe riabiliti i comandi *Check Out, Check In, Undo Check Out* e *Get Latest Version*. Una volta annullato il checkout del file bloccato, è buona pratica ripristinare la riga, per evitare che qualche utente inesperto faccia danni lavorando direttamente dall'interfaccia di Visual Source Safe.

Come distribuire un database

Per distribuire un database, lo si deve togliere dal controllo del codice sorgente. Per farlo, è sufficiente compattare il database e rispondere affermativamente alla domanda di Access se togliere il database dal controllo del codice sorgente.

In produzione, di solito, si fa qualche passo in più.

- Se il database dell'applicazione ha una tabella in cui viene memorizzata la versione, si effettua il check-out di *Data and Misc. Objects*, e si modifica la versione
- Si effettua il check-in di tutti gli oggetti modificati
- Si aggiunge un'etichetta che segnala qual è la versione effettivamente distribuita al cliente
- Si copia il database in una cartella che non sia quella di lavoro (altrimenti, riprendendo a sviluppare, ci si troverebbe con il database non più sotto il controllo di Visual Source Safe)
- Si compatta il database e lo si distribuisce.

Figura 4 La finestra del database di Access con le nuove icone

Tabella 1 Estensioni dei file per i corrispondenti oggetti di Access memorizzati in VSS

Oggetto	Campo di applicazione	Estensione
Query	.mdb	.acq
Maschere	.mdb .adp	.acf
Report	.mdb .adp	.acr
Macro	.mdb .adp	.acs
Moduli	.mdb .adp	.acm
Dati ed oggetti vari	.mdb	.acb
Dati ed oggetti vari	.adp	.acp
Nome del file di database	.mdb .adp	.acn

Ricordarsi di tutte queste procedure non è semplice.

In ogni caso è consigliabile avere sempre sott'occhio almeno una checklist dei passi da compiere.

Lavorare a casa con un DB sotto VSS

Per lavorare staccandosi dal cordone ombelicale di Visual Source Safe, è sufficiente effettuare il check-out di tutti gli oggetti che si prevede di dover modificare (o, se nessun'altro deve sviluppare sulla stessa applicazione, direttamente di *tutti* gli oggetti), e copiare il solo file di database.

Al rientro, si copierà il file sul pc di sviluppo e, dopo aver aperto Access, si farà il check-in di tutto. Visual Source Safe provvederà al salvataggio degli oggetti modificati ed al rilascio di tutti gli altri.

Consiglio: prima di affidarsi completamente a questa procedura, è buona prassi collaudarla, per escludere qualsiasi imprevisto.

Riferimenti alla (poca) documentazione disponibile

In rete, l'articolo più interessante che ho trovato è *Source Code Control in Microsoft Access 2000*, presente sul sito MSDN al link http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnacc2k/html/srcctrl.asp.

Per il resto, la maggior parte di materiale relativo a Visual Source Safe non comprende l'interazione con Microsoft Access.

Per sopravvivere in caso di problemi, resta soltanto l'ottimo Google nella sezione *gruppi*.

Conclusioni

Lavorare con Visual Source Safe, in definitiva, conviene se:

- Si sviluppano molti prodotti con Access. Avere un sicuro sistema di catalogazione con un backup centralizzato fa comodo.
- Si sviluppa un prodotto distribuito in molti esemplari. Avere accesso alle diverse versioni e poter "ramificare" lo sviluppo, è veramente utile.
- Si lavora in ambiente multiutente. Sopra i due utenti, un sistema di versionamento è praticamente indispensabile.
- Si lavora in due ma uno dei due è spesso dal cliente o, comunque, sviluppa fisicamente non nello stesso ambiente. Automatizzare la gestione dei conflitti, avere un merge dei sorgenti semiautomatico e comunque uno strumento che consente in breve tempo di riconciliare modifiche contemporanee allo stesso file, è quasi indispensabile.

Si può fare a meno di Visual Source Safe se i prodotti gestiti sono pochi e lo sviluppatore è unico. In questo caso, i vantaggi di non essere in alcuna maniera legati ad altri software che Microsoft Access e la maggiore rapidità di sviluppo, compensano gli svantaggi di non utilizzare un sistema di versionamento.

Ma, in questo caso, è indispensabile prevedere una buona strategia di backup. Infatti, anche se Access è sempre più affidabile, molto raramente può capitare che un crash danneggi irrimediabilmente un database, rendendo irrecuperabile il lavoro.

Riferimenti

[1] INFO: Required Network Rights for the SourceSafe Directories http://support.microsoft.com/default.aspx?scid=kb;EN-US;131022

[2] Description of the files and the service packs that you have to have so that you can use an Access database under Visual SourceSafe control in Access 2003 http://support.microsoft.com/?id=837136

[3] Visual SourceSafe http://msdn.microsoft.com/vstudio/previous/ssafe/ (seleziona "Access 2003 Plug-in for Visual SourceSafe")

Azioni referenziali con SQL Server 2000

L'implementazione fisica di un database può comportare l'adozione di metodi alternativi tra loro la cui scelta richiede accurate decisioni di progetto. È il caso dell'integrità referenziale, che un DBA SQL Server può implementare in diversi modi.

di Francesco Quaratino

Nel ciclo di vita di un database, la definizione dell'integrità dei dati si colloca in una fase decisiva per la buona riuscita di un'applicazione orientata ai dati: il disegno logico. A seconda che si tratti di un sistema utilizzato per la registrazione di transazioni (OLTP - OnLine Transaction Processing) o di supporto decisionale (OLAP – OnLine Analytical Processing), il progettista seguirà differenti metodologie per derivare il disegno logico da quello concettuale definito dall'analista - quelle applicate ai database OLTP focalizzano la loro attenzione sui tempi di completamento della singola transazione e sulla precisione nell'aggiornamento dei dati, mentre quelle applicate ai database OLAP si incentreranno sui tempi di risposta delle interrogazioni analitiche. Il progettista applicherà, quindi, le regole di normalizzazione per preparare il disegno logico di un sistema OLTP all'implementazione fisica attraverso lo schema relazionale, oppure le regole di denormalizzazione per preparare il disegno logico di un sistema OLAP all'implementazione fisica attraverso lo schema dimensionale. Subito dopo, si procederà alla definizione dell'integrità del database, a cui spetta il controllo della consistenza e della precisione dei dati. Questo articolo tratta l'implementazione fisica con SQL Server 2000 dell'integrità dei dati in un database OLTP concentrandosi su quella referenziale.

Consistenza e precisione dei dati

L'applicazione dell'integrità dei dati assicura che un dato possa accettare un determinato insieme di valori (Integrità di dominio), che ogni tabella abbia un insieme di campi che fungano da indentificatore univoco di ogni riga (Integrità di entità) e che le relazioni definite tra le tabelle non possano essere violate (Integrità referenziale). L'integrità referenziale coinvolge dunque due tabelle: una referenced table (tabella referenziata) che contiene la PRIMARY KEY (PK) e una referencing table (tabella referenziante) che si lega ad essa attraverso una FOREIGN KEY (FK). In alternativa, la FK può far riferimento ad una colonna (o un insieme di colonne) su cui è definito un vincolo di unicità (UNIQUE CONSTRAINT). In ogni caso occorre che il tipo di dato delle colonne in relazione sia identico (salvo l'attributo di [NOT] NULL). Tale legame può essere più o meno "forte" e condizionare le modifiche del valore della PK e la cancellazione di un'istanza della tabella referenziata qualora esista almeno un'istanza della tabella referenziante che faccia riferimento ad essa. In ogni caso, la relazione condizionerà la tabella referenziante poiché in essa non sarà possibile inserire o aggiornare il valore di una chiave esterna se tale valore non esiste in un'istanza della PK della tabella referenziata. Lo standard ANSI SQL-92 parla di azioni referenziali al verificarsi di aggiornamenti della tabella referenziata e ne definisce quattro: NO ACTION, SET NULL, SET DEFAULT, CASCADE. La NO ACTION impedisce le modifiche alla PK referenziata da una FK. Essa rappresenta il comportamento di de-

Francesco Quaratino *è consulente in ambito di progettazione e amministrazione di database OLTP e OLAP, oltre che di applicazioni software orientati ai dati. È certificato MCDBA e sviluppa in VB6, ASP, VB.NET e ASP.NET. Può essere contatto all'indirizzo e–mail: fquaratino@infomedia.it*

fault di un legame referenziale nonché l'unico scoglio che un database deve superare perché si possa definirlo conforme allo standard SQL-92. SET NULL consente la cancellazione e la modifica e produce un aggiornamento del valore della FK al valore NULL.

Allo stesso modo, SET DEFAULT imposta il valore della FK al suo valore di DEFAULT. CASCADE consente la cancellazione e la modifica, producendo, nel primo caso, la cancellazione di tutte le istanze in relazione della tabella referenziante, e nel secondo caso l'aggiornamento del valore della FK.

Listato 1 Esempi di trigger che implementano le azioni di SET DEFAULT e SET NULL

```
/* Esempio di creazione di un trigger che implementa
l'azione referenziale di SET DEFAULT
** a seguito dell'aggiornamento della chiave primaria
della tabella referenziata */

CREATE TRIGGER tr_set_default_authors ON authors FOR
UPDATE
AS
DECLARE @rows_effected int
BEGIN

SET @rows_effected = @@ROWCOUNT
IF @rows_effected = 0 RETURN

IF UPDATE(au_id)
  IF @rows_effected = 1
    UPDATE titleauthor
    SET au_id = DEFAULT
    FROM titleauthor INNER JOIN Deleted ON
titleauthor.au_id = Deleted.au_id
    INNER JOIN Inserted ON Inserted.au_id <>
Deleted.au_id
  ELSE
  BEGIN
    RAISERROR ('Si è cercato di aggiornare la chiave
primaria di %d righe',16,1,@rows_effected)
    ROLLBACK TRANSACTION
  END

END
GO

/* Esempio di creazione di un trigger che implementa
l'azione referenziale di SET NULL
** a seguito dell'eliminazione di una riga della tabella
referenziata */

CREATE TRIGGER tr_set_null_authors ON authors FOR DELETE
AS
DECLARE @rows_effected int
BEGIN

IF @@ROWCOUNT=0 RETURN

UPDATE titleauthor
SET au_id = NULL
WHERE au_id IN (SELECT DISTINCT au_id FROM Deleted)

END
```

Come implementare l'integrità referenziale

L'implementazione dell'integrità referenziale in SQL Server 2000 può avvenire in modo dichiarativo, attraverso la definizione di un oggetto interno a SQL Server 2000 denominato FOREIGN KEY, e in modo procedurale, ovvero con l'ausilio di procedure ad-hoc implementate attraverso trigger o stored procedure.

Microsoft suggerisce ufficialmente l'adozione dell'integrità dichiarativa per le azioni referenziali relegando quella procedurale al compito di risolvere logiche aziendali più complicate.

Nulla vieta, però, di implementare proprio l'integrità referenziale attraverso i trigger, soprattutto se teniamo conto che solo due delle quattro azioni referenziali definite dallo standard SQL-92 sono supportate in modo dichiarativo da SQL Server 2000: NO ACTION e CASCADE.

Dichiarare un vincolo di FOREIGN KEY rappresenta sicuramente il modo più semplice di agire sulle azioni referenziali, ma è bene tener presente alcune limitazioni e inconvenienti in cui ci si potrebbe imbattere. Innanzitutto, non è possibile usare contemporaneamente sulla stessa relazione sia l'integrità dichiarativa che i trigger, poiché nel momento della violazione del vincolo dichiarativo, l'istruzione non viene eseguita ed il trigger non parte. Dal punto di vista delle performance, un uso eccessivo di FOREIGN KEY può causare un degrado delle prestazioni, quindi è opportuno considerare la creazione di un index sulla FK (qualora non fosse già parte di un clustered index) per consentire operazioni di join più efficienti. È bene ricordare anche che i comandi di caricamento di massa bcp o BULK INSERT non prevedono, di default, il controllo dei vincoli referenziali – proprio allo scopo di rendere tali operazioni più veloci.

Quello che segue è un esempio di dichiarazione di un vincolo che prevede il NO ACTION a seguito della cancellazione e il CASCADE a seguito dell'aggiornamento:

```
ALTER TABLE titleauthor ADD
CONSTRAINT FK_titleauthor_author
FOREIGN KEY (au_id) REFERENCES authors (au_id)
ON DELETE NO ACTION ON UPDATE CASCADE
```

Usare i trigger

Rispetto all'uso dei vincoli dichiarativi, i trigger consentono al DBA SQL Server alcuni piccoli vantaggi a fronte di un costo di produzione e manutenzione più elevato: danno la possibilità di visualizzare messaggi di errore personalizzati per rendere più esplicativi quelli generici inviati dopo la violazione delle relazioni dichiarative; consentono l'implementazione delle quattro azioni referenziali descritte dallo standard SQL-92 (inclusi SET

NULL e SET DEFAULT); permettono il CASCADE su una tabella auto-referenziata. Il **Listato 1** mostra due esempi che implementano il SET NULL e il SET DEFAULT. Si noti l'uso della variabile di sistema @@ROWCOUNT. Essa contiene il numero di righe coinvolte nell'ultima istruzione ed è opportuno valutarla poiché il trigger scatta anche se l'operazione sulla tabella non influenza nessuna riga. In questo modo è possibile controllare le situazioni in cui un'istruzione non sortisce l'effetto desiderato e, per esempio, lanciare un messaggio personalizzato dall'interno del trigger. Inoltre, si noti che, qualora il valore di @@ROWCOUNT debba essere valutato in più volte all'interno del trigger, esso è assegnato ad una variabile dal momento che viene impostato a 0 (zero) da qualsiasi altra istruzione.

Usando i trigger per implementare le azioni di SET NULL o SET DEFAULT, occorre necessariamente utilizzare i trigger per le azioni di NO ACTION o di CASCADE, a causa dell'incompatibilità fra trigger e integrità dichiarativa. Per impedire la violazione della relazione bisognerà creare un trigger sulla tabella referenziante che gestirà l'INSERT e l'UPDATE verificando la presenza di un'istanza correlata nella tabella referenziante. Per il CASCADE, invece, occorrerà creare sulla tabella referenziata un trigger per gestire l'UPDATE e uno per la DELETE. Il **Listato 2** presenta un esempio di implementazione del CASCADE e del NO ACTION.

Negli esempi precedenti, si può notare come sia stata limitata l'applicazione dei trigger AFTER UPDATE ad istruzioni che non coinvolgono righe multiple di chiavi primarie, infatti, in caso di @@ROWCOUNT > 1 la procedura visualizza un messaggio d'errore personalizzato e annulla la transazione. Intendo riferirmi ad istruzioni come questa:

Listato 2 Esempi di trigger che implementano le azioni di CASCADE e NO ACTION

```
/* Esempio di creazione di un trigger che implementa
l'azione referenziale di
** CASCADE ON UPDATE sulla tabella referenziata */

CREATE TRIGGER tr_update_authors ON authors
FOR UPDATE
AS
DECLARE @rows_effected int
BEGIN

SET @rows_effected = @@ROWCOUNT
IF @rows_effected = 0
  RETURN

IF UPDATE(au_id)
  IF @rows_effected = 1
    UPDATE titleauthor
      SET au_id = Inserted.au_id
    FROM titleauthor INNER JOIN Deleted ON
titleauthor.au_id = Deleted.au_id
      INNER JOIN Inserted ON Inserted.au_id <>
Deleted.au_id
  ELSE
  BEGIN
    RAISERROR ('Si è cercato di aggiornare la chiave
primaria di %d righe',16,1,@rows_effected)
    ROLLBACK TRANSACTION
  END

END
GO

/* Esempio di creazione di un trigger che implementa
l'azione referenziale di
** CASCADE ON DELETE sulla tabella referenziata */

CREATE TRIGGER tr_delete_authors ON authors
FOR DELETE
AS
BEGIN

IF @@ROWCOUNT=0
  RETURN

DELETE FROM titleauthor
  WHERE au_id IN (SELECT DISTINCT au_id FROM Deleted)

END
GO

/* Esempio di creazione di un trigger che implementa
l'azione referenziale di
** NO ACTION ON INSERT e ON UPDATE sulla tabella refe-
renziante */

CREATE TRIGGER tr_insert_titleauthor ON titleauthor
FOR INSERT, UPDATE
AS
BEGIN

IF @@ROWCOUNT = 0
  RETURN

IF UPDATE(au_id)
  IF NOT EXISTS
    (SELECT authors.au_id FROM authors INNER JOIN In-
serted
      ON authors.au_id = Inserted.au_id)
  BEGIN
    RAISERROR ('Non esiste un record correlato nella
tabella padre',16,1)
    ROLLBACK TRANSACTION
  END

END
GO
```

```
UPDATE authors SET phone = au_id + 10
```

con la quale non è possibile stabilire la corrispondenza tra le righe delle pseudo-tabelle Inserted e Deleted dal momento che per un database relazionale ogni tabella è un insieme non ordinato di righe. Non è quindi una soluzione valida neanche l'uso di cursori per scorrere le due pseudo-tabelle. Una soluzione possibile si basa invece sulla presenza di un altro campo che ammette valori unici (per semplicità di gestione si potrebbe usare un campo IDENTITY) col quale stabilire la corrispondenza tra Inserted e Deleted. Un'altra soluzione è quella di disabilitare temporaneamente i trigger su entrambe le tabelle e procedere ad un aggiornamento di massa delle tabelle relazionate, quindi riabilitare i trigger, usando il seguente comando T-SQL:

```
ALTER TABLE table_name
{ ENABLE | DISABLE } TRIGGER { ALL | trigger_name [ ,...n ] }
```

Conclusioni

La prima cosa che un programmatore dovrebbe pretendere da un database, è che esso mantenga logicamente integri i propri dati e che le scelte d'implementazione fisica delle regole d'integrità non ne compromettano le performance. A tale scopo è fondamentale che un DBA conosca tutte le tecniche che il DBMS mette a sua disposizione, in modo da ponderare bene le sue scelte in fase di disegno logico, considerando anche le inevitabili limitazioni cui queste possono portare. In genere, l'integrità referenziale dichiarativa è la scelta migliore per implementare il NO ACTION poiché riduce al minimo l'introduzione di errori nel codice scritto nei trigger. Questi ultimi rappresentano una carta in più da giocare per fronteggiare logiche di dati più complesse. Chi si avvicina a SQL Server reduce dall'uso DBMS con funzionalità più ridotte (come Access e MySQL), è spesso restio al contatto con i trigger ed in genere con Transact-SQL, ma è bene sapere che è stato SQL Server 6.0 ad introdurre in casa Microsoft i vincoli di chiave primaria ed esterna, nonché le relazioni dichiarative - anche se limitate al NO ACTION - e solo con l'avvento di SQL Server 2000 si è avuto il supporto del CASCADE (che già Access possedeva da un bel po'). Insomma, con le versioni precedenti alla 6.0 bisognava, per forza di cose, essere dei discreti programmatori T-SQL e scrivere i trigger necessari a mantenere integri i dati.

Bibliografia

[1] Dino Esposito - "Dati integri e controllati con i trigger", VBJ n. 59

[2] Kalen Delaney - "Inside Microsoft SQL Server 2000", Mondadori Informatica/2000

Definizione esplicita dei [WebMethod]

L'attributo [WebMethod] è estremamente utile e potente per lo sviluppo dei Web service in .NET. Ma se non usato correttamente può avere degli effetti collaterali che possono impedire il corretto funzionamento del servizio.

di Ingo Rammer

I servizi Web di ASP.NET e l'attributo [WebMethod] hanno sicuramente rivoluzionato lo sviluppo dei Web service. Invece di creare a mano e interpretare i messaggi XML, questo attributo crea per noi la definizione del servizio in WSDL (Web Service Definition Language) e permette di usare trasparentemente XmlSerializer lato client e lato server.

Ci sono essenzialmente due approcci che si possono adottare nello sviluppo dei Web service: "Code-First" e "Contract-First". Nel primo caso, usato per default in Visual Studio .NET, si definisce un Web service con l'attributo [WebMethod] e ci si affida al .NET Framework per la creazione della rappresentazione XML sottostante. Per accedere da un client alla descrizione del servizio e allo schema XML si aggiunge "?WSDL" all'URL del Web service. Dato che il WSDL in realtà definisce l'interfaccia del servizio, cioè il *contratto* con i suoi client, si sta diffondendo l'approccio "Contract-First", che prevede prima la creazione dello schema XSD e del WSDL. Poi con tool come XSD.EXE e WSDL.EXE forniti col .NET Framework si possono generare automaticamente i pezzi di codice corrispondenti in C# e VB da includere nel progetto. Il mio collega in thinktecture Christian Weyer ha inoltre creato un add-in gratuito per Visual Studio chiamato WSCF (WsContractFirst) che semplifica questa procedura. È disponibile in **[1]**. Se si usa il modello Contract-First, bisogna assicurarsi che il *contratto* tra il client e il server sia definito esplicitamente in modo che non ci siano modifiche non pianificate. Il problema in questo caso è che la creazione dell'XSD e del WSDL è ancora abbastanza complessa perché ci sono pochi tool disponibili. Invece di perdere tempo tra i dettagli degli schemi XSD e WSDL, nell'articolo mostrerò come definire direttamente i [WebMethod] in modo esplicito.

thinktecture

***Ingo Rammer** è il fondatore di thinktecture, una compagnia che aiuta gli sviluppatori e gli architetti software a progettare e implementare applicazioni e Web service .NET. Partecipa come speaker alle conferenze internazionali dedicate a tali argomenti ed è autore del best–seller Advanced .NET Remoting (APress). È Microsoft Regional Director per l'Austria e può essere contattato tramite il sito http://www.thinktecture.com/staff/ingo.*

Modifiche non pianificate allo schema

Prima di tutto mostriamo qual è il vero problema: supponiamo di aver creato un Web service con la seguente definizione:

```
[WebService(Namespace="http://
                thinktecture.com/prod")]
public class ProductService : System.Web
                    .Services.WebService
{
   public void AddArticle(Article art)
   {
      // ...
   }
}

public class Article
{
   public int ID;
   public double Price;
   public String Description;
}
```

Supponiamo di aver bisogno, dopo un po' di tempo, di modificare alcune

parti dell'applicazione. Durante il refactoring, notiamo che i campi nella classe *Article* hanno un ordine differente rispetto a quello del database. Decidiamo quindi di cambiarli in modo che *Description* preceda *Price*:

```
public class Article
{
   public int ID;
   public String Description;
   public double Price;
}
```

Questa è una modifica legale nel .NET Framework, e non invalida i client precompilati. Ma cosa accade al Web service? *Tutti* i client continueranno a funzionare? Sembra di sì, ma è solo apparenza. Nella prima versione di *Article*, .NET aveva generato il seguente schema:

```
<s:complexType name="Article">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="ID" type="s:int" />
    <s:element minOccurs="1" maxOccurs="1" name="Price"
                                              type="s:double" />
    <s:element minOccurs="0" maxOccurs="1" name="Description"
                                              type="s:string" />
  </s:sequence>
</s:complexType>
```

Questo significa che l'XML che invoca il servizio avrebbe potuto essere:

```
<AddArticle xmlns="http://thinktecture.com/prod">
  <art>
    <ID>4711</ID>
    <Price>47.11</Price>
    <Description>Demo Article</Description>
  </art>
</AddArticle>
```

Nella seconda versione gli ultimi due <s:element> dello schema sono stati invertiti. Se il Web service riceve un messaggio nel "vecchio" formato (con *Price* prima di *Description*), .NET assegnerà comunque i valori corretti ai rispettivi campi. Tutto *sembra* funzionare correttamente.

Ma questa è solo una parte della verità, perché il messaggio in entrata non è valido secondo lo schema XML. Se si usasse un validatore automatico di schemi simile a [2], si noterebbe come il controllo fallirebbe e il messaggio sarebbe incorretto.

Il problema si rivela quando si comunica con differenti piattaforme (come Java) o si usano software di orchestration come BizTalk. Ecco perché bisogna evitare questo tipo di modifiche, anche se si lavora esclusivamente in .NET.

Lo stesso problema si ha se si cambia il nome dei parametri del metodo. È una modifica legale in .NET, ma non su altre piattaforme.

Namespace condivisi

Se si usano frammenti di XML basati su classi (come *Article*) molto spesso si avrà la necessità di metterli nello stesso namespace. Nell'esempio precedente l'elemento <art> è nello namespace del Web service. Se si volesse riutilizzare la classe *Article* in un secondo Web service, la classe finirebbe in un namespace differente. Così, per assicurare la riusabilità tra gli elementi dello schema, è bene definire esplicitamente il namespace XML per ogni classe che si intende serializzare in XML:

```
[XmlRoot("Article", Namespace="http://thinktecture.com/art")]
public class Article
{
  public int ID;
  public String Description;
  public double Price;
}
```

Se si chiama un Web service così definito, sarà generato un XML nel SOAP Body simile al seguente:

```
<AddArticle xmlns="http://thinktecture.com/prod">
  <art xmlns="http://thinktecture.com/art">
    <ID>4711</ID>
    <Description>Demo Article</Description>
    <Price>47.11</Price>
  </art>
</AddArticle>
```

L'elemento <art> (che è basato sul tipo *Article*) e i suoi contenuti vivono in un namespace separato; quindi la definizione dello schema può essere riutilizzata tra più Web service.

Si possono usare attributi simili per definire esplicitamente il nome dei parametri del metodo serializzati nel messaggio. Con l'attributo [XmlElement], si può per esempio modificare il nome dell'elemento XML indipendentemente dal nome dei parametri nel sorgente:

```
[WebMethod]
public void AddArticle(
   [XmlElement("Article", Namespace="http://thinktecture.
                                        com/art")] Article art)
{
   // ...
}
```

Quando si usa questa definizione, si riceverà un frammento XML simile al seguente. Notate come l'informa-

zione sull'articolo è ora serializzata in un elemento chiamato <Article>, anche se il parametro del metodo è comunque chiamato "art":

```
<AddArticle xmlns="http://thinktecture.com/prod">
  <Article xmlns="http://thinktecture.com/art">
    <ID>4711</ID>
    <Description>Demo Article</Description>
    <Price>47.11</Price>
  </Article>
</AddArticle>
```

Definizione esplicita di messaggi

Sono sicuro che negli esempi precedenti avrete notato che è stato generato un altro elemento <AddArticle>. Questo elemento aggiuntivo wrappa il parametro reale del metodo. Il messaggio di risposta del Web service conterrà un elemento chiamato <AddArticleResponse>. Questo potrebbe far pensare che l'utilizzo di questi elementi aggiuntivi sia obbligatorio nel mapping tra il messaggio XML e il nome del metodo. In realtà, comunque, il vero scopo dei wrapper è semplicemente combinare tutti i parametri del metodo in un singolo elemento. Molti puristi dei Web service – e credo di essere uno di questi – non guardano tanto alla nozione di "chiamata a metodo" per Web service, ma piuttosto ai messaggi che si scambiano tra mittente e ricevente. Questo wrapper quindi è un'interferenza e prende il posto degli elementi top-level. Per questo ASP.NET offre il cosiddetto metodo "bare", in cui lo sviluppatore ha la completa responsabilità di definire il body del metodo. Così si possono definire esplicitamente i messaggi di richiesta e risposta. Per avere una comunicazione simile a quella precedente in cui la richiesta contiene un messaggio NewArticle e la risposta un messaggio NewArticleResponse, si crea un codice simile al seguente. Come primo passo si definiscono i messaggi di richiesta e risposta come classi serializzabili in XML:

```
[XmlRoot("NewArticle", Namespace="http://thinktecture.com/prod")]
public class NewArticle
{
   [XmlElement("Article", Namespace="http://thinktecture.
                                                com/art")]
   public Article Article;
}

[XmlRoot("NewArticleResponse", Namespace="http:/
                                    thinktecture.com/prod")]
public class NewArticleResponse
{
}
```

Dopo, è possible marcare il Web service con [SoapDocumentMethod] del namespace System.Web.Services.Protocols. Questo permette di selezionare la modalità "bare" per lo stile dei parametri.

Così facendo è permesso un solo parametro in input e un solo valore di ritorno (senza parametri *out* o *ref*).

Questi parametri conterranno il messaggio XML completo.

```
[WebMethod]
[SoapDocumentMethod(ParameterStyle=SoapParameterStyle.Bare)]
public NewArticleResponse AddArticle(NewArticle req)
{
  return new NewArticleResponse();
}
```

Si ha il pieno controllo sui messaggi trasferiti al/dal server. Il SOAP Body diventa:

```
<NewArticle xmlns="http://thinktecture.com/prod">
  <Article xmlns="http://thinktecture.com/art">
    <ID>4711</ID>
    <Description>Demo Article</Description>
    <Price>47.11</Price>
  </Article>
</NewArticle>
```

Conclusioni

Se si definiscono i messaggi dei servizi Web in maniera esplicita, si ottiene il massimo disaccoppiamento tra messaggi XML e codice sorgente.

Questo è estremamente importante perché permette di assicurare che non ci saranno modifiche non pianificate nella definizione del messaggio. Altrimenti le modifiche al codice sorgente (anche semplici, come il riordinamento dei field di classe o la modifica del nome dei parametri) potrebbe cambiare lo schema e invalidare i messaggi esistenti inviati dai client.

Potrebbe sembrare un'esagerazione, ma è molto importante che i sistemi client siano sviluppati al di fuori della vostra responsabilità. Se l'applicazione offre servizi a sviluppatori al di fuori della propria compagnia o dipartimento dovete garantire che non ci saranno modifiche non pianificate nel formato dei messaggi.

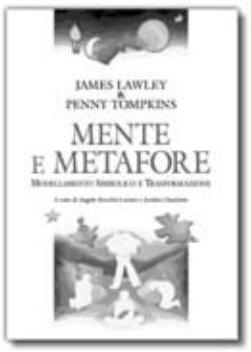
Altrimenti i client esistenti non funzioneranno più. Per farla breve: definite i messaggi esplicitamente!

Costa solo qualche linea di codice, ma può far risparmiare – a voi e ai vostri utenti – molto tempo in debugging futuro.

Bibliografia

[1] WSCF (WsContractFirst): http://www.thinktecture.com/WSCF

[2] Aaron Skonnard, *"Webmethod [Validation] and [Assert] Attributes"*, http://www.skonnard.com/articles/200.aspx



pagg. 311
ISBN 8881500124
€ 20.00

GRUPPO EDITORIALE
INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

MENTE E METAFORE

modellamento simbolico e trasformazione

James Lawley & Penny Tompkins

Cosa fate in qualità di terapisti, insegnanti, medici o manager quando i vostri clienti, studenti, pazienti o colleghi dicono "È come se stessi sbattendo la testa contro un muro " oppure "Ho un nodo allo stomaco" o "Sto cercando la via giusta da intraprendere"?
Mente e Metafore descrive come dare agli individui l'opportunità di scoprire come sono organizzate le proprie percezioni simboliche, cosa deve accadere perché queste possano cambiare, e come da questo, come risultato, essi possano trasformarsi.
Basato sull'approccio terapeutico pionieristico di David Grove e sull'uso del Clean Language, il Modellamento Simbolico è una forma emergente, sistemica e iterativa di facilitazione del processo psicoterapeutico. Questo libro parla in modo esaustivo della teoria della metafora, dei sistemi auto-organizzanti e del modellamento simbolico; della pratica del Clean Language; del Processo Terapeutico in Cinque Stadi; ed include tre trascrizioni di sedute con pazienti.

WEB: http://www.infomedia.it - e-mail: book@infomedia.it - tel: 0587/736460

La programmazione generica

La programmazione generica, finora riservata solo ai programmatori C++ o ai cultori di linguaggi di programmazione d'élite, come Eiffel, sta per essere resa disponibile anche agli sviluppatori C# e VB

di Lorenzo Vandoni

La programmazione generica costituisce un potente strumento di lavoro, che permette di risolvere in modo elegante ed efficace molti tipi di problemi. Tramite la programmazione generica diventa infatti possibile scrivere classi e funzioni generalizzate, che possano essere istanziate od eseguite con argomenti di tipi diversi. Per esempio, è possibile scrivere una classe "lista di X", dove X è un tipo qualsiasi, oppure una funzione di sort in grado di ordinare diversi tipi di elementi.

In generale, ciò che si può fare con la programmazione generica può essere implementato anche con strategie di tipo diverso. Questo è il motivo principale per cui molti linguaggi non la mettono a disposizione. Questo modo di lavorare, che sarà nuovo per molti, fornisce però alcune caratteristiche peculiari, che in alcuni casi possono portare a soluzioni migliori o più efficienti. Ma cerchiamo di capire anzitutto quali possono essere queste strategie alternative.

Le alternative alla programmazione generica

Una possibile alternativa alla programmazione generica consiste nell'uso delle interfacce. Per esempio, potremmo scrivere una classe *Lista* che permetta di gestire oggetti che implementino un'interfaccia *IListable*. Questa interfaccia dovrebbe definire i metodi necessari ad un oggetto *Lista* per gestire gli elementi mantenuti al suo interno, e dovrebbe essere implementata per ogni tipo di dato che si volesse utilizzare insieme con questa classe.

Questo modo di procedere però non permetterebbe di utilizzare l'oggetto *Lista* come contenitore di oggetti di tipi predefiniti, come stringhe o numeri, o che siano istanze di classi che non possono essere modificate, magari per l'impossibilità di accedere al loro codice sorgente.

Una seconda alternativa potrebbe essere quella di fare sì che tutti gli oggetti che fanno parte del linguaggio derivino da un'unica classe base *Object*. In questo caso si potranno creare classi lista come contenitori di *Object*, algoritmi che siano in grado di ordinare *Object*, e così via, con la sicurezza che queste classi e questi algoritmi possano essere utilizzati con tutti gli oggetti del linguaggio.

Questo è l'approccio adottato sia da Java che da .NET, almeno fino alla versione attuale, e anche da C++, fino all'introduzione dei template, sfruttando il "trucco" dei puntatori a *void*. Si tratta però di un'alternativa più debole rispetto alla programmazione generica, perché non permette di sfruttare eventuali caratteristiche più avanzate degli oggetti gestiti. La classe *Lista*, cioè, potrà usare solo i pochissimi metodi resi disponibili dalla classe *Object*.

Diventa così impossibile utilizzare questo approccio per creare classi generalizzate di tipo più avanzato, come ad esempio algoritmi di *sort* in grado di ordinare diversi tipi di collezioni. Questi algoritmi, infatti, avrebbero la necessità di sfruttare metodi specifici comuni a tutti i tipi di collezioni, ma non compresi nella semplice interfaccia della classe *Object*.

***Lorenzo Vandoni** è laureato in Informatica, ed è uno specialista di progettazione e sviluppo con tecniche e linguaggi object–oriented. Ha collaborato alla realizzazione di framework, strumenti di sviluppo e software commerciali in C++, Java e Visual Basic. Può essere contattato tramite e–mail all'indirizzovandoni@infomedia.it.*

Una strategia possibile potrebbe essere quindi quella di adottare le interfacce per i casi più complessi, come quello dell'algoritmo di sort, e sfruttare la classe *Object*. A meno di poter sfruttare, appunto, la programmazione generica.

La programmazione generica

La programmazione generica permette di creare funzioni e classi in grado di utilizzare oggetti di un generico tipo T, offrendo la possibilità di validarne la correttezza durante la compilazione. Questo significa che se una funzione *sort* richiede la disponibilità di un metodo *item*, il compilatore sarà in grado di avvisarci qualora questo metodo venga utilizzato per ordinare una collezione che non disponga di tale metodo. L'utilizzo di un elemento generico, come una funzione *sort* o una classe *Lista*, provoca, con modalità diverse da linguaggio a linguaggio, la creazione di un'istanza specifica di quell'elemento. Per esempio, se in punti diversi di uno stesso programma la classe *Lista* venisse utilizzata come contenitore di interi e di stringhe, il compilatore tratterebbe questi due casi come se fossero state definite due classi del tutto indipendenti tra loro. Questo provoca due importanti vantaggi:

- prima di tutto, non sarà possibile introdurre all'interno di una classe *Lista di interi* elementi di tipo diverso; questo tipo di controllo non viene garantito da nessuna delle soluzioni alternative precedentemente esaminate;

- inoltre, la conoscenza a priori del tipo degli elementi utilizzati (stringhe o interi, nell'esempio), evita al compilatore i controlli necessari per risalire al tipo dell'elemento; questi controlli, in presenza di ereditarietà e polimorfismo, si traducono in un notevole appesantimento delle performance; la programmazione generica è quindi in grado, nella maggior parte dei casi, di offrire prestazioni decisamente superiori.

Tra i linguaggi di uso comune, l'unico che supporta la programmazione generica è C++, tramite un costrutto denominato *template*. Anche .NET, però, a partire dalla sua versione 2.0, introdurrà la programmazione generica a livello di Intermediate Language (IL), e quindi permetterà a tutti i linguaggi di programmazione .NET di supportarla. I progettisti di .NET si sono affrettati a dire che la programmazione generica da loro implementata è molto diversa dai template C++, ma questo è vero solo in parte. Delle differenze ci sono, ma sicuramente i programmatori C++, o chi ha studiato Eiffel, non potranno non sentirsi a casa loro.

La programmazione generica in .NET

La definizione di una classe generica in Visual Basic.NET può essere scritta in questo modo:

```
Public Class Lista(Of T)
  Private moItem() As T

  Public Sub Add(ByVal oItem As T)
   ...
  End Sub
End Class
```

La parola chiave *Of* serve ad indicare un parametro di tipo generico, in questo caso rappresentato dal marcatore *T*. Tutte le volte in cui viene successivamente incontrato, questo marcatore indicherà un elemento dello stesso tipo. La classe conterrà cioè una variabile di istanza *moItem*, costituita da un array di elementi di tipo T, e una funzione *Add*, che riceverà un analogo argomento. Per usare questa classe si potrà scrivere:

```
Dim oLista1 As New Lista(Of Integer)
Dim oLista2 As New Lista(Of String)
```

La dichiarazione di questi due oggetti provocherà due diverse istanziazioni della classe generica, in cui il marcatore T verrà rispettivamente sostituito dai tipi *Integer* e *String*. In questo modo, una chiamata del tipo

```
oLista1.Add("abc")
```

provocherà un errore di compilazione. Il compilazione cioè sarà in grado di controllare che l'utilizzo di ognuno dei due oggetti sia conforme alla loro dichiarazione, evitando dolorosi errori a runtime.

Vincolare i tipi ammissibili

Il codice mostrato nell'esempio precedente permette di creare un tipo "lista di T" con T qualsiasi. Non viene posta cioè nessuna restrizione per il tipo T. Abbiamo visto, però, che in alcuni casi può essere importante imporre delle restrizioni. Una funzione generica di *sort*, per esempio, potrebbe lavorare con diversi tipi di collezioni, ma non riuscirebbe a gestire un parametro di tipo *Integer*. Per imporre dei vincoli di questo tipo si può scrivere:

```
Public Class GenericClass(Of T As Base)
```

dove *Base* è il nome di una classe o di un'interfaccia. Nel primo caso, il tipo *T* dovrà coincidere con la classe specificata o essere una sua sottoclasse; nel secondo caso, dovà implementare l'interfaccia specificata. Come si può vedere, queste due possibilità corrispondono grosso modo alle due alternative citate nel primo paragrafo, con il grosso vantaggio, però, di ottenere un controllo da parte del compilatore sui tipi effettivamente utilizzati. Questa soluzione risulta a mio parere anche un passo avanti rispetto a quella adottata dai template C++, che permettono di imporre tali vincoli solo indirettamente, cioè tramite l'utilizzo, all'interno

della classe generica, di specifici metodi che T dovrà fornire, senza però dichiararli esplicitamente. Un altro tipo di vincolo che si può imporre è quello per cui il tipo indicato come parametro debba disporre di un costruttore pubblico senza parametri. Questo vincolo può rendersi necessario qualora la classe generica abbia la necessità di creare istanze del tipo T, e può essere imposto scrivendo:

```
Public Class GenericClass(Of T As New)
```

Altre possibilità

Per concludere, vediamo una breve carrellata delle altre possibilità offerte da .NET relativamente alla programmazione generiche. Ho spesso citato come esempio, in questo articolo, la possibilità di creare funzioni generiche, come *sort*. Supponendo di volere creare una funzione di ordinamento per array di qualsiasi tipo, potremo ottenerla scrivendo:

```
Sub Sort(Of T)(ByVal oArr() As T)
...
End Sub
```

La funzione potrà essere poi richiamata in questo modo, supponendo che *oIntArray* sia un array di interi (l'indicazione del tipo *Of Integer* è opzionale, in quanto il compilatore è in grado di derivarla dal tipo dell'oggetto utilizzato come parametro):

```
Sort(Of Integer)(oIntArray)
```

Un altro tipo di possibilità che viene offerta da .NET è quella di dichiarare classi generiche parametrizzate con più di un tipo, come ad esempio:

```
Public Class GenericCollection(Of KeyType, ElementType)
```

L'utilizzo di queste classi è del tutto identico a quello delle classi generiche mostrate negli esempi precedenti, con la differenza che per ogni loro istanziazione occorrerà specificare il tipo di entrambi i parametri.

Infine, si può citare il fatto che .NET ammette, oltre alla definizione di classi generiche, anche la creazione di strutture, interfacce e delegate di tipo generico.

Conclusioni

Si può dibattere a lungo sui pro e contro della programmazione generica, vista in alcuni contesti come una contrapposizione all'approccio object-oriented.

Molto pragmaticamente, però, ritengo che debba essere considerato come uno strumento in più a disposizione del programmatore, che potrà decidere, a seconda dei casi, se utilizzarlo o meno.

Sempre meglio avere uno strumento in più, che uno in meno.


corsi
di formazione
ON-SITE
Java
.NET
PHP
Html
Visual Basic
per informazioni:
education@infomedia.it - 0587 73.64.60

Service Oriented Architecture

Analizziamo gli aspetti salienti della Service Oriented Architecture per gli architetti e gli sviluppatori del software.

di Paolo Pialorsi

I concetti alla base delle architetture distribuite dovrebbero ormai essere noti a tutti e sono utilizzati da diversi anni dagli architetti del software. Nel 1998 faceva la sua comparsa nel mondo delle architetture software lo sviluppo Windows DNA (Windows Distributed Network Architecture) i cui pilastri sono rappresentati nella **Figura 1**. Allora si pensava ad architetture distribuite basate su componenti, in particolare nel caso di Windows si parlava di componenti COM/DCOM, MTS e poi COM+. In realtà di COM+ si parla ancora molto anche oggi e trovo che sia giusto così :-)!

Componenti: guardiamoci alle spalle

Volendo guardare indietro per un attimo, proviamo a valutare pregi e difetti delle applicazioni basate su componenti. Una delle ragioni che ci ha spinto a sviluppare dei componenti è stato sicuramente il fatto di poter riutilizzare delle porzioni di codice.

I componenti hanno infatti il vantaggio di poter essere condivisi da N diverse applicazioni. Pensiamo ai classici componenti COM installati sul client una volta, configurati nel registro di sistema con il mitico REGSVR32 e poi utilizzati da numerose applicazioni, scritte per esempio con Visual Basic 6. I componenti possono inoltre essere installati su server dedicati, detti application server, per essere poi utilizzati da N PC client via DCOM, MTS e oggi COM+, consentendoci di avere una maggiore scalabilità, una più comoda gestione e contesti di sicurezza differenziati a seconda dello strato applicativo.

Vista così sembrano tutte "rose e fiori" e verrebbe spontaneo chiedersi perchè allora qualcuno stia pensando a delle architetture nuove e parzialmente diverse dalla filosofia dei componenti.

Proviamo a pensare al primo problema che emerge quando si sviluppano applicazioni basate su componenti. Se avete mai sviluppato applicazioni di questo tipo, scommetto che avete pensato al versioning!

Un tipico problema dello sviluppo basato sui componenti, in particolare sui componenti COM nei vari "gusti" (DCOM, MTS, COM+), è la gestione delle versioni. A livello teorico doveva essere tutto semplice e rapido, a livello pratico sappiamo che una nuova release di un componente COM significava lacrime e sangue. Per non parlare del fatto che in molti ancora oggi fanno confusione relativamente alla compatibilità binaria e alla compatibilità di progetto, nello sviluppo di ActiveX Objects con Visual Basic 6. Quest'ultimo è più un problema di informazione che non di tecnologia, ma rimane il fatto che spesso è faticoso gestire in modo comodo il versioning di componenti.

Un'altra caratteristica dei componenti, che diventa un loro limite, è il fatto che generalmente sono utilizzati da client che li conoscono molto bene e a loro volta utilizzano componenti che conoscono altrettanto bene. Questo ci garantisce il corretto funzionamento delle nostre applicazioni, ma ci rende anche enormemente vincolati. Inoltre i componenti condivisi con degli application server sono di solito utilizzabili solo all'interno di

devleap

***Paolo Pialorsi** è un consulente e autore specializzato nello sviluppo di Web Service e soluzioni Web con il Framework .NET di Microsoft. Lavora nell'omonima società Pialorsi Sistemi S.r.l. e fa parte del gruppo DevLeap. Può essere contattato via email: paolo@devleap.it.*

una rete locale o di una Virtual Private Network, in quanto richiedono di utilizzare servizi e protocolli difficilmente disponibili su una connessione Internet generica (ADSL, PSTN, ecc.).

Infine le tecnologie a nostra disposizione per lo sviluppo di componenti generalmente non sono state pensate per essere interoperabili con altre piattaforme. Si pensi a COM/DCOM, CORBA, .NET Remoting, ecc.

Servizi: guardando al domani

Per le ragioni appena viste sono ormai alcuni anni che si investe nella ricerca e sviluppo di Web Service - da qui in poi preferisco chiamarli servizi SOAP - spesso visti come alternativa ai componenti.

I servizi SOAP sono infatti utilizzabili da diversi client, anche remoti. Non richiedono costi o attività di installazione sul client. Tipicamente i servizi non condividono informazioni di stato, oggetti e strutture dati con i client SOAP che li utilizzano.

Sul "cavo" passa solo SOAP, cioè XML, poi è compito di ciascun nodo SOAP interpretare i messaggi XML e ricostruire oggetti e strutture dati particolari. Dal punto di vista dell'utilizzatore di un servizio, possiamo tranquillamente ignorare come e con quale linguaggio o piattaforma sia stato scritto il servizio stesso. Se ben disegnati i servizi SOAP sono inoltre in grado di rendere trasparenti eventuali aggiornamenti di versione, senza quindi costringerci a repentini aggiornamenti e ricompilazioni dei client, nè tantomeno richiedendo dei tempi di fermo macchina per l'aggiornamento.

Infine i servizi SOAP sono nati per fornire interoperabilità tra le piattaforme. Oggi questo requisito è più che mai vero e realizzato, non gratis per carità, ma pagando il giusto prezzo in termini di disegno e sviluppo, si ottengono risultati significativi nella direzione della interoperabilità pura tra piattaforme (.NET, Java, PHP, Delphi, ecc.).

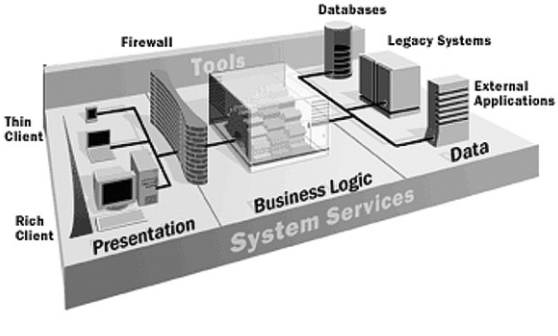


Figura 1 Architettura Windows DNA

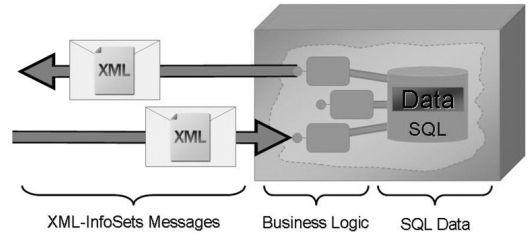


Figura 2 Schema pratico di un servizio in architettura SOA

SOA

I requisiti delle applicazioni moderne richiedono soluzioni interoperabili e scalabili, che siano in grado di aggregare dati provenienti da fonti dati differenti, con strutture dati anche completamente diverse.

Spesso è un requisito irrinunciabile la possibilità di utilizzare Internet - via HTTP - per dialogare e non una LAN o una VPN ad hoc.

Infine a causa della rapidità con la quale oggi si evolvono i mercati e gli scenari, i requisiti di un progetto possono cambiare diverse volte durante le fasi del disegno, dello sviluppo e della messa in produzione. La Service Oriented Architecture (SOA) tenta di dare delle risposte proprio a questo tipo di esigenze.

L'idea alla base di SOA è quella di mettere in comunicazione porzioni autonome di software che espongono servizi, in modo indipendente dal protocollo di trasporto, dalla piattaforma e dall'architettura.

SOA Tenets

Quando si parla di Service Oriented Architecture si fa spesso riferimento ai SOA Tenets (che in inglese non letterale può essere interpretato come "Principi di SOA"). Sono solo quattro, ma riassumono tutte le caratteristiche salienti di SOA. Vediamoli insieme:

1. **Boundaries are explicit**: i confini dei servizi devono essere espliciti. Significa che dobbiamo sapere esattamente quali sono i confini dei nostri servizi, intendendo per confini i punti di contatto con l'esterno. Tutto ciò che è all'interno di questi confini deve essere assolutamente ignoto agli utilizzatori del servizio stesso. Linguaggi utilizzati, architetture interne, eventuali motori di database sul backend, ecc. sono tutte informazioni che non devono interessare a chi usufruirà del servizio.

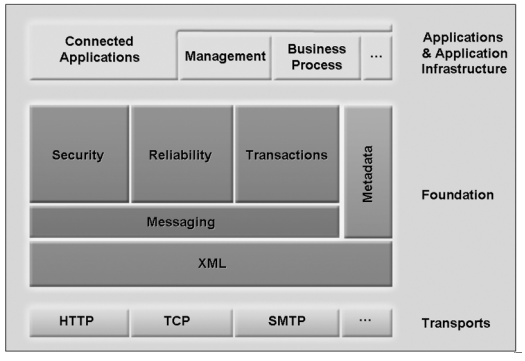


Figura 3 Web Services Architecture

2. **Services are autonomous**: i servizi possono e devono dialogare tra loro, ma singolarmente sono autonomi e autosufficienti. Non possono esistere servizi che funzionano solo insieme ad altri. Non devono esistere servizi che si fidano degli altri servizi. Ogni servizio deve validare e verificare gli input che riceve. In caso di fallimento di un servizio, altri servizi che lo utilizzino non devono esserne condizionati.

3. **Services share schema and contract, not class**: le informazioni che i servizi condividono per dialogare sono solo e unicamente quelle visibili all'esterno dei loro confini, come gli schema XSD e i contratti WSDL. Se due servizi che devono dialogare hanno bisogno di condividere delle classi o delle DLL, non sono autonomi e non sono SOA.

4. **Compability is based upon policy**: i criteri di compatibilità sono descritti, scambiati e verificabili con delle regole (policy) comprensibili a tutti, indipendenti cioè dalle piattaforme e dai linguaggi.

Come si vede chiaramente queste regole non sono applicabili solo ai Web Service, anzi in alcuni casi esistono dei Web Service che sono tutto tranne che Service Oriented. Può sembrare una contraddizione, ma non lo è affatto. Pensiamo ai classici Web Service ASMX che restituiscono elenchi di record sotto forma di DataSet. Sono dei Web Service, non vi è dubbio, ma presuppongono che sia il servizio che il client conoscano il concetto di DataSet, che sia il client che il servizio siano .NET. Non sono autonomi e non sono dotati di confini espliciti. Sono Web Service ma non sono SOA. Questo significa che non dobbiamo e/o non possiamo sviluppare servizi .NET per la gestione di record, i classici CRUD (Create, Read, Update, Delete) perchè non sono SOA? Assolutamente no!

Dobbiamo solo essere consapevoli del fatto che se vogliamo sviluppare architetture SOA, i Web Service CRUD non sono adatti. Se vogliamo disegnare architetture .NET distribuite e disaccoppiate, ma senza l'esigenza di essere SOA, allora i Web Service CRUD si possono utilizzare e funzionano senza problemi. In altre parole: non tutti i Web Service devono essere SOA; SOA non significa sempre e solo Web Service. Proviamo a dare una spiegazione ad alcuni dei concetti chiave delle architetture SOA.

Disegno dei messaggi

Per ottenere i risultati appena illustrati dobbiamo probabilmente cambiare il nostro modo di ragionare. Proviamo a pensare alle entità di business presenti nelle nostre applicazioni, a prescindere da come le renderemo disponibili ai nostri interlocutori. Generalmente possono cambiare le operazioni da svolgere e la logica di funzionamento delle singole operazioni, al cambiare dei requisiti, ma non cambieranno mai completamente le entità in quanto tali. Se dobbiamo disegnare un'architettura SOA per la gestione dei clienti, potranno cambiare i requisiti funzionali, ma non cambierà il concetto di cliente. Al massimo potremo avere diverse versioni (release) del concetto di cliente. Potremo inoltre avere diverse applicazioni che lavorano con le stesse entità di business, magari presentando ciascuna qualche piccola specializzazione del concetto di cliente. Conviene descrivere sotto forma di schema, ad oggi tramite XSD, l'entità di business più corretta possibile e più astratta possibile dalle singole operazioni. Questo per evitare che siano necessarie N diverse rappresentazioni dello stesso concetto, nel nostro caso il cliente. Ovviamente la ricerca di una versione "standard" di cliente comporta la scelta di quali sono le vere informazioni salienti per descrivere un cliente e quali invece possono essere sacrificate nel passaggio da un servizio all'altro.

> I confini dei servizi devono essere espliciti

All'esterno ci presenteremo con questa versione di cliente, internamente alla nostra architettura dovremo farci carico di associare le singole rappresentazioni interne con la rappresentazione esterna. Ad esempio se i nostri componenti di accesso ai dati (Data Access Layer) restituiscono un DataSet ADO.NET, con un DataTable di clien-

ti, l'elenco di clienti che restituiremo a chi utilizza i nostri servizi sarà una lista di entità cliente, convertite dalle singole DataRow al concetto più astratto di entità di business di tipo Cliente. Immaginiamo che si tratti del seguente schema:

XML (XSD)

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema targetNamespace="http://schemas.paolo.com/Customer"
  elementFormDefault="qualified"
  xmlns="http://schemas.paolo.com/Customer"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="Customer">
  <xsd:complexType>
    <xsd:attribute name="name" type="xsd:string"
use="required" />
    <xsd:attribute name="email" type="xsd:string"
use="optional" />
    <xsd:attribute name="id" type="xsd:int" use="required" />
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Nel disegnare i messaggi dovremo consentire ai servizi di sopravvivere ai cambiamenti. Se per esempio il nostro cliente dovesse avere in futuro bisogno di un'informazione aggiuntiva, come il numero di telefono, uno schema come quello illustrato in precedenza risulterebbe troppo rigido.

Una versione più tollerante ai cambiamenti potrebbe essere la seguente:

XML (XSD)

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema targetNamespace="http://schemas.paolo.com/Customer"
  elementFormDefault="qualified"
  xmlns="http://schemas.paolo.com/Customer"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="Customer">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:any processContents="lax" minOccurs="0"
                                    maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string"
                                          use="required" />
    <xsd:attribute name="email" type="xsd:string"
                                          use="optional" />
    <xsd:attribute name="id" type="xsd:int" use="required" />
    <xsd:anyAttribute processContents="lax" />
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Il fatto di prevedere espansioni future, attraverso l'aggiunta di attributi ignoti a priori (anyAttribute) o elementi altrettanto ignoti a priori (any) ci permette di rendere sempre comprensibile il concetto di cliente, anche quando dovessimo estenderne la struttura.

Nel disegnare i messaggi dovremo consentire ai servizi di sopravvivere ai cambiamenti

L'attributo processContents valorizzato come lax permette di dire a un validatore XSD di messaggi che gli attributi o gli elementi aggiuntivi dovranno essere validati solo in caso di presenza di uno schema, altrimenti saranno ignorati, seppur presenti nel messaggio.

Di solito nel parlare di messaggi XML si fa riferimento al concetto di XML Infoset [3] e non di XML 1.0, per sottolineare il fatto che non conta tanto che si tratti di file testuali XML, ma piuttosto che le informazioni siano organizzate in alberi di nodi navigabili.

Contract First, Idempotenza e Reliable Messaging

I contratti dovranno definire le operazioni che vogliamo esporre, basate sui messaggi appena disegnati. Quando si parla di SOA si fa spesso riferimento al concetto di "contract first", che rappresenta l'idea di disegnare prima i messaggi e le operazioni, sotto forma di XSD e WSDL, per poi implementarle solo in un secondo momento. In passato, nello sviluppo di componenti, si disegnavano le interfacce per poi implementarle con delle apposite classi. Oggi si disegnano i contratti (WSDL) e i messaggi (XSD) e li si implementa tanto dal lato del provider (servizio) che del consumer (client) rispettando fedelmente i contratti. Questo approccio ci garantisce la massima interoperabilità, oltre che una vera indipendenza dalla piattaforma.

Non a caso esistono e stanno nascendo dei tool per disegnare i WSDL, indipendentemente dal framework di sviluppo. Esistono poi dei tool e degli Add-In per Visual Studio .NET che generano il codice del provider e del consumer a partire da un WSDL astratto [4].

Nel disegno dei contratti dovremo pensare alle operazioni e non alle funzioni di un oggetto di business. Come vedremo più avanti, ovviamente alle spalle di queste ope-

razioni vi saranno degli oggetti con dei metodi, ma all'esterno questo non dovrà essere significativo.

Un altro aspetto fondamentale delle architetture SOA è il fatto che a volte è necessario costruire dei dialoghi basati su messaggi e non delle semplici interazioni mono-operazione. Affinchè sia possibile costruire dei dialoghi abbiamo bisogno di essere in grado di identificare i singoli messaggi e le singole fasi del dialogo.

Dobbiamo prevedere la possibilità di inviare più volte uno stesso messaggio, qualora non venisse ricevuto dal destinatario. Inoltre si parla di idempotenza dei messaggi, riferendosi al fatto che uno stesso messaggio, qualora venga ripetuto, deve fornire lo stesso risultato funzionale. La ripetizione può essere proprio dovuta al fatto che uno dei nodi in gioco, durante un dialogo, rimanda un messaggio che crede non sia stato ricevuto. In questo caso il destinatario del messaggio deve essere in grado di gestire la situazione senza creare inutili disagi al nodo chiamante.

> Disegnare i contratti e i messaggi per poi implementarli tanto dal lato del provider che del consumer

Data e Business layer

Rimane da chiarire come ci si debba muovere all'interno dei confini dei nostri servizi. Da un punto di vista implementativo dovremo pur sempre occuparci di definire la logica di funzionamento, perchè per presentarci all'esterno con determinate operazioni, basate su entità definite, ci vuole qualcosa che sia in grado di elaborare i messaggi e produrre le singole istanze di entità. Questo significa che i componenti di business e lo strato di accesso ai dati continueranno ad esistere. Inoltre avremo bisogno di persistere una serie di informazioni all'interno di un database o di un repository permanente.

Possiamo allora pensare allo schema riportato in **Figura 2** dove si vede chiaramente che i componenti di business continuano a esistere, che si appoggiano su strati di accesso ai dati, che a loro volta interrogano i classici database relazionali.

La differenza è data dal fatto che la presentazione di queste entità di business passa attraverso servizi che rappresentano i messaggi sotto forma di XML Infoset.

In un certo senso i servizi diventano uno strato di presentazione delle operazioni e delle entità. Questo modo di ragionare è sicuramente vincente perchè ci permette di variare rapidamente le tecniche di comunicazione tra i provider e i consumer, senza dover necessariamente rivedere la logica di business o il data access layer.

Viceversa quando decideremo di estendere le funzionalità dei servizi o il contenuto informativo delle entità, saremo in grado di farlo sfruttando il fatto di aver disegnato i messaggi pensando al versioning.

Web Services Architecture e Microsoft roadmap

Le più grandi aziende del settore informatico (Microsoft, IBM, SAP, BEA, Verisign, RSA, Tibco, ecc.) stanno investendo pesantemente nella ricerca e sviluppo di servizi SOAP e di protocolli infrastrutturali per realizzare architetture SOA complete. Web Services Architecture (WSA) è un insieme di specifiche - elaborate e rese pubbliche da queste aziende - pensate per realizzare in modo stabile e interoperabile servizi SOA. Alcune di queste specifiche sono già definitive, altre sono in via di definizione. Come si vede in **Figura 3** la famiglia di specifiche WSA prevede aspetti relativi al Reliable Messaging, per descrivere e realizzare dialoghi, alla sicurezza, la transazionalità, la descrizione, lo scambio dei messaggi, ecc.

Se la direzione presa dal mercato rimarrà questa, e personalmente ne sono convinto, da qui a pochi anni assisteremo ad un proliferare di servizi SOAP in architettura SOA, nonché di framework per il loro sviluppo, pensati per supportare le specifiche descritte da WSA, man mano che diventeranno definitive.

Dal punto di vista di Microsoft un grosso salto in avanti sarà segnato dall'avvento di .NET 2.0 e di Visual Studio 2005. In Visual Studio 2005 saranno infatti presenti degli strumenti per disegnare e implementare i servizi in architettura SOA. Il vero salto di qualità per Microsoft nel mondo SOA, sarà però determinato dal rilascio di Indigo, l'infrastruttura di comunicazione che sarà resa disponibile indicativamente fra un anno e mezzo circa e che sarà intermedia tra il rilascio di .NET 2.0 e la prossima versione di Windows, ad oggi chiamato Longhorn. Indigo sarà un framework di comunicazione completo, pensato per sviluppare servizi SOA. Prevederà funzionalità di sicurezza, reliable messaging, transazionalità, ecc. applicabili a servizi in grado di esporre operazioni basate su messaggi XML Infoset.

Bibliografia

[1] Jeffrey Hasan - *"Expert Service-Oriented Architecture in C#: Using the Web Services Enhancements 2.0"*, APress, 2004

Riferimenti

[2] http://msdn.microsoft.com/architecture/soa/
[3] http://www.w3.org/TR/xml-infoset/
[4] http://www.thinktecture.com/Resources/Software/WSContractFirst/

euro 7,50

36 racconti
per stuzzicare
la tua *abilità*
matematica

I rompicapo del Doktor Morb

pagg. 80
ISBN 8881500132
€ 7.50

WEB: http://book.infomedia.it
e-mail:book@infomedia.it
Tel. 0587/736460

GRUPPO EDITORIALE
INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

a cura di Davide Mauri
dmauri@programmers.net

Kevin Gearing's FormShield

Chi cerca di promuovere i propri "servizi" tramite l'utilizzo di *spam* inventa ogni giorno meccanismi sempre più fini e ingegnosi per poter lasciare il proprio messaggio ovunque.

Non sono più al sicuro neanche quei siti dove per poter entrare è necessario riempire un form di autenticazione perché ormai i *"bot"* sono abbastanza sofisticati da potersi registrare in automatico decine, centinaia o migliaia di volte, utilizzando dati verosimili, camuffandosi da utenti umani ed ingannando quindi i semplici controlli eventualmente presenti.

I suddetti programmi, conosciuti in gergo come *bot*, possono essere causa di non pochi problemi.

L'ultimo attacco effettuato dagli *spambot* – che come dice il termine sono *bot* che portano con sè messaggi spam – è relativo ai blog, in particolare alla sezione dedicata ai commenti dei post.

Il *bot* riempe il form di commento come se fosse un utente vero e ne fa il submit; ovviamente il messaggio inviato è tutt'altro che interessante e quasi sempre pubblicizza prodotti più o meno "seri" (per non utilizzare altre parole probabilmente più adatte…).

Il risultato è che diversi blog hanno deciso di eliminare la possibilità di lasciare commenti, diminuendo così drasticamente le potenzialità della community, e che i vari proprietari dei rispettivi blog hanno devono cancellare *a mano* i vari commenti non desiderati.

Un bel fastidio…e a volte anche un danno economico, in particolare per le società di hosting e per i loro clienti che si vedono la loro banda ed il loro spazio sottratto per scopi illeciti ed inutili.

I problemi non si fermano qui però. Un altro strumento utile e fortemente attaccabile dai *bot* sono i questionari. Volete essere sicuri che la vostra opinione sia quella della maggioranza?

Facile, basta scrivere un *bot* che per voi continuerà a effettuare la votazione, variando continuamente il nome dell'utente ed i suoi parametri, in modo da simulare la presenza di un essere umano ed non di un meccanismo automatico, così da far levitare la vostra preferenza.

Il problema non è quindi da ignorare ed ormai in molti siti (soprattutto le community) utilizzano un controllo particolare per essere sicuri che l'entità che sta compilando il form sia un essere umano e non una macchina.

Come fare? Per ora ci si può basare sul fatto che le macchine, benché estremamente potenti e veloci mancano di fantasia e di immaginazione. In particolare un uomo riesce piuttosto facilmente a leggere un testo distorto mentre un sistema automatico ha – oggi – molti più problemi.

Ecco quindi l'idea: oltre ai dati di registrazione è sufficiente far inserire all'utente il codice che vede scritto in modo distorto ed alterato all'interno di un'immagine per chiudere immediatamente la porta a tutti i *bot* presenti nella rete.

Tale tecnica prende il nome di *Human Interaction Proof (HIP)* ed un esempio è visibile in **Figura 1**: l'immagine è

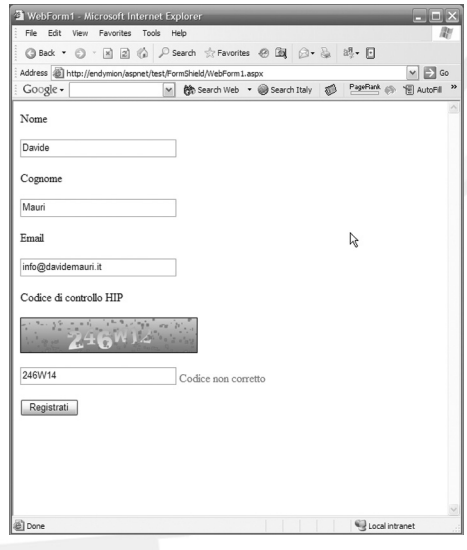


Figura 1 Le immagini generate dal controllo per autenticare la presenza di una persona fisica e non di un bot

Davide Mauri *è un consulente freelance che si occupa di SQL Server 2000 e di sviluppo di soluzioni Web basate sul .NET Framework. All'attività di consulenza affianca una cospicua attività di docenza e di formazione presso Mondadori Informatica. Il suo sito personale è www.davidemauri.it.*

generata tramite il controllo FormShield, aderendo alle indicazioni del progetto *CAPTCHA™*.

Il controllo è totalmente gratuito anche per uso commerciale, ma purtroppo non è distribuito con il codice sorgente (ho contattato l'autore direttamente e mi ha detto che sta pensando di rilasciarlo in un prossimo futuro).

Poco male però, per fortuna è sviluppato davvero molto bene, è estremamente configurabile ed è distribuito come Custom Control quindi lo si può utilizzare piazzandolo direttamente nella toolbar di Visual Studio.

Insieme al controllo è distribuita anche la documentazione, piuttosto completa ed utile a capire il significato delle numerosissime opzioni configurabili, riportate di seguito:

1. Generazione di immagini BMP, PNG, GIF, JPG, TIFF
2. Impostazione delle dimensioni dell'immagine del testo alternativo
3. Immagine composta da 3 layer, configurabili in modo indipendente: Background, Foreground, Noise
4. Più di 60 stili combinabili per generare l'immagine
5. Supporto per la scelta dei caratteri da utilizzare (numerici, alfabetici, simboli, oppure una loro combinazione)
6. Supporto per la scelta dei font da utilizzare (anche più font contemporaneamente)
7. Più di 20 effetti impostabili sul testo, per avere un buon compromesso tra leggibilità e protezione
8. Creazione totalmente casuale utilizzando gli effetti selezionati
9. 6 tipologie di immagini già preselezionate e pronte ad essere usate
10. Le immagini sono generate sempre in real-time, non vengono salvate su disco
11. Criptazione della querystring per evitare che il codice possa essere intercettato
12. Integrabile con i Validator

L'ultimo punto è particolarmente utile, in quanto rende estremamente banale l'utilizzo del controllo all'interno di un form di registrazione; per utilizzare il controllo sono necessari pochi semplici passi:

1. Piazzare sul form il controllo FormShield
2. Aggiungere una TextBox
3. Aggiungere un CompareValidator
4. Specificare la TextBox creata al punto 2 come proprietà ControlToCompare del CompareValidator
5. Specificare il controllo FormShield creato al punto 1 come proprietà ControlToValidate del CompareValidator

Ed il gioco è fatto!

In conclusione un controllo molto utile e molto semplice da utilizzare, consigliato a chiunque abbia la necessità di assicurarsi che gli utenti registrati al sito siano persone non *bot*, in modo da evitare spam.

Prodotto
Kevin Gearing's FormShield

Url di riferimento
http://dotnetfreak.co.uk/blog/archive/2004/11/06/166.aspx
http://www.captcha.net/
http://msdn.microsoft.com/asp.net/using/building/web/default.aspx?pull=/library/en-us/dnaspp/html/hip_aspnet.asp

Stato Release
1.0.1

Semplicità d'uso ✦✦✦✦✦
Semplicissimo! Lo si inserisce nella toolbox e lo si trascina sulla pagina. Finito.

Utilità ✦✦✦✦✦
Lo spam è un problema, in qualsiasi sua forma. Un controllo per aiutare a combatterlo non può che essere utile.

Qualità prodotto ✦✦✦✦✧
Molto buona.

Qualità documentazione ✦✦✦✦✧
Le due righe che spiegano come utilizzare il controllo sono disponibili solo sul blog (ed ora anche su VBJ), mentre è disponbile il file di help offline che spiega la sintassi delle API.

ExcelXmlWriter

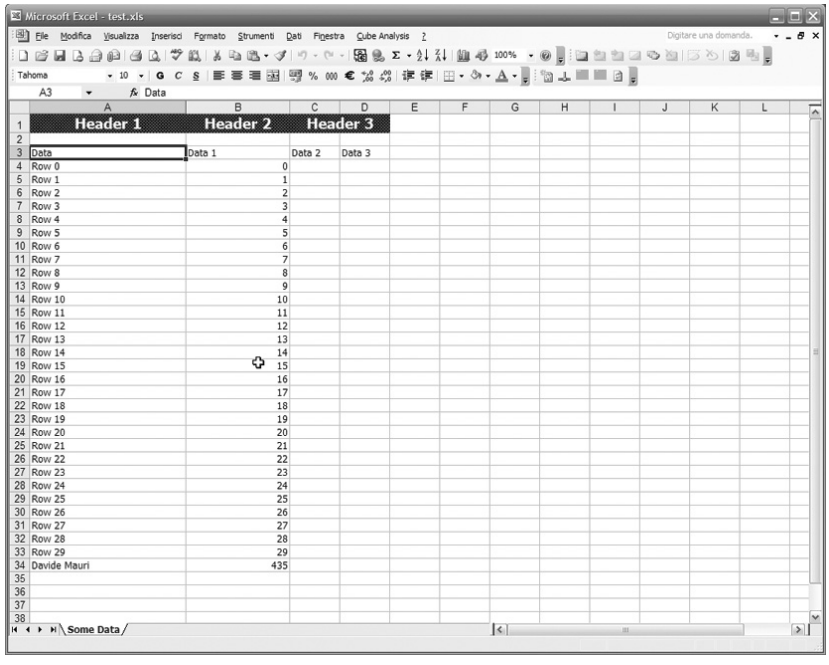
Esportare dati in formato Excel è un'operazione che il 99% degli sviluppatori si è trovata a dover fare.

I modi per poter ottenere il risultato sono molteplici, quelli che però non richiedono la presenza di Excel installato sulla macchina sulla quale deve avvenire l'esportazione sono forniti solo da società terze, che, com'è giustamente prevedibile, richiedono l'acquisto di una licenza d'uso del proprio prodotto.

Non sempre però tale spesa è giustificabile, in quanto magari solo una piccola parte delle funzionalità messe a disposizione è utilizzata, oppure perché il tool che si sta sviluppando è solo un strumento di uso temporaneo sulla quale non si vogliono investire troppe risorse.

Bene, in questi ed in tutti gli altri casi, ExcelXmlWriter può essere di notevole aiuto in quanto permette di creare fogli di lavoro Excel in modo totalmente autonomo, senza dover per forza di cose passare tramite un driver ODBC oppure tramite il classico CreateObject(...). In questo modo l'applicazione risulta essere molto più leggera, così come il relativo stress a carico del server.

Il componente è scritto interamente in C# (anche in questo caso, però, non viene distribuito il codice sorgente), e produce un file .xls in formato XML (ecco spiegato il perché del nome di questo componente).



	Header 1	Header 2	Header 3
3	Data	Data 1	Data 2	Data 3
4	Row 0	0		
5	Row 1	1		
6	Row 2	2		
7	Row 3	3		
8	Row 4	4		
9	Row 5	5		
10	Row 6	6		
11	Row 7	7		
12	Row 8	8		
13	Row 9	9		
14	Row 10	10		
15	Row 11	11		
16	Row 12	12		
17	Row 13	13		
18	Row 14	14		
19	Row 15	15		
20	Row 16	16		
21	Row 17	17		
22	Row 18	18		
23	Row 19	19		
24	Row 20	20		
25	Row 21	21		
26	Row 22	22		
27	Row 23	23		
28	Row 24	24		
29	Row 25	25		
30	Row 26	26		
31	Row 27	27		
32	Row 28	28		
33	Row 29	29		
34	Davide Mauri	435		

Figura 2 Esempio di foglio excel generato dal ExcelXmlWriter

Questo significa che tale file è leggibile solamente dalle versioni XP e 2003 (testata personalmente su quest'ultimo) e quindi l'utilità generale né viene un po' sminuita, ma se non avete la necessità di mantenere una compatibilità con le versioni precedenti di Office, questo piccolo componente fa un eccellente lavoro.

Un esempio di come è possibile generare un file Excel è visibile di seguito:

```
using CarlosAg.ExcelXmlWriter;

class TestApp {
    static void Main(string[] args) {
        Workbook book = new Workbook();0
        Worksheet sheet = book.Worksheets.Add("Sample");
        WorksheetRow row =  sheet.Table.Rows.Add();
        row.Cells.Add("Hello World");
        book.Save(@"c:\test.xls");
    }
}
```

Come si nota è stato implementato un piccolo Object Model che permette di gestire una pagina di lavoro di Excel, gestendo righe, colonne, pivot, attributi estetici, formule e via dicendo.

Insieme alla libreria viene fornito anche un file di help che mostra tutte le classi, i metodo e le proprietà disponibili (in modo non troppo approfondito in effetti, ma per fortuna i nomi sono generalmente autoesplicativi); oltre a questo ci sono anche due piccoli esempi che mostrano come creare una semplicissima applicazione che sfrutta la libreria e come sfruttare anche funzionalità più avanzate in modo da creare fogli Excel personalizzati secondo le proprie esigenze (**Figura 2**).

Un'ultima nota importante è che la libreria – data la sua natura molto semplice – non supporta la generazione di grafici: è stata pensata infatti non per far concorrenza a soluzioni molto più belle e complesse (e costose), ma per sopperire alla mancanza di un piccolo ed efficiente tool per esportare dati verso Excel in modo semplice e veloce.

Prodotto
ExcelXmlWriter

Url di riferimento
http://www.carlosag.net/

Stato Release
1.0.3

Semplicità d'uso ✦✦✦✦✦
Molto semplice.

Utilità ✦✦✦✧✧
L'esportazione verso un programma come Excel è sempre utilissima. Il punteggio è relativamente basso perché è compatibile solo con le versioni XP e superiori.

Qualità prodotto ✦✦✦✦✧
Molto buona

Qualità documentazione ✦✦✦✧✧
Più che sufficiente, fa quello che deve senza troppi fronzoli.

Applicazioni Web con ASP .NET

Il Web è ormai diventato una vera e propria piattaforma per lo sviluppo di applicazioni. In questo ambito ASP.NET si propone come una valida tecnologia per la realizzazione di applicazioni Web professionali ed affidabili, rappresentando un significativo passo avanti rispetto alla tecnologia Active Server Pages. "Applicazioni Web con ASP.NET" illustra le caratteristiche di questa nuova tecnologia descrivendo, con un approccio orientato alla pratica, gli elementi messi a disposizione dello sviluppatore: dall'uso dei controlli server all'inter-facciamento con ADO.NET per l'accesso a database, dalla creazione di controlli personalizzati alla configurazione e all'ottimizzazione delle applicazioni, dalla gestione della sicurezza alla creazione di servizi Web, dal debugging alla migrazione da ASP. Il tutto è corredato da numerosi esempi per fissare i concetti ed una serie di appendici di riferimento per aiutare lo sviluppatore nella realizzazione delle proprie applicazioni.

Applicazioni Web con ASP .NET
Andrea Chiarelli
pagg. 330
ISBN 8881500116
€ 39.00

dal SOMMARIO

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

technical BOOK

Clicca su www.infomedia.it per leggere il capitolo d'esempio

WEB: http://book.infomedia.it - e-mail: book@infomedia.it - Tel: 0587/736460

Gestire il ciclo di vita dei dati - prima di annegarci dentro

di Jim Lee

I dati sono alla base di ogni organizzazione e man mano che le aziende ne accumulano crescenti volumi, la loro gestione e memorizzazione diviene ben presto uno dei compiti più critici del settore IT. I responsabili IT ricercano metodi sempre più efficaci per migliorarne la gestione e lo storage, pur riducendo i costi per massimizzare il proprio investimento.

Di conseguenza, lo Storage Resource Management (SRM) sta assumendo rapidamente una importante posizione di mercato nel supportare le aziende che vogliono conseguire questi obiettivi.

Man mano che la quantità e il valore dei dati aziendali aumenta, man mano che gli ambienti di calcolo diventano più complessi e i costi di gestione dello storage salgono alle stelle, l'importanza dell'SRM aumenta in modo esponenziale.

L'esplosione dei dati ha causato un crescente fabbisogno di enterprise storage, creando la necessità per le aziende di disporre di una varietà di tecnologie di storage tra le quali le Storage Area Network (SAN), le Networked Attached Storage (NAS), il Hierarchical Storage Management (HSM) e il Direct Attached Storage.

Considerando l'estensione dello storage aziendale, è necessaria una soluzione SRM esauriente per fornire una vista globale dei dati e delle risorse di storage di un'azienda, compreso il monitoraggio dello stato, la gestione efficace delle risorse di storage, assicurando la disponibilità e garantendo l'evoluzione futura.

Oggigiorno, l'Active Archiving è riconosciuta come strategia collaudata ed efficace per gestire complessi database relazionali a rapida crescita, controllando nel contempo l'eccessiva crescita del database a lungo termine.

L'Active Archiving opera in un framework di varie tecnologie di storage e di SRM per offrire un approccio 'best practice' per gestire risorse di storage e ridurre i costi di gestione. L'integrazione dell'Active Archiving con l'SRM garantisce alle aziende la capacità di affrontare efficacemente il compito di gestire crescenti volumi di dati.

La giusta combinazione e adozione di queste tecnologie fa sì che le aziende possano affrontare le proprie necessità di gestione dati, di conservazione dei dati e di storage al costo più basso.

Gestire i dati enterprise per l'intero ciclo di vita

Un'appropriata gestione dei dati e dello storage prende atto che i dati hanno un proprio ciclo di vita. Tipicamente il ciclo di vita dei dati ha inizio

Jim Lee *è Vice Presidente, Product Marketing della Princeton Softech (www.princetonsoftech.com). Princeton Softech è espositore alla Storage Expo, il più grande evento in UK dedicato allo storage, che si tiene a Olympia Londra. (www.storage-expo.com)*

con una necessità di tipo business, dapprima acquisendoli e in seguito referenziandoli regolarmente durante l'attività lavorativa. Col tempo, questi dati perdono vitalità e l'accesso è meno frequente, perdendo gradualmente di importanza nel business, ed infine finiscono per essere gettati via. Tuttavia, per gran parte del ciclo di vita, questi dati vengono conservati online.

Il semplice, ma critico principio che tutti i dati attraversano gli stadi del ciclo di vita è la chiave per migliorare la gestione dati. Comprendendo come vengono utilizzati i dati e quanto devono essere conservati, le aziende possono sviluppare una strategia per mappare pattern di utilizzo e supporto di memorizzazione ottimale, minimizzando quindi il costo totale di storage dei dati in tutto il ciclo di vita.

Gli stessi principi si applicano quando i dati vengono memorizzati in un database relazionale; tuttavia, il compito di gestire e memorizzare i dati relazionali è complesso a causa delle complessità intrinseche nelle relazioni tra i dati. I database relazionali rappresentano un principale consumatore di storage e sono anche fra i più difficili da gestire poiché ad essi si accede con regolarità. Senza la possibilità di gestire efficacemente i dati relazionali, relativamente alle necessità di utilizzo e di storage, la sfuggente crescita del database si tradurrebbe in un aumento dei costi operativi, in prestazioni scadenti e in una limitata disponibilità delle applicazioni che si basano su questi database.

La soluzione ideale è gestire i dati memorizzati nei database relazionali all'interno di una soluzione aziendale SRM globale.

Impatto di crescita del database relazionale

L'accelerazione della crescita dei database nelle aziende e nelle applicazioni, combinata con la drammatica crescita della grafica, dei supporti audio e video, ha creato una domanda crescente di modi migliori per gestire dati ed un fabbisogno di soluzioni di storage più efficienti e meno costose in tutto il ciclo di vita dei dati. In seno al mercato del data storage, il compito più difficile è gestire la crescita dei database relazionali che guidano le applicazioni mission-critical — la spina dorsale degli attuali processi decisionali aziendali e del vantaggio competitivo.

L'impatto della crescita dei database va ben oltre l'aumento dei costi di storage ed è anche critica alla continuità operativa e alle strategie di disaster recovery. I database più grossi consumano in modo significativo più tempo per le fasi di ricostruzione e di restore, mentre i database relazionali sovraccarichi degradano le prestazioni e limitano la disponibilità delle applicazioni critiche. Il tuning del database e il costo degli upgrade hardware, software e di storage offrono dei ritorni inferiori. Infine, in base alle politiche di conservazione dei dati, le aziende mantengono online gran parte dei propri dati storici per motivi legali e di auditing, benché a molti di questi dati si acceda raramente.

Pur se la gestione del ciclo di vita dei dati è un aspetto critico per l'azienda, ad oggi esistono pochi standard che assistono le aziende nel formulare ed implementare strategie di data retention a lungo termine. Basandosi su aspetti operativi e legislativi, le organizzazioni IT devono sviluppare un piano per gestire i dati aziendali in un complesso ambiente di database relazionale. Quindi, come possono implementare le aziende la migliore metodologia per gestire questi dati critici per l'intero ciclo di vita?

Il ruolo dell'Active Archiving

L'Active Archiving è essenziale per gestire in modo efficiente il ciclo di vita dei dati, nel rispetto delle necessità di data retention e di riduzione dei costi.

L'Active Archiving è l'unico modo con cui i dati a cui si accede raramente possono essere archiviati con sicurezza e rimossi dal database relazionale online e spostati in un altro supporto di storage, pur conservando una facilità di accesso ai dati archiviati nel loro contesto business.

Tuttavia, prima di sviluppare una strategia di Active Archiving, un'organizzazione deve prima identificare tutti i tipi di dati aziendali per assicurare una completa comprensione della conformazione e dell'utilizzo dei dati, e per identificare le necessità di data retention e di storage appropriato.

Tra i tipici dati aziendali sono compresi tutti quelli transazionali inerenti le applicazioni business e i database associati, come i sistemi di paghe e stipendi, i sistemi di gestione clienti e quelli di gestione degli acquisti.

Questa analisi assicura il migliore mix di quali dati debbano rimanere online e quali devono essere archiviati per assicurare un bilancio economico di tutto il loro ciclo di vita. Questo processo assicura anche che le applicazioni database dell'azienda siano mantenute entro una dimensione gestibile che garantisca le prestazioni e la disponibilità dei sistemi critici.

L'obiettivo di una efficiente gestione del ciclo di vita dei dati è di mantenere i dati storici finché sono necessari, ma non oltre. Questo approccio è assimilabile a una sorta di 'Just-in-Time' in termini di accessibilità dei dati.

Cos'è l'Active Archiving?

Al di là della definizione tradizionale di archiviazione, l'Active Archiving è una solida tecnologia che archivia e rimuove con sicurezza dai complessi database relazionali opportuni sottoinsiemi di dati raramente utilizzati con un'accuratezza del 100%. Le aziende possono memorizzare i dati archiviati e mantenerli 'attivi' per un acces-

so agevole secondo necessità. Inoltre, viene preservata l'integrità referenziale e il contesto business.

Gli utenti possono anche accedere ed effettuare il restore dei dati archiviati in modo selettivo e integro da un punto di vista referenziale, eliminando la necessità di effettuare il restore di tutti i dati archiviati solo per recuperare alcune righe.

Queste possibilità riducono in modo drammatico l'overload del database, permettendo alle aziende di ridurre il fabbisogno di storage, di migliorare la responsività delle applicazioni e di riallocare le attuali capacità nel supportare più utenti e più transazioni.

L'Active Archiving consente alle organizzazioni IT di massimizzare i benefici delle attuali soluzioni di storage SAN, NAS e HSM, essendo complementare a queste tecnologie; specialmente nei sistemi HSM, per abilitare un approccio di best-practice 'organizzata' per la gestione dei dati relazionali storici che possono essere parte integrale dell'SRM aziendale.

Active Archiving e HSM

È vero che l'Active Archiving e l'HSM risolvono entrambe il problema della crescita esplosiva dei dati spostandoli su dispositivi di memorizzazione più economici. Tuttavia, l'Active Archiving è progettato per i dati relazionali, mentre l'HSM è più indicato per altri tipi di dati, ad esempio file di documenti, immagini e videoclip. Anche se l'HSM è ideale per gestire questi tipi di dati, è poco indicato per la gestione di tabelle di database relazionali, che possono essere di dimensioni notevoli.

L'Active Archiving gestisce i dati relazionali a livello di record, mentre la gestione dell'HSM è a livello di tabella o di dataset (una tabella relazionale è fisicamente memorizzata come file). L'HSM esegue la funzione di migrazione in base all'ultima volta che una particolare tabella o file di database ha avuto accesso.

È probabile che gli utenti debbano accedere a una piccola parte del database almeno una volta durante nel periodo in cui l'amministratore dei sistemi di storage ha designato che i dati debbano essere mantenuti al massimo livello. Per questa ragione, è molto probabile che l'intero file di database relazionale continuerà a risiedere al massimo livello di storage.

Ad esempio, una tabella Clienti di database sarà soggetta ad un acceso con frequenza regolare, mantenendola a Livello Uno.

Tuttavia, solo un subset di questi dati resta 'significativo' (ossia, i clienti attuali), di conseguenza, l'intero dataset deve essere mantenuto sul server poiché l'HSM non può distinguere i dati relazionali a livello di record.

Le aziende che hanno già adottato l'HSM comprenderanno i benefici di una gestione dati "organizzata". Con l'Active Archiving, le aziende ottengono simili benefici con i dati relazionali.

Anche se l'HSM può migrare tabelle di database relazionali in alto e in basso nella gerarchia HSM, la dimensione del database non cambia.

Per contro, l'Active Archiving riduce i database relazionali archiviando e rimovendo subset di dati correlati referenzialmente integri.

Questa possibilità riunisce le migliori caratteristiche di entrambi, combinando l'Active Archiving per i database relazionali, ed applicando le regole HSM per gestire i dati archiviati.

Scelta delle metodologie di archiviazione

La Princeton Softech ha clienti che hanno archiviato e rimosso il 65% dei propri database già dal primo archivio di produzione.

Questa possibilità libera una tremenda potenza di calcolo migliorando le prestazioni, la disponibilità e l'implementazione di nuove applicazioni, senza necessità di upgrade. Inoltre, si libera una gran quantità di spazio su disco per altri impieghi.

Se schedulato con regolarità l'Active Archiving continua a liberare spazio su disco, risparmiando milioni in upgrade hardware e software.

Essendo l'Active Archiving una soluzione efficace a lungo-termine al problema della crescita esplosiva dei database, è un aspetto critico di una strategia di data storage aziendale.

Una completa metodologia aziendale di Active Archiving deve fornire la possibilità di archiviare dati da una varietà di piattaforme e database relazionali.

La soluzione ideale di Active Archiving deve anche garantire di mantenere l'integrità referenziale e il contesto business dei dati archiviati e offrire un accesso agevole. Inoltre, deve essere possibile gestire e memorizzare i dati archiviati sui supporti di memorizzazione più convenienti (online in un archivio database, quasi-online in un file server o su un dispositivo ottico o offline su nastro).

Conclusioni

Un efficace Storage Resource Management permette alle aziende di ridurre i costi di storage, migliorare la gestione di dati e mantenerli accessibili in tutto il ciclo di vita. Insieme alle tecnologie di punta di storage, l'Active Archiving deve essere parte integrante di ogni iniziativa SRM.

Le aziende possono rimuovere i dati storici a cui si accede di rado dai database sovraccarichi e memorizzarli su supporti più economici.

La soluzione migliore e più completa ai problemi di esplosione dei dati e per gestire i dati in tutto il loro ciclo di vita richiede la visione generale fornita dell'SRM in combinazione con l'approccio più raffinato e garantito offerto dall'Active Archiving.

Applicazioni Web con ASP .NET

Il Web è ormai diventato una vera e propria piattaforma per lo sviluppo di applicazioni. In questo ambito ASP.NET si propone come una valida tecnologia per la realizzazione di applicazioni Web professionali ed affidabili, rappresentando un significativo passo avanti rispetto alla tecnologia Active Server Pages. "Applicazioni Web con ASP.NET" illustra le caratteristiche di questa nuova tecnologia descrivendo, con un approccio orientato alla pratica, gli elementi messi a disposizione dello sviluppatore: dall'uso dei controlli server all'inter-facciamento con ADO.NET per l'accesso a database, dalla creazione di controlli personalizzati alla configurazione e all'ottimizzazione delle applicazioni, dalla gestione della sicurezza alla creazione di servizi Web, dal debugging alla migrazione da ASP. Il tutto è corredato da numerosi esempi per fissare i concetti ed una serie di appendici di riferimento per aiutare lo sviluppatore nella realizzazione delle proprie applicazioni.

Applicazioni Web con ASP .NET
Andrea Chiarelli
pagg. 330
ISBN 8881500116
€ 39.00

dal SOMMARIO

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

technical BOOK

Clicca su www.infomedia.it per leggere il capitolo d'esempio

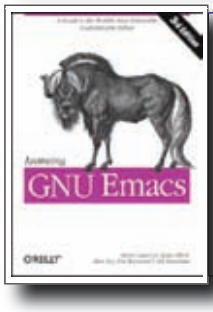
IN OFFERTA VBJ 61

C-Didattica e programmazione di A. Kelley - I. Pohl

750 pp. euro 39,00
ISBN 8871922190

MySQL Tutorial di L. Welling - L. Thomson

320 pp. euro 24,00
ISBN 8871922220



Learning GNU Emacs, 3rd Edition di Cameron, D. - Elliott J. - Loy, M.

III ed. dicembre 2004

ISBN 0596006489
534 pp. euro 39,95

Scrivi a book@infomedia.it specificando nell'oggetto della e-mail:

IN OFFERTA VBJ n. 61

OPPURE

inviaci il coupon sottostante al numero di fax

0587/732232

Potrai acquistare i libri qui riportati con uno

SCONTO ECCEZIONALE

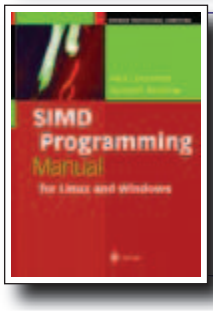
del **10%** anche se acquisti solo **un libro**

OPPURE

del **20%** se acquisti **3 libri**

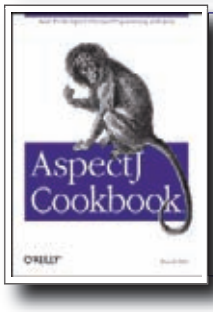
Architettura e organizzazione dei calcolatori 6/E - Progetto e prestazioni A cura di Ottavio D'Antona di W. Stallings

850 pp. euro 45,00
ISBN 8871922018



SIMD Programming Manual for Linux and Windows di Cockshott P. - Renfrew, K.

351 pp. euro 64,15
ISBN: 185233794X



AspectJ Cookbook di Miles R.

354 pp. euro 44,95
ISBN: 0596006543

GRUPPO EDITORIALE INFOMEDIA

Web: http://book.infomedia.it • E-mail: book@infomedia.it • Fax: 0587/732232

THE SOFTWARE SIDE OF YOUR MIND — VBJ 61

DATI ANAGRAFICI

NOME		COGNOME		CODICE CLIENTE
VIA/PIAZZA		CAP	CITTÀ	PROV.
TELEFONO (*)	FAX	E-MAIL (*)		
DITTA				
INTESTAZIONE FATTURA			P.IVA	

Garanzia di riservatezza - Gruppo Editoriale Infomedia garantisce la massima riservatezza dei dati da lei forniti e la possibilità di richiederne gratuitamente la rettifica o la cancellazione scrivendo a: Responsabile Dati Gruppo Editoriale Infomedia Srl - v. Valdera P. 116 - 56038 Ponsacco (Pi). Le informazioni custodite nel nostro archivio elettronico verranno utilizzate al solo scopo di inviarle proposte commerciali. In conformità alla legge 675/96 sulla tutela dei personali.

ORDINE

TITOLO	CODICE ISBN	PREZZO	QUANTITÀ

(*) campo obbligatorio

SPESE DI SPEDIZIONE

☐ PER POSTA - + €3,00 ☐ A MEZZO CORRIERE - euro 7,00

TIPO DI PAGAMENTO

☐ ALLEGO ASSEGNO BANCARIO INTESTATO A "GRUPPO EDITORIALE INFOMEDIA Srl" - NON TRASFERIBILE

☐ CONTRASSEGNO (solo per spedizione a mezzo posta + €3,00)

☐ ALLEGO VERSAMENTO SU CC. POSTALE N. 14291561 INTESTATO A "GRUPPO EDITORIALE INFOMEDIA SRL - PONSACCO". Sul modulo di C.C.POSTALE scrivere la causale del versamento.

☐ ALLEGO FOTOCOPIA DEL BONIFICO BANCARIO EFFETTUATO SU C/C 10005424 CAB 71120 ABI 6370 CASSA DI RISPARMIO DI VOLTERRA AGENZIA DI PONSACCO

☐ CARTA DI CREDITO VISA/MASTERCARD/CARTASÌ

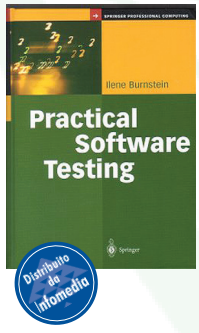
N. _____ SCADENZA _____

TITOLARE _____

FIRMA _____ NATO IL _____

Le spese di spedizione sono soggette a variazioni in base alle tariffe postali. Per i pagamenti con assegno, c/c postale e bonifico bancario consigliamo di chiedere conferma in redazione oppure di consultare il nostro sito internet. Grazie.

LIBRI



Practical Software Testing

Autore	Ilene Burnstein
Editore	Springer-Verlag
ISBN	0387951318
Lingua	Inglese
Anno	2003
Pagine	710
Prezzo	€ 74,85

La complessità crescente delle applicazioni software (in termini di funzionalità, integrazione, interoperabilità e usabilità) impone l'adozione di metodologie efficaci per la progettazione, l'implementazione e la manutenzione del codice. In questo contesto, il processo di test ricopre un ruolo sempre più importante, cruciale per raggiungere alti livelli di qualità.
Il testo introduce il lettore al testing nell'economia complessiva del software engineering, illustrando le problematiche in gioco e i benefici ottenibili con una rigorosa pianificazione delle attività di test. L'autore entra, quindi, nel dettaglio delle metodologie ed approcci al testing. Ciascuna strategia è studiata in profondità e i numerosi inserti (grafici, tabelle, diagrammi) forniscono un utilissimo supporto al lettore–studente. L'autore utilizza il Testing Maturity Model /TMM) quale framework per il processo di testing.
Il corpo principale del testo (quasi 600 pagine) esplora l'organizzazione dei test, delle attività, del controllo e del monitoraggio. Tanta dovizia e precisione, però, rende talvolta il libro un po' faticoso nella lettura, forse anche per l'orientamento universitario–specialistico impresso dall'autore. Aspetti importanti, quali metriche e programmi di training, sono affrontati in apposite sezioni.
Una ricca appendice fornisce riferimento pratico per la gestione delle attività. Infine, ciascun capitolo è corredato da una breve proposta di esercizi, utili per fissare i concetti acquisiti e prendere dimestichezza con le pratiche presentate.

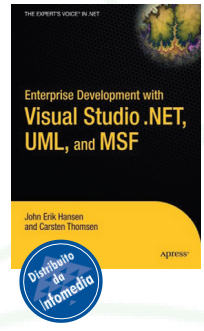
Pro

L'autore affronta brillantemente (e con rara padronanza) un tema complesso, ma allo stesso tempo imprescindibile per produrre software di qualità. È un libro "per studiare" e, come tale, risulta completo e rigoroso. Notevole, per ulteriori approfondimenti, la bibliografia segnalata per ciascuna sezione del testo.

Contro

È un libro "denso" e metodico, sconsigliato a chi cerca frettolosamente qualche informazione sulla pianificazione dei test.

Stefano Sanna



Enterprise Development with Visual Studio .NET, UML, and MSF

Autore	J. E. Hansen, C. Thomsen
Editore	Apress
ISBN	1590590422
Lingua	Inglese
Anno	2004
Pagine	1000
Prezzo	€ 60,94

I libri presenti sul mercato dedicati al linguaggio UML si sprecano, così come ve ne sono un certo numero che parlano dello sviluppo di applicazioni utilizzando Microsoft Visual Studio .Net. Questo libro, invece, parla di entrambi gli argomenti ed affronta il primo in relazione al secondo: analizza, cioè, l'UML e lo strumento di sviluppo di casa Microsoft congiuntamente. Questo suo approccio fa sì che si ponga particolare attenzione a quelle parti di UML che hanno una rilevanza specifica nel mondo VS.NET e che si tralascino quegli aspetti che, invece, non si applicano. Il libro non è rivolto ad un lettore che si occupi di sviluppo in senso stretto, bensì a coloro che di mestiere progettano e gestiscono progetti di tipo *Enterprise*. I capitoli che vanno dal primo al settimo analizzano in maniera non troppo approfondita l'UML e le sue applicazioni in un progetto. Gli aspetti teorici sono spesso intervallati da esempi pratici ed esercizi proposti al lettore. La seconda parte del libro (dal capitolo 8 al 15) approfondisce, invece, gli aspetti legati a Visual Studio .Net ed ai suoi tool per la gestione dei progetti. Molto interessanti le pagine dedicate agli Enterprise Project Template, meno quelle che parlano di Visual Source Safe, argomento che – onestamente – poteva anche essere omesso. L'ultima parte del libro, poi, parla dei diversi tool UML presenti sul mercato dando per alcuni di essi una descrizione sommaria sulle principali particolarità. La tipologia dell'argomento ed il carattere introduttivo con cui è affrontato, fanno di questa sezione quella meno significativa del testo in questione. Merita una menzione particolare una delle appendici nella quale sono riportati decine di link e riferimenti suddivisi per argomento mediante i quali è possibile approfondire in autonomia i temi esposti nel libro. In conclusione, un ottimo libro per chi si avvicina da "neofita" alla gestione ed allo sviluppo di progetti di una certa entità con Visual Studio .Net; un libro superficiale per coloro i quali possono vantare già qualche anno di esperienza sul campo.

Pro

Sono presenti molti esempi ed esercizi che stimolano il lettore ad applicare praticamente quello che ha appena letto.

Contro

Alcuni capitoli potevano essere omessi (VSS), altri presentano contenuti troppo superficiali.

Lorenzo Braidi

XML Secondo .NET

Come sarebbe a dire XML secondo .NET? Non avranno mica avuto la faccia tosta di fare anche un XML.NET?

di Dino Esposito

Già mi immagino le facce di alcuni fra voi. Come sarebbe a dire XML *secondo* .NET? Non avranno mica avuto la faccia tosta di fare anche un XML.NET? O peggio ancora un XML# o X# per brevità? Non s'era detto e scritto che XML è la quintessenza stessa del concetto di universalità? Non s'era valutato che dove c'è XML ci può stare di tutto, e che se si parla XML, moderno esperanto, si comunica con tutti? E allora che significa XML secondo .NET?

Eppure "XML secondo .NET" non è sbagliato o fuori misura. È giusto un tantino ("un tantinellino", come direbbe Totò) generico. Quel po' di generico che autorizza interpretazioni e alla fine genera persino sospetti.

XML è dunque un linguaggio per descrivere dati. Un documento XML è un file di testo in cui i marcatori identificano e qualificano specifiche parti di testo. In quanto testo può viaggiare su qualsiasi mezzo di trasporto e in quanto linguaggio universalmente riconosciuto viene perfettamente compreso a qualsiasi fermata decida di scendere. Non importa la piattaforma, non importano gli usi e costumi delle applicazioni attive da quelle parti. Perché tutto questo? E cosa ha cambiato .NET adattandolo alle sue necessità?

XML e il parser

Quali che siano le ragioni dell'universalità di XML (ce ne sono diverse), questa è un fatto. Su qualunque piattaforma ci si avventuri, con Java o senza Java, un parser per XML si rimedia sempre. Non importa se sia DOM o SAX, conta che ci sia e che permetta di "capire" il contenuto dei marcatori di testo ed eventualmente lo schema con cui tali marcatori sono stati disposti. Una volta che si dispone di un parser (e ce ne sono su qualunque piattaforma) trasformare il testo XML in una entità software perfettamente funzionale sulla piattaforma è un gioco da ragazzi non più difficile che aprire e chiudere file.

A parte il formato binario, il parser è unico? In altre parole, le funzionalità che implementa sono standard come per esempio HTTP? Sì e no. Diciamo che vi sono due tipi di parser: DOM e SAX. Ciascuno ha vantaggi e svantaggi e si presta bene a risolvere certi problemi piuttosto che altri. Di sicuro quello che fa il DOM non lo fa il SAX e viceversa. Gli insiemi delle loro funzionalità sono in larga parte disgiunti. L'intersezione, se effettivamente di intersezione si può parlare, è o no vuota? In certe implementazioni di parser DOM vengono lanciati eventi man mano che il modello del documento viene costruito. Il che può essere avvicinato al modo di lavorare di SAX. Un parser SAX, invece, scandisce i tag del documento XML man mano che li legge e lancia un evento per ogni apertura e chiusura. Il che potrebbe consentire di costruirsi al volo una specie di DOM fatto su misura. A parte la possibilità di utilizzare ciascuno di essi in modo da arrivare ad ottenere le funzionalità dell'altro, DOM e SAX hanno nulla in comune e, anzi, nascono da approcci, anche filosofici, diametralmente opposti. DOM ha a che vedere con l'uso e l'abuso della memoria e la facilità di accesso ai dati e vede lo stato come un sicuro

Dino Esposito *è consulente e formatore per Wintellect, per cui tiene il corso ADO.NET. Collabora con MSDN Magazine e MSDN Voices. È autore di Building Web Solutions with ASP.NET e ADO.NET (Microsoft Press), è anche co–fondatore di www.vb2themax.com. È raggiungibile via e–mail all'indirizzo esposito@infomedia.it*

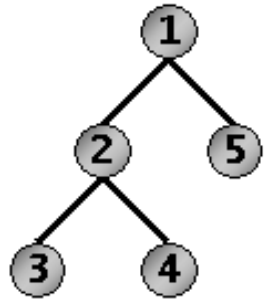


Figura 1 Visita di un albero in–depth first

rifugio e una dispensa riccamente fornita di ogni bendidio. SAX è per chi cerca solo quello di cui ha bisogno, che non ha memoria né tempo da sprecare.

In termini di complessità computazionale temporale (correggetemi se sbaglio) sono allo stesso livello ed entrambi gli algoritmi alla fine visitano una volta tutti i nodi. Essi sono nulla più che visite dell'albero. Dunque O(n) per entrambi. La differenza abissale sta nella complessità in spazio delle loro implementazioni. La complessità spaziale di SAX è pressoché costante o al massimo dipendente dal numero di attributi di un dato nodo. La complessità in spazio del DOM è sempre O(n). Tutti i nodi ed attributi del DOM sono sempre in memoria. Il DOM man mano che visita l'albero aumenta l'occupazione di memoria perché – per definizione – il DOM crea un'immagine in memoria dell'albero XML.

Alla luce delle applicazioni pratiche, è chiaro che il SAX è ideale per scandire documenti XML di una certa dimensione e lascia al programmatore, caso per caso, la piacevole incombenza di decidere cosa fare. Il DOM è un albero read/write che prima fa quello che deve fare e poi lascia ulteriori incombenze al programmatore. Creare un'instanza di documento con il DOM è relativamente oneroso e strettamente dipendente dalla dimensione dell'albero XML. Creare un'istanza di un documento con SAX è qualcosa che viene completamente lasciato alla fantasia e immaginazione del programmatore e del suo problema. Questa libertà può portare a creare alberi XML più piccoli e fatti su misura.

Orbene, DOM e SAX sono tutto quello di cui un programmatore adulto, vaccinato e con la barba ha bisogno oggigiorno? Sono essi i soli strumenti che servono? O non è forse il caso di fare un salto in un negozio di bricolage o ferramenta per vedere gli ultimi ritrovati della scienza e della tecnica?

Come detto, DOM e SAX nascono da approcci diametralmente opposti e, probabilmente per questa ragione, offrono soluzioni abbastanza estreme. Siamo sicuri che non ci sia spazio per qualcosa di nuovo e, soprattutto, di intermedio? A conti fatti, questo è XML secondo .NET.

Modello a cursore

Non è che XML .NET sia un nuovo tipo di XML. Però con .NET ci troviamo di fronte ad una nuova tipologia di parser che casca più o meno nel mezzo di DOM e SAX. A dire il vero, più vicino a SAX che a DOM. Dunque, il titolo XML secondo .NET sottintende proprio questo nuovo modello di parser che, al momento, è specifico solo della piattaforma .NET. È un problema? Certamente no e proprio perché è un parser: finché siete su .NET lo usate per leggere, scrivere e validare documenti XML. Una volta fuori da .NET quello che resta sono solo file di testo XML. Magari con schema. Ma lo schema è anch'esso una raccomandazione del W3C.

Il modello a cursore riprende e adatta il concetto di cursore che conosciamo dal mondo dei database. Allo stesso modo in cui su database ci si sposta di record in record e quando si sta su un record lo si può investigare in lungo e largo, così lavorando su un file XML ci si sposta di nodo in nodo e quando si sta su un nodo lo si può scandagliare attributo per attributo. L'aderenza tra i due modelli mi pare quantomai stretta. Vediamo il modello a cursore in relazione a DOM e SAX. Non è chiaramente come DOM dal momento che non c'è mai, in nessun momento, tutto l'albero in memoria. L'occupazione di memoria è limitata e paragonabile a quella dell'algoritmo SAX. Il modello a cursore è più potente di SAX per il fatto che pur limitandosi ad una passata consente di vedere una piccola porzione di stato quando il cursore è posizionato su un certo nodo. Il paragone più vicino è indubbiamente quello con SAX. Il modello a cursore riprende la filosofia di base di SAX ma lo rende più specifico e lo adatta al dato da rappresentare. Sapere che un tag è stato aperto o chiuso nella scansione fisica del file XML è significativo fino ad un certo punto. Un documento XML è un albero fatto da nodi, ciascuno dei quali ha un suo stato fatto di attributi. La scansione ha più senso farla nodo per nodo traducendosi in un algoritmo di visita "logica" di un albero piuttosto che in una scansione del file alla ricerca di caratteri < e />. Il modello a cursore visita ciascun nodo dell'albero in ordine anticipato o *in-depth first*. In sostanza, quando si è posizionati su un nodo radice di un sottoalbero il nodo successivo è quello più a sinistra (il primo in sequenza nel file). La visita va giù prendendo sempre il successivo finché si arriva ad una foglia. Dalla foglia si risale al primo nodo al livello del nodo padre. Come in **Figura 1**.

Il modello a cursore è veloce quanto SAX e molto più di DOM per un motivo molto semplice. Si tratta di un cursore forward-only e sostanzialmente read-only. Da ciascun nodo non è possibile tornare indietro ma solo muovere al nodo successivo. L'approccio è sostanzialmente read-only

perché le classi .NET che lo implementano sono o read-only o write-only, il che in aggiunta alla mono-direzionalità rende particolarmente efficace e veloce la visita.

Classi .NET per documenti XML

Nel .NET Framework tutte le classi che hanno a che fare con XML sono concentrate nel namespace *System.Xml*. Altri namespace interessanti sono *System.Xml.Serialization*, *System.Xml.XPath*, e *System.Xml.Xsl*. Quali classi contengano gli ultimi due è abbastanza chiaro. Il namespace Serialization contiene le classi per la serializzazione di tipi di dato (cioé classi) .NET in XML pronti per l'esportazione verso altre piattaforme.

Le classi XML in .NET sono costruite al di sopra di standard riconosciuti quali DOM Level 2, XPath 1.0, XSLT 1.0, XML Schemas (XSD), SOAP. In aggiunta a tutto ciò vi sono estensioni significative come appunto il modello a cursore.

La classe base astratta per la manipolazione di documenti XML è *XmlReader*. Essa rappresenta uno stream con cui leggere i dati di un nodo senza fare caching e in modo forward-only. A differenza di SAX, la classe XmlReader non effettua il push dei dati verso il client tramite interfacce predefinite. Al contrario, XmlReader lascia il client abbastanza libero di accedere e memorizzare solo i dati che effettivamente servono in un particolare contesto.

La classe XmlReader è alla base di tutte le funzionalità XML in .NET. Ciò significa anche che l'implementazione del modello DOM in .NET avviene tramite opportuni reader. In sostanza la rappresentazione in memoria di un documento XML tramite un DOM viene costruita accedendo ai nodi del documento XML tramite una classe derivata da XmlReader.

Leggere dati con XmlReader

Nel **Listato 1** vediamo il sorgente di una semplice applicazione .NET che legge un file XML. La classe *XmlTextReader* è una classe che implementa concretamente le funzionalità base della classe XmlReader. Essa prende il nome del file XML a livello del costruttore di classe.

```
XmlTextReader xtr = new XmlTextReader(fileName);
```

A questo punto il reader è fermo all'inizio del file in una posizione che equivale al BOF di alcuni cursori di database. Per spostarsi sul primo nodo è necessario chiamare il metodo *Read*. Da notare il fatto che in .NET vi sono numerose classi che funzionano a cursore e sono tutte quelle il cui nome finisce per Reader. Per esempio, *SqlDataReader*. Tutte queste classi spostano il puntatore del "cursore" tramite il metodo Read e una volta che il puntatore è a posto usano una sintassi ad-hoc per leggere il contenuto del particolare elemento. Il codice che segue mostra come leggere il nome dell'i-esimo nodo e scriverlo tra parentesi angolate sul dispositivo di output.

```
xtr.Read();
Console.WriteLine("<{0}>", xtr.Name);
```

Le righe che seguono mostrano invece come leggere tutto il contenuto del nodo corrente e chiudere il tag.

```
Console.WriteLine(xtr.ReadInnerXml());
Console.WriteLine("</{0}>", xtr.Name);
xtr.Close();
```

È vero che da un nodo non si può saltare troppo in là e che lo stato del nodo corrente si limita agli attributi, ma se fosse necessario il metodo *ReadInnerXml* legge tutto il codice XML relativo al sottoalbero centrato nel nodo corrente.

Il metodo Read sposta il puntatore interno al nodo in maniera relativamente cieca senza considerare la tipologia del nodo.

Così capita che Read possa spostare il puntatore da un nodo di tipo commento al nodo radice dei dati e da un certo nodo attributo al successivo nodo dati. Il reader però dispone di metodi che permettono di saltare a piè pari tutti i nodi senza dati. Per esempio il codice che se-

Listato 1 Lettura di un file XML con le classi .NET

```
using System;
using System.Xml;

class MyXmlApp
{
  public static void Main(String[] args)
  {
     try {
        String fileName = args[0];
        XmlTextReader xtr = new XmlTextReader(fileName);

        // Open the stream and moves to the root
        xtr.Read();

        Console.WriteLine("<{0}>", xtr.Name);
        // Read the whole content of the node
        Console.WriteLine(xtr.ReadInnerXml());
        Console.WriteLine("</{0}>", xtr.Name);
        xtr.Close();
     }
     catch (Exception e) {
        Console.WriteLine("Error:\t{0}", e.Message);
     }

     return;
  }
}
```

gue mostra come aprire un reader e spostare il cursore sull'effettiva radice dei dati saltando eventuali commenti, direttive e processing instruction.

```
XmlTextReader reader = new XmlTextReader(fileName);
reader.Read();
reader.MoveToContent();
```

Solitamente in XML—almeno nei file XML compatibili con lo standard XML 1.0—il primo nodo è una dichiarazione

```
<? xml version="1.0" ?>
```

In altri casi, il primo nodo può essere una processing instruction, così come un commento, un doctype e, naturalmente, anche un elemento dati. Invocando il metodo *MoveToContent* ci si assicura che il primo nodo effettivamente elaborato è un nodo dati, cioè un nodo di tipo *element*.

Un altro metodo interessante è *MoveToElement* che rappresenta il solo caso in cui nel modello a cursore un metodo permette di tornare indietro ad un nodo già visitato. In realtà tornare indietro significa semplicemente riportarsi sul nodo principale dopo aver visitato i suoi attributi.

Il codice che segue mostra il tipico loop che legge e scrive in output tutti i nodi (nome e tipo) di un file XML.

```
while (reader.Read())
{
  Console.Write("Node Type: ");
  Console.Write(reader.NodeType.ToString());
  Console.Write(", Node Name: ");
  Console.Write(reader.Name);
}
```

Il metodo Skip consente di saltare un nodo proseguendo con quello immediatamente successivo. La classe *XmlTextReader* funziona bene se il documento XML è ben formato ma non assicura nulla circa l'aderenza ad un dato DTD o schema. Tuttavia XmlTextReader verifica che un DTD dichiarato sia effettivamente presente e sintatticamente corretto. Per effettuare la validazione, però, bisogna ricorrere ai servizi della classe XmlValidatingReader.

XML e .NET

Le classi .NET per lavorare con XML hanno una doppia interfaccia: classi reader che funzionano secondo il modello a cursore e classi DOM tradizionali. Più un pugno di classi per integrare ADO .NET e XML. Dentro la famiglia delle classi a cursore si riconoscono diverse tipologie: lettura, scrittura e validazione. In questo articolo ci siamo soffermati sulla sola lettura. In seguito torneremo sull'argomento per analizzare la validazione e la creazione di documenti XML.

Campagna abbonamenti 2005

SI!

MI ABBONO ALLE RIVISTE INFOMEDIA

	PREZZO ABBONAMENTO	
Computer Programming (11 numeri)	€ 50,00	☐
DEV (11 numeri)	€ 50,00	☐
Visual Basic Journal (6 numeri)	€ 44,00	☐
Login (6 numeri)	€ 44,00	☐
CP Web (11 numeri)	€ 42,00	☐
DEV Web (11 numeri)	€ 42,00	☐
VBJ Web (6 numeri)	€ 42,00	☐
Login Web (6 numeri)	€ 42,00	☐

		CP	DEV	VBJ	Login
2 riviste cartacee a scelta	€ 82,00	☐	☐	☐	☐
3 riviste cartacee a scelta	€ 107,00	☐	☐	☐	☐
4 riviste cartacee	€ 132,00	☐	☐	☐	☐
		CP W.	DEV W.	VBJ W.	Login W.
2 riviste Web a scelta	€ 75,00	☐	☐	☐	☐
3 riviste Web a scelta	€ 90,00	☐	☐	☐	☐
4 riviste Web	€ 110,00	☐	☐	☐	☐
4 riviste cartacee+ 4 riviste Web	€ 220,00	☐			

Gli abbonamenti decorrono dal primo numero raggiungibile. Per attivazioni retroattive contattaci al numero 0587/736460 o invia un'e-mail all'indirizzo abbonamenti@gruppoinfomedia.it.

I prezzi degli abbonamenti per l'estero sono maggiorati di spese di spedizione.

GRUPPO EDITORIALE INFOMEDIA S.R.L.
Via Valdera P. 116 - 56038 Ponsacco (Pi) - Tel. 0587/736460 - **Fax 0587/732232**
abbonamenti@infomedia.it

Per abb.ti spedizione posta prioritaria aggiungere:

+ euro 18,00 per CP
+ euro 18,00 per DEV
+ euro 10,00 per VBJ
+ euro 10,00 per Login

INDIRIZZO DI SPEDIZIONE

CODICE CLIENTE _____

Nome/Società

Indirizzo

CAP Città Prov.

Telefono

Fax

E-Mail

MODALITA' DI PAGAMENTO

- ☐ Allego fotocopia del Bonifico Bancario effettuato sul **C/C 000010005424** CAB 71120 ABI 6370 Cin "M" - Cassa di Risparmio di Volterra - Agenzia di Ponsacco
- ☐ Allego fotocopia della ricevuta del versamento sul C/C Postale **N.14291561** intestato a "Gruppo Editoriale Infomedia S.r.l. - Ponsacco"
- ☐ Allego assegno bancario intestato a "Gruppo Editoriale Infomedia S.r.l." NON TRASFERIBILE
- ☐ Contrassegno (+ € 7,00 contributo spese postali)
- ☐ Autorizzo l'addebito dell'importo di € _____ sulla mia CARTASI'/VISA

N. __ __ __ __ __ __ __ __ __ __ __ __ __ __ __ __

Scad. __ __ __ __ mm/aa

Nome del Titolare _____

Data di nascita _____

Firma del Titolare _____

- ☐ Collegati al sito www.infomedia.it dove potrai effettuare il pagamento con carta di credito in modalità sicura (banca SELLA)

PRIVACY
Si autorizza al trattamento dei dati personali ai sensi delle vigenti norme sulla privacy

FIRMA _____

L'IVA sul prezzo dell'abbonamento è assolta dall'Editore e non sussiste l'obbligo di emissione della fattura, ai sensi del D.M. 9 aprile 1993, art.1, comma 5; pertanto, ai fini contabili, farà fede la sola ricevuta di pagamento; perciò la fattura verrà emessa solo se esplicitamente richiesta al momento dell'ordine.

Garanzia di riservatezza - Gruppo Editoriale Infomedia garantisce la massima riservatezza dei dati da lei forniti e la possibilità di richiederne gratuitamente la rettifica o la cancellazione scrivendo a: Responsabile Dati - Gruppo Editoriale Infomedia Srl - Via Valdera P. 116 - 56038 Ponsacco (PI). Le informazioni custodite nel nostro archivio elettronico verranno trattate in conformità alla legge 675/96 sulla tutela dati personali.

Collezione VBJ su 3 CD-Rom

Tutti gli articoli pubblicati su VBJ fino al n. 36 (Dicembre 2000) in formato PDF e HTML

E INOLTRE:

- L'indice di tutti gli articoli pubblicati su VBJ
- Freeware, Shareware ed altro software per chi programma in VB

2 CD IN OMAGGIO!

GRUPPO EDITORIALE INFOMEDIA

E-mail: cdrom@infomedia.it
Tel. 0587/736460
Fax 0587/732232

CD VBJ 1999/2000+
CD VBJ 1998+
CD VBJ 1995/1997=

3 CD al prezzo di 25,00 euro

Garanzia di riservatezza - Gruppo Editoriale Infomedia garantisce la massima riservatezza dei dati da lei forniti e la possibilità di richiederne gratuitamente la rettifica o la cancellazione scrivendo a: Responsabile Dati - Gruppo Editoriale Infomedia Srl - Via Valdera P. 116 - 56038 Ponsacco (PI). Le informazioni custodite nel nostro archivio elettronico verranno trattate in conformità alla legge 675/96 sulla tutela dati personali.

☐ ALLEGO FOTOCOPIA DEL BONIFICO BANCARIO EFFETTUATO SUL C/C 000010005424 - CAB 71120 - ABI 6370 - Cin "M" - Cassa di Risparmio di Volterra - Agenzia di Ponsacco

☐ ALLEGO ASSEGNO BANCARIO INTESTATO A "GRUPPO EDITORIALE INFOMEDIA Srl" - NON TRASFERIBILE

☐ CONTRASSEGNO (solo per spedizione a mezzo posta euro 3,00)

☐ ALLEGO VERSAMENTO SU C/C POSTALE N.14291561 INTESTATO A "GRUPPO EDITORIALE INFOMEDIA SRL - PONSACCO" Sul modulo di C/C POSTALE scrivere la causale del versamento.

☐ CARTA DI CREDITO VISA/MASTERCARD CARTASÌ

N. _____ SCADENZA _____

TITOLARE _____

FIRMA _____ NATO IL _____

SPESE DI SPEDIZIONE ☐ PER POSTA - euro 3,00 ☐ A MEZZO CORRIERE - euro 7,00

Le spese di spedizione sono soggette a variazioni in base alle tariffe postali. Per i pagamenti con assegno, c/c postale e bonifico bancario consigliamo di chiedere conferma in redazione oppure di consultare il nostro sito internet.

NOME		COGNOME	
DITTA			
INDIRIZZO DI SPEDIZIONE		CAP	CITTÀ	PROV.
TELEFONO	FAX		E-MAIL	
INDIRIZZO DI FATTURAZIONE				
			P. IVA	

Programmazione delle Smart Card

Standard, specifiche e piattaforme di sviluppo per Java, C e Visual Basic

Internet non è in grado di proporre un mezzo di identificazione certificata degli interlocutori né un livello adeguato di protezione delle trasmissioni: le tecnologie basate su smart card permettono di sviluppare applicazioni sicure per i settori delle telecomunicazioni mobili, PayTV, commercio elettronico, sistemi bancari e di accesso sicuro ai sistemi informativi.

Tra gli argomenti affrontati nel testo:

lo standard ISO 7816 - la piattaforma Javacard - Architettura PC/SC, le specifiche PKCS#11 - il Framework OpenCard - utilizzo di smart card con Windows, Outlook e Internet Explorer - Crittografia.

Viene inoltre descritto un emulatore di smart card e di dispositivo di lettura utilizzabile per esercitarsi con le istruzioni APDU ISO 7816.

€ 18.00 - pagg. 160
ISBN 8881500140

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

WEB: http://www.infomedia.it - e-mail: book@infomedia.it - tel: 0587/736460

Collezione VBJ su 3 CD-Rom

Tutti gli articoli pubblicati su VBJ fino al n. 36 (Dicembre 2000) in formato PDF e HTML

E INOLTRE:

- *L'indice di tutti gli articoli pubblicati su VBJ*
- *Freeware, Shareware ed altro software per chi programma in VB*

2 CD IN OMAGGIO!

GRUPPO EDITORIALE INFOMEDIA

E-mail: cdrom@infomedia.it
Tel. 0587/736460
Fax 0587/732232

CD VBJ 1999/2000+
CD VBJ 1998+
CD VBJ 1995/1997=

3 CD al prezzo di 25,00 euro

Garanzia di riservatezza - Gruppo Editoriale Infomedia garantisce la massima riservatezza dei dati da lei forniti e la possibilità di richiederne gratuitamente la rettifica o la cancellazione scrivendo a: Responsabile Dati - Gruppo Editoriale Infomedia Srl - Via Valdera P. 116 - 56038 Ponsacco (PI). Le informazioni custodite nel nostro archivio elettronico verranno trattate in conformità alla legge 675/96 sulla tutela dati personali.

- ☐ ALLEGO FOTOCOPIA DEL BONIFICO BANCARIO EFFETTUATO SUL C/C 000010005424 - CAB 71120 - ABI 6370 - Cin "M" - Cassa di Risparmio di Volterra - Agenzia di Ponsacco
- ☐ ALLEGO ASSEGNO BANCARIO INTESTATO A "GRUPPO EDITORIALE INFOMEDIA Srl" - NON TRASFERIBILE
- ☐ CONTRASSEGNO (solo per spedizione a mezzo posta euro 3,00)
- ☐ ALLEGO VERSAMENTO SU C/C POSTALE N.14291561 INTESTATO A "GRUPPO EDITORIALE INFOMEDIA SRL - PONSACCO" Sul modulo di C/C POSTALE scrivere la causale del versamento.
- ☐ CARTA DI CREDITO VISA/MASTERCARD CARTASÌ

N. _____ SCADENZA _____

TITOLARE _____

FIRMA _____ NATO IL _____

SPESE DI SPEDIZIONE ☐ PER POSTA - euro 3,00 ☐ A MEZZO CORRIERE - euro 7,00

Le spese di spedizione sono soggette a variazioni in base alle tariffe postali. Per i pagamenti con assegno, c/c postale e bonifico bancario consigliamo di chiedere conferma in redazione oppure di consultare il nostro sito internet.

NOME _____ COGNOME _____

DITTA _____

INDIRIZZO DI SPEDIZIONE _____ CAP _____ CITTÀ _____ PROV. _____

TELEFONO _____ FAX _____ E-MAIL _____

INDIRIZZO DI FATTURAZIONE _____

P. IVA _____