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.62

Marzo/Aprile 2005 - bimestrale - Anno 11 - Euro 7,75

Strumenti e metodi per l'ACCESSO ai DATI

Scrivere add-in per MS Access
Stored Procedure sotto controllo
Typed Data Object
Logica applicativa in VB.NET

Enterprise
SOA e il .NET Framework

Architect's Corner
Un cluster in 15 clic

Software Engineering
I Design Pattern più famosi

.NET Tools
Firefox Extensions e Query Commander

I.A.
Oracoli di Turing e Sistemi Logicamente Aperti

I miti
Configurazione e deployment in ASP.NET

Primo piano
Verso la sanità elettronica

GRUPPO EDITORIALE INFOMEDIA

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

EDITORIALE

Il fascino delle coding guidelines

Di recente ho scritto un libro sull'uso delle linee guida e best practice nella scrittura di applicazioni .NET. Non voglio approfittare di questo spazio per fare pubblicità al libro (e quindi ometterò di menzionarne il titolo...) però avendo dedicato così tanto tempo a questo argomento ho pensato di condividere qualche riflessione con i lettori di Visual Basic & .NET Journal.
Anche se i termini *guideline* e *best practice* sono spesso usati in modo intercambiabile, i due termini si riferiscono a concetti differenti. Le linee guida sono regole a cui occorrerebbe attenersi quando si scrive codice, quando si assegna un nome a una classe o metodo, quando si organizzano le varie parti di un progetto, e così via. Il loro obiettivo è rendere il codice più leggibile e ordinato, manutenibile, riutilizzabile, e auto–documentante.
Per *best practice*, invece, di solito si intende una raccomandazione sull'uso corretto del linguaggio e delle classi del .NET Framework, allo scopo di creare programmi più sicuri, performanti, scalabili, e robusti. Secondo questa distinzione, ad esempio, l'uso di comandi ADO.NET parametrizzati è una best practice (e non una guideline) perché riduce la probabilità di riuscita di un attacco di tipo "SQL injection". Ovviamente sia le linee guida che le best practice non sono da intendersi come verità assolute, e ci sono casi in cui è conveniente o addirittura necessario infrangere le regole.
Già da questa descrizione sommaria si può intuire che in genere le best practice sono recepite più facilmente dagli sviluppatori delle guideline, perché offrono un vantaggio immediato e tangibile: codice più sicuro, più veloce, e così via. Al contrario, utilizzare una coding guideline richiede molta disciplina e attenzione e sembra offrire molto poco in cambio: codice più ordinato e pulito, ma non necessariamente più veloce. Ma questa è una percezione decisamente errata. Alcune guideline riguardanti i linguaggi .NET possono considerarsi "ufficiali", in quanto promulgate direttamente da Microsoft: ad esempio, i nomi delle classi eccezioni devono terminare con "Exception", il nome dei metodi e dei campi pubblici deve essere in PascalCase (ossia, le varie parole che lo compongono devono cominciare con una lettera maiuscola), e così via. Le guideline ufficiali Microsoft non coprono davvero tutti i possibili casi, ed è un bene perché linee guida troppo pignole e invasive andrebbero a scontrarsi con lo stile di programmazione che ciascun programmatore ha sviluppato nel corso degli anni. Però questa non è una buona scusa per non adottare *nessuno* stile di codifica: una guideline qualsiasi è sempre meglio di nessuna guideline. Quando due o più sviluppatori lavorano sullo stesso progetto, diventa importante mettersi d'accordo su uno stile di codifica il più uniforme possibile, in modo da poter facilmente intervenire sul codice scritto dai propri colleghi, oppure rivedere il codice a mesi o anni di distanza. Chiunque abbia la responsabilità di coordinare un gruppo (anche piccolo) di sviluppatori dovrebbe sentire una forte esigenza di uniformare il codice prodotto, per ridurre le spese di manutenzione, non dipendere troppo dal turnover delle persone, e avere la possibilità di spostare più facilmente uno sviluppatore da un progetto all'altro. Ovviamente non basta accordarsi sullo stile di codifica per rendere gli sviluppatori realmente interscambiabili, visto che ognuno ha le sue particolari competenze (database, grafica, Web, ecc.), però è un passo nella direzione giusta.

Francesco Balena
fbalena@codearchitects.com

VISUALBASIC & .NET JOURNAL
online.infomedia.it
n. 62 - marzo/aprile 2005
bimestrale - anno undicesimo

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

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

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

Collaboratori
Marco Aldrovandi
Gionata Aladino Canova
Dino Esposito, Andrea Ferendeles
Ignazio Licata, Gianluca Masina
Davide Mauri, Paolo Pialorsi
Francesco Quaratino, Ingo Rammer
Alberto Rosotti, 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


Your potential. Our passion.™
Microsoft®
Prima pensa in grande. Poi costruisci.
Con Visual Studio .NET 2003 puoi realizzare applicazioni Web aziendali scrivendo meno codice: le tue idee diventano realtà più velocemente di quanto tu possa immaginare. Lo stile RAD delle Web Form ti permette di creare rapidamente applicazioni per qualsiasi browser e piattaforma. In più, l'utilizzo della tecnologia IntelliSense all'interno del migliorato editor HTML ti aiuta a velocizzare la scrittura dei tag. Tutto ciò significa che puoi diventare più produttivo e più abile nel concretizzare le tue idee. **Misco Italy S.p.A**, fornitore di prodotti per informatica e ufficio, ha realizzato in ASP.NET un nuovo sito per offrire servizi completi ai propri clienti. L'utilizzo di Visual Studio .NET e del debugger integrato, l'uso dei controlli Web di ASP.NET e il nuovo linguaggio C# hanno garantito tempi di sviluppo e di test senza precedenti.
microsoft.com/italy/vstudio/value/
© 2004 Microsoft. Tutti i diritti riservati. Tutti i marchi registrati citati sono di proprietà delle rispettive società.
TRADE
TOTAL QTY
TX
Microsoft®
Visual Studio®

SOMMARIO

MARZO/APRILE

N.62

SPECIALE

ENTERPRISE

SOFTWARE ENGINEERING

RUBRICHE

INTELLIGENZA ARTIFICIALE

Oracoli di Turing e Sistemi Logicamente Aperti 41

In questo articolo, anteprima assoluta, parliamo di Nuove Frontiere della Computazione: "frontiere" che hanno a che fare con il concetto stesso di computazione oltre il limite di Turing. In particolare sono illustrate due teorie della computazione, unite da un mix di idee nuove, tra cui la ricerca sui "sistemi logicamente aperti".

di Ignazio Licata

ARCHITECT'S CORNER

Un cluster in 15 clic 46

Specialità del giorno. Soluzioni pratiche invece della teoria.

di Ingo Rammer

PRIMO PIANO

Verso la sanità elettronica 51

di Alberto Rosotti

I MITI

Configurazione e deployment in ASP.NET 56

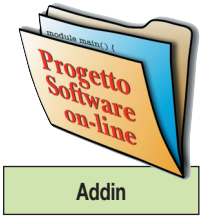
Vediamo come ASP.NET risolve molte delle limitazioni che affliggevano ASP.

di Dino Esposito

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.*

Scrivere add-in per Microsoft Access



Automatizzare i compiti ripetitivi riduce la possibilità di commettere errori, migliora l'efficienza e rende il programmatore più sorridente.

di Gionata Aladino Canova

Il programmatore è un animale particolare, generalmente pigro ma dotato di molta inventiva. Chi ha letto "Gli uomini vengono da Marte e le donne da Venere", capisce facilmente che, per le sue caratteristiche, il programmatore è quasi sempre maschio. Date un problema ad un programmatore ed esso ve lo risolverà. Affidate allo stesso individuo un compito ripetitivo: dopo poche iterazioni vi svilupperà un sistema che faccia il lavoro al posto suo; questo succede anche se, in termini di lavoro complessivo, la seconda soluzione dovesse essere più onerosa! Tralasciando le considerazioni filosofiche, tutte le volte che si automatizza un processo ripetitivo, si produce più in fretta e si riduce la possibilità di commettere errori. In questo articolo vedremo come sviluppare uno strumento atto allo scopo, che funziona dal lato Access, mentre in un prossimo appuntamento vedremo come sviluppare una ulteriore aggiunta per l'editor VBE. Il codice è disponibile come sempre al sito FTP di Infomedia ed il suo utilizzo è gratuito; è fornito completo di sorgenti che possono essere modificati e ridistribuiti a condizione di non rimuovere le indicazioni sull'autore. Per la comprensione dell'articolo serve una discreta padronanza dell'ambiente di Access e la conoscenza del VBA; i concetti esposti sono validi da Microsoft Access 2000 in poi.

Motivi per estendere le funzionalità di Microsoft Access

Microsoft Access ha avuto un buon successo poiché unisce la potenza di un database relazionale alla facilità di utilizzo tipica dei prodotti Microsoft. Sono fornite di base autocomposizioni che creano piccole applicazioni già funzionanti; esistono poi una vasta gamma di wizard che aiutano l'utente nei compiti più disparati.

Il limite di questi strumenti sta nell'essere stati creati per accontentare il massimo numero possibile di utenti (si tratta cioè di strumenti *orizzontali)*. Quando, nel creare le nostre applicazioni, ci troviamo di fronte ad un compito ripetitivo per cui non esiste nessuno strumento in Access, probabilmente abbiamo un'esigenza *verticale*. Il primo passo per creare la nostra cassetta degli attrezzi sarà un'attenta analisi di quello di cui avremo veramente bisogno. Appena scoperto cosa si può fare con una semplice toolbar, è facile cadere nella tentazione di aggiungerci tutto quello che ci viene in mente. Selezioniamo invece ciò che ci è realmente utile e sviluppiamolo al meglio.

Tipi di aggiunte

Le funzionalità di Microsoft Access possono essere estese in diversi modi. Per lavorare con Microsoft Access è necessario avere aperto un database.

Per i nostri scopi, consideriamo di avere aperto il database VBJ.mdb. Possiamo scrivere una routine VBA che colori tutte le maschere del database con un colore specifico, sal-

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 tramite e–mail all'indirizzo info@aladinoinformatica.it*

varla dentro VBJ.mdb stesso e richiamarla dalla finestra di debug. Oppure possiamo salvare la routine in un database di libreria ed agganciarlo dall'editor VBE tramite l'opzione Strumenti\Riferimenti. Lo svantaggio di questi approcci è che le routine sono incluse nell'applicazione e che serve una certa conoscenza per utilizzarle.

Il primo limite si avverte particolarmente quando le applicazioni gestite sono molte, mentre il dover conoscere bene uno strumento per poterlo utilizzare è controproducente nell'ambito di un gruppo, in quanto potrebbe portare alcune persone a non servirsene. I wizard, oltre a velocizzare il lavoro, aiutano ad osservare determinati standard, sia di interfaccia grafica che di codifica. Le aggiunte vere e proprie, o *add-in*, si classificano in

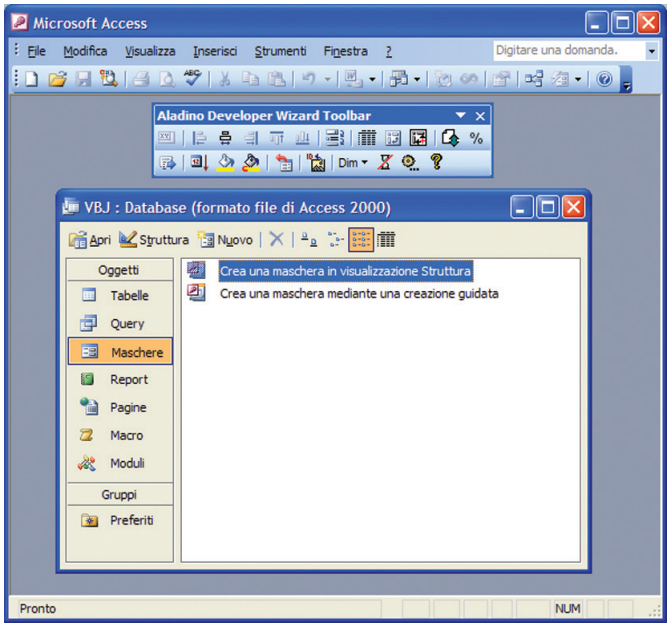


Figura 1 La barra degli strumenti "Aladino Developer Wizard Toolbar"

- *Menu componenti aggiuntivi*: forniscono pulsanti o menu per funzionalità aggiuntive e si trovano sotto Strumenti\Componenti aggiuntivi;
- *Creazioni guidate*: vengono lanciate creando degli oggetti, come le maschere;
- *Creazioni guidate di controlli*: sono gli aiuti lanciati automaticamente alla creazione di un nuovo controllo;
- *Generatori*: vengono lanciati dal pulsante "Genera" presente sul foglio delle proprietà di un oggetto ed aiutano ad impostare la proprietà a cui appartengono.

Inoltre si possono utilizzare aggiunte anche per l'editor VBE. Vediamo adesso come costruire una barra degli strumenti come menu componente aggiuntivo. Per maggiori dettagli e per gli altri tipi di aggiunte, si veda [1].

Creare ed utilizzare un add-in

Il processo per creare un menu componente aggiuntivo è poco più complicato di quello che serve per generare un'applicazione. Creiamo il database VBJWiz.mdb. Tramite il menu File/Proprietà scriviamo nei campi

- Titolo: "Il nostro primo add-in"
- Società: "VBJ"
- Commenti: "Versione 1.0 rilasciata il 1 marzo 2005"

Impostiamo una tabella con i campi revisione di tipo testo e descrizione di tipo memo. Salviamola con il nome "tblHistory" e lasciamo che Microsoft Access imposti per noi un campo contatore. Adesso creiamo su questa tabella, tramite l'autocomposizione, una maschera standard. Salviamola con il nome "frmHistory". Creiamo un modulo in cui inseriamo il codice riportato nel **Listato 1**.

Salviamo il modulo con il nome "modStart". Tramite Strumenti\Proprietà di Access<numero versione>, impostiamo il nome del progetto a "VBJWizard". Creiamo una toolbar chiamandola "VBJToolbar". Aggiungiamo dalla sezione Comandi\Categorie\Struttura Maschera\Report il pulsante "Allinea a sinistra". Duplichiamo il pulsante e impostiamo sul duplicato, tramite tasto destro, Proprietà:

- Didascalia: "Visualizza la data"
- Azione: =VBJWizMiaFunzione()

Volendo, possiamo cambiarne l'icona. Chiudiamo il database e rinominiamo il file in "VBJWiz.mda", che è l'estensione standard per le aggiunte.

Listato 1 Il codice da inserire nel modulo dell'add–in

```
Option Compare Database
Option Explicit

Function VBJWizStart()
    DoCmd.OpenForm «frmHistory»
    MsgBox "Il Wizard è partito!", vbInformation, "VBJ Wizard"
End Function

Function VBJWizMiaFunzione()
    Dim str As String
    Select Case Application.CurrentProject.ProjectType
        Case acADP
            str = "Il wizard sta lavorando per un file .adp."
        Case acMDB
            str = "Il wizard sta lavorando per un file .mdb."
        Case Else
        str "Impossibile determinare il tipo di progetto."
    End Select
    MsgBox str & vbCrLf & "Oggi è il " & Date, vbInformation, "VBJ Wizard"
End Function
```

Adesso dobbiamo creare una tabella che consenta ad Access di recuperare le informazioni di installazione della nostra aggiunta. Tramite il menu Strumenti\Opzioni\Visualizzazione, abilitare l'opzione "Oggetti di sistema" che consente la visualizzazione degli oggetti di sistema. Creiamo una tabella con i seguenti campi:

- Subkey (testo -255 caratteri)
- Type (Intero lungo)
- ValName (testo -255 caratteri)
- Value (testo -255 caratteri)

Salviamo la tabella con il nome "UsysRegInfo" ed aggiungiamo quattro record con le informazioni riportate in **Tabella 1**.

Il campo SubKey è uguale per tutti e quattro i record; se utilizzate Access 2003 il valore giusto è 11.0, per Access XP è 10.0 e per Access 2000 è 9.0. Nel campo value del secondo record è riportata la funzione che verrà lanciata al caricamento dell'aggiunta e nel terzo record il nome del file contenente la nostra aggiunta.

Come si nota, l'aggiunta potrebbe essere memorizzata in qualsiasi cartella, sarebbe sufficiente sostituire la keyword *ACCDIR* con il percorso completo. Il quarto record è in realtà opzionale: se non viene aggiunto, l'add-in sarà disponibile solo per i database .mdb.

Se il valore del campo value è impostato a 1, l'add-in sarà disponibile solo per i database .mdb; 2 per rendere disponibile l'add-in per i file di progetto .adp; 3 rende disponibile l'add-in per entrambi i tipi di file. In realtà la tabella USysRegInfo consente molte altre possibilità: per maggiori dettagli vedere [2]. Copiamo il file VBJWiz.mda in C:\Documents and Settings\<nome utente>\Dati applicazioni\Microsoft\AddIns. È ovviamente possibile creare il database già nella cartella di destinazione, ma conviene copiarvi solo le versioni dell'add-in già collaudate e lasciare quelle in sviluppo in una cartella separata. Creiamo ora un database che si chiama "VBJ.mdb". Tramite il menu Strumenti\Componenti aggiuntivi\Gestione Componenti aggiuntivi, troverete tra le altre eventuali aggiunte, quella appena creata. Fatevi un doppio click. Se non succede nulla, allora è tutto a posto :-). Aprite nuovamente Strumenti\Componenti aggiuntivi e selezionate la nuova opzione VBJ Toolbar. Ecco il vostro Wizard che vi saluta. Aggiungendo alla funzione VBJWizStart come prima riga

```
Function VBJWizStart()
    CommandBars("VBJToolbar").Visible = True
    MsgBox "Il Wizard è partito!", vbInformation, "VBJ
                                                  Wizard"
End Function
```

e riavviando il wizard dal database VBJ.mdb, noterete che compare la toolbar creata nell'add-in:

Figura 2 I pulsanti della "Aladino Developer Wizard Toolbar"

lavorando su una qualsiasi form, adesso avrete a portata di mano un pulsante per allineare i controlli, interamente gestito da Access ed uno per vedere l'ora, alla cui pressione viene lanciato il vostro codice.

Accedere agli oggetti da modificare

Un add-in è un database utilizzato da un altro database. Entrambi hanno, ad esempio, l'insieme delle maschere.

Per far riferimento agli oggetti del database in sviluppo, utilizzeremo CurrentProject, mentre con CodeProject accederemo a quello dell'add-in. Apriamo VBJ.mdb e lanciamo il nostro add-in: dalla finestra di debug possiamo scrivere

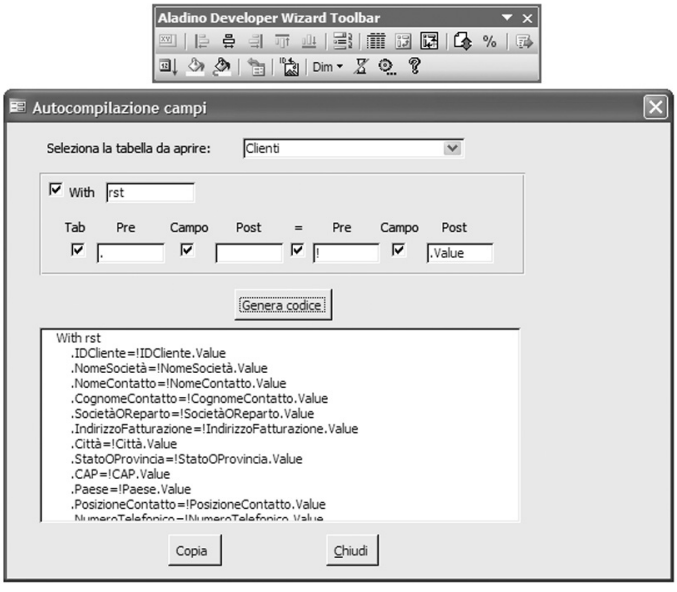


Figura 3 Compilare del codice basato sui campi di una tabella diventa velocissimo

```
?codeproject.AllForms(0).Name
```

ed otterremo *frmHistory*, che è il nome della sola maschera contenuta nel file VBJWiz.mda.

L'add-in può lavorare sulla maschera selezionata tramite le istruzioni

```
Screen.ActiveForm.SetFocus
Set frm = Screen.ActiveForm
```

Ovviamente, una volta che si è impostato il riferimento alla maschera, si può iterare sull'insieme dei controlli o navigarne il modello a oggetti.

E se volessimo aggiungere del codice che gestisce un evento? Il modello ad oggetti di Access ci aiuta: il codice descritto nel **Listato 2** aggiunge l'evento Click alla form selezionata in quel momento. Un particolare degno di nota è il metodo CreateEventProc, che crea automaticamente la dichiarazione della funzione con i parametri necessari e la posiziona al giusto posto nel modulo associato alla maschera.

In pratica, una volta creato il riferimento al modulo associato alla maschera, è sufficiente dichiarare l'evento da creare e poi inserirvi le righe di codice volute.

Progettare un add-in

L'esempio appena visto consente di capire che si possono costruire tabelle, form e codice che stanno in un database separato da quello su cui si lavora; come si può intuire, le potenzialità sono enormi. Per avere un'idea di cosa è possibile fare, è sufficiente considerare che i wizard di Access sono realizzati in Access e VBA!

Per progettare un add-in, è bene considerare i seguenti passi:

- Identificare quali sono le funzioni desiderate
- Dividere le funzioni già fornite da Access da quelle da creare; ad esempio, allineare a destra i controlli è una funzione già inclusa
- Realizzare le funzioni non incluse nel database dell'add-in e collaudarle
- Realizzare nell'add-in una toolbar o un menu che richiami tutte le funzioni volute
- Pubblicare l'add-in

Una menzione speciale merita la denominazione degli oggetti. Conviene cercare un prefisso inusuale sia per gli oggetti di database che per le funzioni.

Access è abbastanza intelligente da tenere separati gli spazi dei nomi; infatti, creando nel database VBJ.mdb una funzione che si chiama "VBJWizMiaFunzione", ossia uguale a quella presente nell'add-in, il pulsante della toolbar continuerà a richiamare quella giusta. Comunque, esplorando il progetto può essere utile riconoscere al volo dove una certa funzione sia stata implementata, grazie al suo nome.

Tabella 1 Informazioni da inserire nella tabella USysRegInfo

Subkey	Type	ValName	Value
HKEY_LOCAL_MACHINE\Software\Microsoft\Office\ 11.0\Access\Menu Add–Ins\&VBJ Toolbar	0		
HKEY_LOCAL_MACHINE\Software\Microsoft\Office\ 11.0\Access\Menu Add–Ins\& VBJ Toolbar	1	Expression	=VBJWizStart()
HKEY_LOCAL_MACHINE\Software\Microsoft\Office\ 11.0\Access\Menu Add–Ins\& VBJ Toolbar	1	Library	IACCDIR\ VBJWiz.mda
HKEY_LOCAL_MACHINE\Software\Microsoft\Office\ 11.0\Access\Menu Add–Ins\& VBJ Toolbar	4	Version	3

Per distribuire un add-in senza fornire l'accesso ai sorgenti, ci sono due possibilità: proteggere il progetto dalla visualizzazione tramite una password, oppure convertire il database in formato .MDE. Il file MDE impedisce l'accesso ai sorgenti, che sono stati fisicamente rimossi, ed anche alla struttura della maggior parte degli oggetti. I wizard di Access sono (purtroppo) generati proprio con questa tecnica. In alcuni casi, con le versioni per sviluppatori di Access, essi sono stati rilasciati anche in formato sorgente e, per chi vuole studiare determinati meccanismi, si tratta di una vera manna.

Listato 2 Codice che aggiunge l'evento Click

```
Function VBJWizAggiungiEvento()
    Dim frm As Form
    Dim ctl As Control
    Dim mdl As Module
    Dim lng As Long
    Dim strCode As String

    Screen.ActiveForm.SetFocus
    Set frm = Screen.ActiveForm

    Set mdl = frm.Module

    ' Aggiunge la procedura evento.
    lng = mdl.CreateEventProc("Click", Replace("Corpo", " ", "_"))

    ' Cancella una riga vuota
    mdl.DeleteLines lng + 1, 1

    strCode = vbTab & "' Evento aggiunto dal Wizard" & vbNewLine & vbTab &
                                "MsgBox ""Evento On_Click"", VbInformation"

    ' Inserisce il corpo della procedura.
    mdl.InsertLines lng + 1, strCode

End Function
```

Un caso reale

Per stuzzicare l'appetito, ho corredato quest'articolo dell'add-in che utilizzo per programmare, visibile in **Figura 1**. Sia perché ci sono delle funzioni a mio parere utili, sia perché è possibile trovare più materiale da analizzare che in un esempio didattico. Partiamo dalla password per visualizzare il progetto che è "VBJ" (maiuscolo!). La password è utile per evitare che, lavorando sul database di applicazione, ci si trovi aperta ed espansa nell'albero del progetto anche la sezione relativa all'add-in. Nella tabella nascosta *USysRegInfo*, i record sono 9 invece che 3, per permettere all'add-in di essere utilizzato sotto Access 2000/2002/2003.

I moduli sono divisi per funzionalità. In *modInizializza* ci sono le funzioni di inizializzazione e di richiamo delle maschere. Il modulo *modAggiungiAMenu* contiene il codice per realizzare il menu a tendina che compare premendo il pulsante Dim della toolbar, mentre in *modAladinoWizard* si trovano le routine di automazione.

L'ultimo modulo, *modAPI*, contiene il codice basato su API per ridimensionare la finestra di Access.

Il wizard è utilizzabile a fini didattici o per lo sviluppo di applicazioni. Le migliorie possibili sono molte.

Per esempio, il wizard non lavora correttamente, in alcuni casi, selezionando una sottomaschera. Oppure, talvolta,

richiede che sia abilitata la visualizzazione dell'intestazione e del pié di pagina maschera.

Le funzioni

In **Figura 2** è rappresentata la toolbar dell'Aladino Wizard con dei numeri per identificare ogni pulsante. I numeri 1,2,4,5,6,7,8,9,10 sono le comuni funzioni di formattazione di Access, riportate nella barra per praticità. Il pulsante 3 risolve il problema di centrare più controlli tra loro.

Tramite il pulsante 11, si imposta l'ordine di tabulazione dei soli controlli selezionati. Ad esempio, se si sta realizzando una maschera con due colonne, si seleziona la prima colonna di controlli e si preme il pulsante. Al termine, viene comunque visualizzata la finestra standard per l'ordine di tabulazione di Access.

Le opzioni 12, 13 e 14 sono utili nel realizzare maschere tabulari. Per vederle in azione, creare una maschera standard a colonne, poi premere 13: il wizard sposterà tutte le etichette nell'intestazione. Dopo aver impostato i campi alla larghezza voluta, premere 12: il wizard formatterà la maschera per voi. Selezionando un campo e premendo 14 verrà creato, nel pié di pagina, un campo con origine *=Somma(<nomecampo>)*. La differenza rispetto a quelle create da Access sta nel poter modificare la larghezza di un singolo campo e riformattare correttamente tutta la maschera premendo un solo tasto.

Il 15 serve nelle maschere continue o in quelle visualizzate in modalità a foglio dati. Introduce del codice che intercetta la pressione della freccia in basso ed in alto e le trasforma in un movimento in avanti o indietro di un record. Il codice associato a questo pulsante dimostra come sia facile inserire del codice associato ad un controllo nel nostro progetto. Il tasto 16 associa a tutti i campi percentuale una routine che, se il valore inserito è maggiore o uguale di 1, lo divide per 100. In pratica, scrivendo 20 nel campo, la routine lo trasformerà in 0,20, che Access vede come 20%.

Il tasto 17 apre una maschera che permette di creare del codice basato sui campi di una tabella del database locale, come si vede nella **Figura 3**. Il tasto 18 aggiunge una routine che visualizza un calendario, utile sui campi di tipo data. Se la provate, non funzionerà, semplicemente perché il codice

```
app.SelezionaData Screen.ActiveControl, Me.hwnd ' Apre il
                                                calendario
```

fa riferimento ad una routine di una mia libreria. Ho ritenuto opportuno includerla perché mostra come si possa automatizzare lo sviluppo di applicazioni, definendo alcune routine all'interno di una libreria ed applicandole facendo uso di un wizard. Le opzioni 19 e 20 colorano un campo con il colore predefinito per il collegamento. Nello standard che ho adottato, ogni campo dove funziona il doppio click è colorato con un giallo chiaro. Con il tasto 21 viene generato il codice che apre una maschera con determinati criteri. Ad esempio, in un ordine ho l'IDAnagrafica: volendo aprire la maschera frmAnagrafica selezionando l'IDAnagrafica che ho nel codice, il wizard scrive il codice giusto. Il tasto 22 offre una funzione carina. Permette infatti di scegliere tra le migliaia di icone fornite con Access. Ogni icona ha un ID: una volta saputo l'ID, con poche righe di codice la si può impostare. Tutte le icone della toolbar sono state scelte in questo modo. Un esempio è la **Figura 4**. Nella maschera è riportato anche lo schema di codice utile per raggiungere il controllo di cui cambiare l'icona. L'opzione 23 consente di dimensionare la finestra di Access in formati standard. Si rivela utile per collaudare un'applicazione all'effettiva risoluzione alla quale verrà utilizzata, per evita-

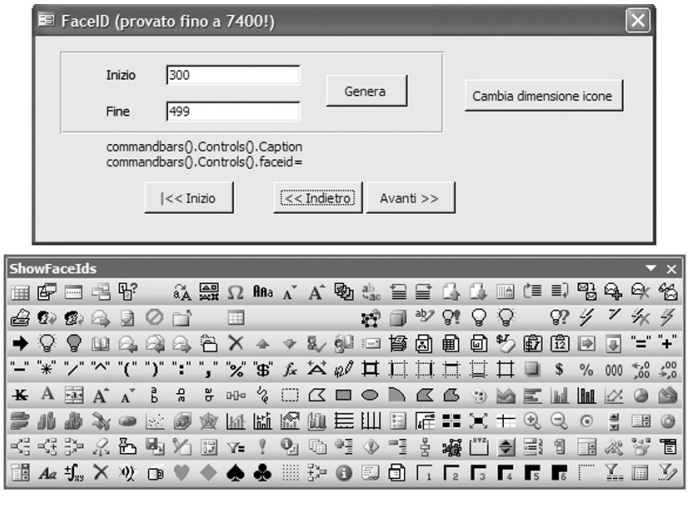


Figura 4 Uno strumento per scegliere tra le migliaia di icone presenti in Access

Listato 3 Codice per creare il menu a discesa nella barra degli strumenti

```
Public Sub InsertDropDownx()
    Dim ctlCombo As CommandBarComboBox
    Set ctlCombo = CommandBars("Aladino Developer Wizard Toolbar").Controls.
                                                    Add(msoControlDropdown)
    With ctlCombo
        .BeginGroup = True
        .AddItem ("640x480")
        .AddItem ("800x600")
        .AddItem ("1024x768")
        .AddItem ("1280x1024")
        .AddItem ("Massimizza")
        .BeginGroup = True
        .AddItem ("Dimensione massima")
        .Caption = "Dimensione finestra di Access"
        .Tag = "DimensioneFinestraAccess"
        .Width = 120
    End With
End Sub
```

re che maschere sviluppate magari a 1280x1024 non entrino nello schermo dell'utente che – sic – lavora ancora a 800x600. Per creare il menu a discesa nella barra degli strumenti viene utilizzata la routine descritta nel **Listato 3**, che dimostra ancora una volta la versatilità delle toolbar e la maneggevolezza delle stesse, fornita dal modello ad oggetti. L'opzione 24 disattiva la clessidra, rimasta attiva dopo un blocco del programma. Infine le opzioni 25 e 26 visualizzano rispettivamente le opzioni e le informazioni sul wizard.

Trucchi e problemi

La parte più complessa nella realizzazione di un wizard, consiste nel prevedere tutti i possibili ambienti in cui esso verrà impiegato. Una differenza rilevante è tra lavorare su un database (file .mdb) o su un progetto (file .adp); potrebbe essere necessario, ad esempio, generare codice SQL diverso nei due casi, anche solo variando i caratteri jolly che sono l'asterisco per l'mdb e la percentuale per l'adp. Possiamo dotare il wizard della capacità di discernere su quale tipo di file sta lavorando. Nella funzione VBJWizMiaFunzione viene stampato un messaggio che evidenzia il tipo di file correntemente aperto, grazie alla proprietà *Application.CurrentProject.ProjectType*.

Se si effettua un ciclo sull'insieme dei controlli, è necessario assicurarsi di trattare correttamente i diversi tipi di oggetti. Ad esempio, nel mio wizard, se in una maschera aggiungete una riga e provate la funzione 12, otterrete l'errore di "Proprietà o metodo non supportati dall'oggetto". Ovviamente non ho mai la necessità di formattare maschere con righe nel mezzo, ma se volessi commercializzare il wizard, dovrei eliminare anomalie simili. Quando si lavora su maschere, è necessario discriminare se il controllo selezionato dall'utente si trovi su una maschera o su una sottomaschera, impostando correttamente i riferimenti.

La gestione degli errori riveste una parte fondamentale; intercettare correttamente un errore e identificare la routine dove esso si è verificato è fondamentale per motivi diversi, sia in fase di sviluppo che in produzione.

Durante il collaudo dell'add-in, è importante identificare la parte di codice che produce l'errore, poiché può essere già complesso riprodurre la situazione in cui esso si verifica. Si consideri un ambiguo messaggio tipo "Impossibile trovare il nome di funzione immesso nell'espressione". Senza altre indicazioni, può trattarsi del fatto che Access non riesce a richiamare la funzione specificata nel pulsante, oppure può trattarsi di un errore generato nella funzione richiamata. Gestire un errore in produzione, consente probabilmente di proseguire il lavoro senza problemi. Se l'errore non è gestito, l'utente potrebbe interrompere l'esecuzione del codice distruggendo così eventuali variabili globali. A quel punto, il wizard dovrebbe essere rilanciato.

Conclusioni

Se sviluppate spesso applicazioni in Access e non avete mai creato un add-in, è il momento buono per farlo. Create il minimo indispensabile ed usatelo: in breve comincerete ad arricchirlo di funzioni. L'add-in allegato è un buon punto di partenza. Inoltre, se sviluppate in un gruppo, l'adozione di un add-in comune a tutti i programmatori aiuta a mantenere degli standard di programmazione. Invito chiunque abbia qualche buona idea da sviluppare a contattare me o la redazione.

Bibliografia & Riferimenti

[1] Ken Getz, Paul Litwin, Mike Gilbert - "Access 2000 Manuale di programmazione", Jackson Libri, 2000

[2] "Creating the USysRegInfo Table for a Microsoft Access 2000 Add-In ", http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnacc2k/html/usysregacc2k.asp

Stored Procedure sotto controllo

L'architettura a due livelli è tuttora il tipo di architettura più comune nelle applicazioni multi–utente, e le stored procedure sono lo strumento favorito nella sua implementazione fisica. L'uso di stored procedure solleva necessariamente delle problematiche legate alla consistenza dei dati e alla gestione degli errori con T–SQL.

di Francesco Quaratino

L'introduzione delle stored procedure (SP) ha permesso di spostare buona parte della logica dell'applicazione nel DBMS, apportando notevoli benefici nello sviluppo e nella manutenzione dell'applicazione stessa.
L'aspetto forse più evidente delle stored procedure è quello prestazionale, poiché il codice in esso contenuto è compilato ed ottimizzato dal DBMS e lo stesso piano d'esecuzione della SP è riusato da altre sessioni concorrenti.
L'impiego più efficiente della rete rende le architetture basate su SP notevolmente scalabili. Infatti, al momento della richiesta di esecuzione di una SP, il client invia al server unicamente i parametri di input della SP piuttosto che l'intera istruzione T-SQL, producendo in tal modo un traffico di rete minore. Dal punto di vista ingegneristico, le SP garantiscono una forte centralizzazione della manutenzione della logica dell'applicazione e una distribuzione degli aggiornamenti a dir poco immediata, dal momento che modificare il codice di una SP non richiede alcun intervento sul client, il quale si aspetta soltanto di trovare lo stesso nome di procedura e stessi nomi e tipi di parametri nel database a cui è connesso.

Attraverso le SP è possibile anche implementare un meccanismo di sicurezza dei dati – incapsulando definitivamente la logica racchiusa nel codice delle SP – evitando così l'interazione impropria con i dati da parte degli utenti non autorizzati. Infatti, è possibile concedere ad un utente di database il diritto d'esecuzione di una SP indipendentemente dai diritti d'accesso alle tabelle interessate, assegnati allo stesso utente. SQL Server permette di avere fino a 32 livelli di nidificazione (*nesting*) di SP che consentono di incapsulare in ogni procedura singole operazioni secondo i dettami della progettazione strutturata.

È molto meglio scrivere N procedure articolate in una struttura "chiamante/chiamato" piuttosto che scrivere un'unica procedura che racchiude in sé l'intero codice T-SQL.

La prima soluzione è più vantaggiosa per la creazione, il test e la manutenzione e garantisce il riuso più oculato dei piani di esecuzione da parte del DBMS.

Ad ogni modo occorre fare sufficiente attenzione alla consistenza dei dati trattati nella SP, poiché al di là dei vincoli d'integrità referenziale applicati nel database, un controllo superficiale degli errori generati nell'esecuzione di una stored procedure può dar luogo a transazioni non consistenti.

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*

Listato 1 Esempio dell'uso di XACT_ABORT ON per causare il roll back automatico del– la transazione al verificarsi di un errore

```
SET XACT_ABORT ON

BEGIN TRAN

/*Istruzione 1*/
INSERT INTO authors (au_id, au_lname, au_fname,
                                    phone, contract)
  VALUES ('000-00-0000','Alberto','Grossi','801
                                       826-0752',1)

/*Istruzione 2*/
INSERT INTO authors (au_id, au_lname, au_fname,
                                    phone, contract)
  VALUES ('000-00-0000','Franco','Cirri','800 800-
                                         0751',1)

COMMIT TRAN

/*Istruzione 3*/
UPDATE authors SET contract=0
GO
```

Lo scopo del presente articolo è analizzare gli strumenti a disposizione di un DBA SQL Server per la gestione degli errori in T-SQL, fino a giungere a un modello efficace di gestione e diagnostica degli errori nelle SP.

Transazioni e consistenza dei dati

Nel trattare il concetto di consistenza dei dati di un database, il pensiero va innanzitutto alla definizione dei vincoli di integrità referenziale e in secondo luogo alla salvaguardia delle transazioni implicite ed esplicite.

Quelle implicite sono rappresentate dalle singole istruzioni di modifica dei dati (INSERT, UPDATE, DELETE). In presenza di transazione implicita, al verificarsi di un errore, sia esso dell'utente (ad esempio violazione di vincoli) o del sistema (ad esempio indisponibilità del sever), il DBMS si preoccupa di fare il *rollback* dell'istruzione, ovvero riportare lo stato dei dati all'istante immediatamente precedente all'esecuzione non riuscita, mentre se non ci sono intoppi sarà eseguito il *commit* della transazione.

In caso di transazione implicita, la gestione della transazione viene delegata al DBMS ed essa avviene in modo del tutto trasparente all'utente, mentre attraverso le transazioni esplicite il DBA può creare un'unica unità di lavoro (*atomic unit of work*) composta da istruzioni multiple che la logica dell'applicazione vuole che siano completate tutte insieme.

Le transazioni esplicite devono essere dichiarate esplicitamente, racchiudendo le istruzioni coinvolte nella transazione tra un'istruzione di BEGIN TRAN[SACTION] (che ne determina l'inizio) e una COMMIT TRAN[SACTION] o ROLLBACK TRAN[SACTION] a seconda che si voglia rispettivamente dare effetto o meno a tutte le istruzioni precedenti che seguono l'istruzione di BEGIN TRAN.

Ad eccezione degli errori di sistema (*system failure*) che causano un rollback della transazione (sia essa implicita che esplicita), il DBA dovrà preoccuparsi di controllare il verificarsi dell'errore sollevato durante una qualsiasi delle istruzioni contenute nella transazione esplicita, per ordinarne il rollback; altrimenti sarà stato del tutto inutile aver esplicitato la transazione, poiché l'esecuzione del codice T-SQL seguente continuerà ad essere processato.

In altre parole, nell'ambito di un batch o di una transazione, un errore non di sistema causa il fallimento della sola istruzione che genera l'errore senza condizionare le altre istruzioni.

Anche le SP non si sottraggono a questo aspetto spesso frainteso: il fatto che la transazione sia per definizione una singola unità di lavoro lato server, non vuol dire che un errore generato da un'istruzione causi il rollback automatico della transazione.

> Un controllo superficiale degli errori nelle stored procedure può dar luogo a transazioni non consistenti

Tuttavia, impostando a ON la variabile XACT_ABORT, si ottiene questo comportamento da SQL Server (vedi **Listato 1**).

Attenzione, però, ai batch che includono più transazioni, in quanto XACT_ABORT influenza l'intero batch interrompendo l'esecuzione del batch al verificarsi del primo errore.

Di conseguenza ogni altra transazione che seguisse quella al cui interno è stato generato l'errore non sarebbe mai eseguita.

Nell'esempio del **Listato 1**, supponendo che l'Istruzione_2 violi un vincolo di chiave prima-

ria, ne risulterebbe che non solo l'Istruzione 1 non avrebbe effetto, ma anche che l'Istruzione 3 non sarebbe processata.

> Con le transazioni esplicite il DBA può creare delle atomic unit of work composte da istruzioni multiple

Se a prima vista XACT_ABORT può sembrare una soluzione veloce ed efficace da implementare nelle nostre SP, essa non è la migliore, poiché non consente un controllo dell'errore e quindi non dà la possibilità di specificare una reazione a un errore specifico.

Inoltre, tale impostazione non ha effetto sugli errori di sintassi generati a run-time dall'esecuzione di una stringa costruita dinamicamente attraverso il comando EXECUTE ('Stringa_Sql').

Si noti come tali errori sfuggano al controllo sintattico e si verifichino unicamente durante l'esecuzione.

Nested Transaction

Ogni SP può contenere infinite chiamate ad altre SP, mentre può chiamare sé stessa ricorsivamente fino a raggiungere 32 livelli di nidificazione.

Anche le transazioni possono essere nidificate, sia attraverso SP nidificate che in un'unica sequenza di istruzioni T-SQL inclusa in uno o più batch.

Un qualsiasi ROLLBACK TRAN provoca il roll back a tutti i livelli della struttura delle transazioni nidificate, mentre soltanto il COMMIT TRAN del blocco più esterno dà valore a tutti i commit di qualsiasi livello.

Per tenere sotto controllo i livelli di transazione è possibile interrogare la funzione di sistema @@TRANCOUNT che viene incrementata ad ogni BEGIN TRAN e decrementata da ogni COMMIT o ROLLBACK TRAN.

Si noti che solo quando @@TRANCOUNT è uguale a 1 ha effetto il commit. Dopo un ROLLBACK TRAN sarà indispensabile controllare che il valore restituito da @@TRANCOUNT sia diverso da 0, per evitare di effettuare ulteriori COMMIT o ROLLBACK TRAN delle transazione esterne – che altimenti solleverebbero un errore numero 3903 (che denota appunto l'assenza di una transazione aperta).

Allo scopo di rendere più leggibile una struttura di transazioni nidificate, è buona norma dare un nome a ciascuna transazione.

Se si sceglie di seguire questa norma, sarà bene ricordare che è possibile specificare il nome della transazione per ogni BEGIN e COMMIT TRAN e solo per il ROLLBACK TRAN più esterno, altrimenti verrà generato un errore numero 6401.

Un modello di gestione degli errori

Un'alternativa più laboriosa ed efficace a XACT_ABORT ON è rappresentata dall'uso con-

Listato 2 Esempio di gestione degli errori nelle stored procedure nidificate

```
/* Procedura chiamata */
CREATE PROC sp_B
AS
DECLARE @ERROR int
BEGIN
  BEGIN TRAN B
  INSERT INTO …
  SET @ERROR = @@ERROR
  IF @ERROR <> 0
    GOTO ERRORS
  COMMIT TRAN B
  RETURN 0
ERRORS:
  RAISERROR ('Procedure sp_B failed',16,1) WITH LOG
  COMMIT TRAN B
  RETURN @ERROR
END
GO

/* Procedura chiamante */
CREATE PROC sp_A
AS
DECLARE @ERROR int
BEGIN
  BEGIN TRAN A
  EXEC @ERROR = sp_B
  IF @ERROR <> 0
    GOTO ERRORS
  EXEC @ERROR = sp_C
  IF @ERROR <> 0
    GOTO ERRORS
  COMMIT TRAN A
  RETURN 0
ERRORS:
  ROLLBACK TRAN A
  RAISERROR ('Procedure sp_A failed',16,1) WITH LOG
  RETURN @ERROR
END
GO
```

giunto della funzione @@ERROR e delle istruzioni RETURN, GOTO e RAISERROR.

La funzione di sistema @@ERROR ha lo scopo di catturare il numero dell'errore intercorso nell'esecuzione di un'istruzione T-SQL. Se il valore restituito è uguale a 0, significa che è andato tutto liscio.

Nel valutare il valore restituito da @@ERROR occorre fare attenzione al fatto che il suo valore viene alterato dall'esecuzione di una qualsiasi altra istruzione T-SQL che la segue (per intenderci anche un IF).

Ne consegue che è buona norma assegnarne sempre il valore a una variabile, in modo da poterne valutare il contenuto anche in seguito, e soprattutto interrogare la funzione subito dopo ogni istruzione di interazione col database:

```
INSERT INTO …
SELECT @ERROR = @@ERROR
IF @ERROR <> 0
   ROLLBACK TRAN
```

RETURN ordina l'uscita incondizionata da un batch e restituisce alla procedura chiamante il numero intero specificato (esempio RETURN 1). GOTO fa saltare il flusso d'esecuzione ad un'etichetta specificata, mentre RAISERROR restituisce al client un messaggio d'errore definito dall'utente, con la possibilità di registrarlo nel log degli eventi del sistema.

Supponiamo di avere una procedura sp_A che chiama in ordine sp_B e sp_C. Per semplicità le procedure chiamate si intendono identiche (vedi **Listato 2**).

> XACT_ABORT ON causa il roll back automatico della transazione al verificarsi di un errore

Il modello di gestione errori proposto presenta, dal punto di vista delle transazioni, una transazione esplicita in ciascuna delle tre SP ed un solo ROLLBACK TRAN nella sp_A più esterna, mentre le sp_B e sp_C interne eseguono in ogni caso un COMMIT TRAN, che sortirà il solo effetto di decrementare il numero di transazioni attive.

Quindi, sia il roll back che il commit sono delegati alla sp_A più esterna, dove il flusso di esecuzione giunge per mezzo dei due RETURN presenti nelle procedure chiamate. Inoltre, i RETURN comunicano il numero di errore a sp_A, che è quindi in grado di decidere la sorte della transazione dell'intera struttura nidificata.

Si noti il modo in cui viene recuperato il valore restituito dal RETURN durante l'esecuzione delle procedure interne – che avviene attraverso il comando EXEC[UTE].

In presenza di un errore, RAISERROR consente la comunicazione al client di un messaggio ad-hoc, definito dall'utente, e la registrazione dello stesso nel log degli eventi, attraverso l'opzione WITH OPTION.

In questo modo fornisce al DBA uno strumento di diagnostica veloce e facile da implementare.

Conclusioni

L'inerzia della conoscenza è uno dei mali incurabili dell'informatica.

In altre parole, quanto più consolidata è una tecnologia, tanto più difficile risulta modificarla, anche quando i benefici della neo-tecnologia sono palesi e evidenti anche alle menti più conservatrici.

L'architettura a due livelli è una dimostrazione esemplare di tale fenomeno, tenuto conto che è sicuramente il tipo di architettura software più comune nelle applicazioni multi-utente orientate ai dati.

Sovente le stored procedure non vengono sfruttate appieno ma relegate ad assolvere funzioni particolarmente laboriose e lunghe senza poterne sfruttare le qualità prestazionali, ad esempio ignorando la possibilità di nidificazione.

Inoltre, laddove si fa un uso considerevole di SP, è bene non sottovalutare il controllo degli errori che potrebbero causare gravi inconsistenze dei dati e che sarebbero difficilmente rintracciabili senza un metodo di diagnostica efficace.

Bibliografia

[1] K. Delaney – "Inside Microsoft SQL Server 2000", Mondadori Informatica, 2000

[2] L. Braidi – "Database Design", Tecniche Nuove, 2004

[3] G. Drapers – "IF Statements and Stored Procedures Performance", SQL Server Magazine/ January 2005 – InstantDoc ID 44717

Logica applicativa in programmi VB.NET

Prima puntata

Ogni gruppo di sviluppo che si occupi della realizzazione di applicazioni gestionali ha prima o poi sentito l'esigenza di razionalizzare il proprio metodo di lavoro

di Lorenzo Vandoni

.NET fornisce molti strumenti per scrivere programmi con funzionalità di accesso ai dati, ma non offre un vero e proprio framework per la realizzazione di applicazioni gestionali.
Lo stesso vale per tutti gli altri strumenti di sviluppo più diffusi, come Java e Delphi, prova ne sia la proliferazione di strumenti CASE o framework specifici, come ad esempio [1] e [2].

Chi sviluppa abitualmente applicazioni gestionali è abituato a dover implementare spesso funzionalità simili tra loro, come classi per la gestione della logica applicativa, finestre per l'inserimento dei dati, o report riassuntivi.

Quasi tutte le software house che lavorano in questo ambito hanno creato insiemi di componenti, più o meno evoluti, allo scopo di generalizzare queste funzionalità e diminuire quindi il tempo necessario per sviluppare programmi simili.

In questo articolo, che può essere considerato come l'ideale continuazione di [3], verrà discussa una possibile soluzione per la realizzazione di classi .NET che implementino la logica applicativa del programma.

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'indirizzo vandoni@infomedia.it.*

Considerazioni preliminari

In [3], prendendo spunto da una discussione relativa ai paradigmi di accesso ai dati tipici dei linguaggi di programmazione object-oriented, si era arrivati a ipotizzare la realizzazione di un'estensione ad ADO.NET per creare delle classi polimorfiche che facilitassero lo sviluppo di funzionalità lato server.

Riassumo nei punti seguenti le conclusioni a cui si era giunti:

- il problema dell'*impedance mismatch*, tipico delle applicazioni object-oriented, consiste nel fatto che oggetti intrinsecamente complessi, come ad esempio il modello di un'automobile in un programma CAD, debbano essere rappresentati in memoria utilizzando strutture dati diverse da quelle usate per rappresentare gli stessi oggetti in un DBMS relazionale, con la conseguente necessità di scrivere algoritmi di traduzione per convertire la rappresentazione di uno stesso oggetto tra due formati differenti;

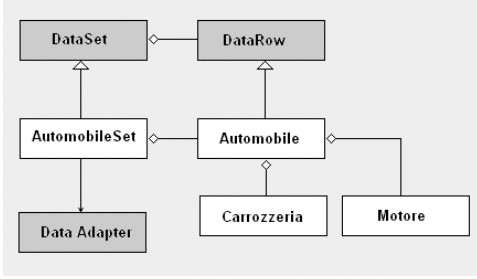


Figura 1 Architettura delle classi necessarie per implementare la logica applicativa

- il modo migliore per affrontare il problema – supponendo che la scelta del DBMS relazionale sia vincolante, e tralasciando, quindi, l'ipotesi di adottare strumenti di tipo diverso per gestire la persistenza dei dati – è quello di incapsularlo in uno strato software omogeneo (middleware), che si preoccupi di convertire i dati da una rappresentazione object-oriented al modello relazionale, e viceversa;

- questo strato di software può essere realizzato implementando un insieme di classi, come ad esempio *Automobile* (**Listato 1**), che forniscano tutti i metodi necessari per espletare tutte le operazioni necessarie per il reperimento, il salvataggio e la ricerca dei dati;

- questa soluzione ha il vantaggio di fornire un corretto incapsulamento del problema relativo all'accesso ai dati, e viene spesso citata, in alcuni testi rivolti ai programmatori VB6, come "il" motivo principale per cui può avere senso creare delle proprie classi, in applicazioni di tipo gestionale. D'altra parte, questa soluzione provoca la creazione di un numero eccessivo di classi del tutto simili tra loro, che in molti casi, soprattutto nel caso di dati strutturalmente semplici come quelli relativi a clienti e fatture, richiedono un'implementazione del tutto banale e ripetitiva;

- una possibile alternativa consiste nella creazione di un'unica classe *Tabella*, che possa essere genericamente utilizzata per accedere al contenuto di qualsiasi tabella sul database.

  Questo approccio è di fatto quello adottato da ADO e ADO.NET, sia pure con modalità diverse, ma risolve il problema dell'*impedance mismatch*, perché obbliga a inserire di volta in volta i comandi SQL che implementano le operazioni di caricamento e salvataggio dei dati.

Considerando pregi e difetti di ciascuna soluzione, si era infine ipotizzata l'adozione di un approccio intermedio, che permettesse di comportarsi in modo diverso a seconda del livello di complessità dell'oggetto rappresentato, utilizzando una classe base *Tabella* nei casi più semplici, e creando delle classi specifiche nei casi più complessi.

Considerazioni progettuali

L'obiettivo che ci poniamo è quello di progettare una valida struttura per la realizzazione di classi .NET che implementino la logica applicativa di un programma, indipendentemente dal fatto che si tratti di un'applicazione Windows o Web, evitando di scrivere codice ripetitivo e ridondante.

Il fatto di limitare la trattazione alla sola logica applicativa significa che il discorso che segue non prenderà in considerazione gli aspetti relativi all'interfaccia grafica o all'interazione con l'utente.

> Chi sviluppa abitualmente applicazioni gestionali è abituato a dover implementare spesso funzionalità simili tra loro

Considerando il classico paradigma MVC (Model-View-Controller), questo significa che prenderemo in esame solo la realizzazione del modello.

La trattazione che segue sarà inoltre molto .NET-oriented.

La progettazione cioè prenderà in considerazione l'attuale struttura delle classi del framework, e in particolare quella di ADO.NET.

Questo non significa però che le stesse considerazioni non possano essere riportate su altri linguaggi e librerie.

Architettura di ADO.NET

L'architettura di ADO.NET è molto più ricca rispetto a quella di ADO, grazie anche alla disponibilità di meccanismi come ereditarietà e il polimorfismo.

Questo rende possibile progettare le nostre classi applicative come estensioni delle classi del framework, ereditandone tutte le funzionalità di base. Potremo quindi cercare all'interno del framework una classe che costituisca un'astrazione del concetto di tabella, e utilizzarla come classe base per le classi applicative più specifiche, come ad esempio *Automobile*.

La struttura gerarchica delle classi di ADO.NET, però, prevede anche una separazione di responsabilità fra le classi dedicate al reperimento dei dati, che fanno parte dei *managed provider*, come ad esempio *DataReader* e *DataAdapter*, e le classi dedicate al mantenimento degli stessi dati in memoria, come *DataTable* e *DataSet*.

Questa struttura rende meno ovvie alcune scelte di progettazione.

Tornando a considerare il **Listato 1**, possiamo vedere come l'ipotesi di partenza fosse quella di creare delle classi che fornissero sia dei metodi generici per permettere l'accesso ai dati, come *Leggi* e *Salva*, sia dei metodi specifici, come *Capovolgi* e *Ruota*. In questa ipotesi, si era sfruttata la possibilità di definire dei metodi *Shared* – o *statici* – per implementare le operazioni che dovevano essere applicati all'insieme di oggetti, e non al singolo oggetto (per esempio, il metodo *Trova* applica una condizione all'insieme delle automobili per estrarre un'unica automobile).

La decisione di cercare di sfruttare ed estendere ADO.NET, piuttosto che creare una gerarchia di classi ad-hoc, ci costringe a rivedere questa ipotesi.

La nostra classe *Automobile*, cioè, dovrà essere costruita in modo da adattarsi all'architettura sottostante.

Progettazione

La soluzione proposta è schematizzata in **Figura 1**. L'idea è la seguente:

- l'oggetto che rappresenta il disegno di un'automobile, al quale devono poter essere applicate operazioni specifiche quali *Capovolgi* e *Ruota* non è assimilabile ad un *DataSet*, che rappresenta una collezione di tabelle, ma piuttosto ad un *DataRow*, cioè ad una singola riga di una tabella.
  Per questo motivo, la classe *Automobile* verrà creata come sottoclasse di *DataRow;*

- un *DataSet* costituisce una rappresentazione in memoria di una struttura relazione costituita da un insieme di tabelle e dalle loro relazioni. Per questo motivo, è il candidato ideale per definire il mapping tra le due rappresentazioni dei dati, relazionale ed object oriented. La classe *AutomobileSet* viene definita come sottoclasse di *DataSet*, per implementare questo mapping nel caso specifico;

- un *Data Adapter* viene incapsulato all'interno della classe *AutomobileSet*, ed utilizzato per accedere ai dati sul DBMS.
  L'obiettivo progettuale, in questo caso, è quello di incapsulare all'interno di un'unica classe la logica di mapping, invece di lasciare all'applicazione client il compito di creare il *Data Adapter* necessario per accedere ai dati;

- la rappresentazione in memoria dei dati può essere costituita da altri oggetti collegati al disegno dell'automobile, come ad esempio *Carrozzeria* e *Motore*.
  La rappresentazione object-oriented di questi oggetti potrà essere diversa dalla corrispondente rappresentazione sul DBMS.

Listato 1 Scheletro di una classe che maschera l'accesso ai dati mantenuti in una tabella

```
Class Automobile
   'metodi specifici dell'oggetto
   Public Sub Ruota(nGradi As Integer)
      ...
   Public Sub Capovolgi()
      ...
   Public Sub Stampa()
      ...
   'metodi per la gestione del DBMS
   Public Sub Salva()
      ...
   Public Shared Sub Leggi(nObjId As Long)
                                        As Automobile
      ...
   Public Shared Sub Trova(sCondition As String) As
                                          Automobile
      ...
End Class
```

Vantaggi e svantaggi

Questo tipo di soluzione, senza pretendere di essere ottimale, offre diversi vantaggi. Anzitutto, si appoggia sull'architettura di ADO.NET invece di costruire una gerarchia di classi separate.

Questo consente di continuare a sfruttare tutti gli automatismi forniti da ambienti di sviluppo e classi di libreria, tra cui la generazione automatica del codice, e la possibilità di visualizzare i dati in griglie e altri dispositivi di interfaccia, e permette anche di non creare classi specifiche, utilizzando al loro posto le classi base *DataSet* e *DataRow*, in tutti i casi in cui gli oggetti siano strutturalmente semplici. Inoltre, consente di separare la logica applicativa dalla gestione dell'interfaccia utente, permettendo di creare delle classi che risultino facilmente riusabili in progetti Windows e Web. Infine, permette di risolvere il problema dell'impedance mismatch, incapsulando la logica di mapping in un'unica classe. Sicuramente, progettare e sviluppare un insieme di classi di questo tipo richiede più tempo rispetto ad un approccio più diretto e prototipale, come quello di implementare la logica di accesso ai dati direttamente all'interno delle form o pagine di interfaccia.

Come sempre accade, i benefici effetti di un'architettura più ordinata si faranno sentire nel tempo.

Conclusioni

La creazione di un insieme di classi che forniscano una gestione completa della logica applicativa può essere molto utile anche per la suddivisione del programma in più strati, o tier, e può semplificare il riuso di queste classi in applicazioni diverse.

In questo articolo abbiamo esaminato dal punto di vista progettuale una possibile soluzione a questo problema.

Nel prossimo articolo parleremo delle problematiche di implementazione, discutendo un esempio completo.

Bibliografia

[1] S.Visconti, "Realizzare una Web Application con il framework Struts", DEV n. 108, Giugno 2003

[2] L.Vandoni, "BLOBJ Business Logic Objects", CP n.125, Giugno 2003

[3] L.Vandoni, "L'accesso ai dati nei linguaggi object-oriented", VBJ n.60, Novembre 2004

TDO Typed Data Object

TDO è l'acronimo di Typed Data Object ("oggetto dati tipizzato"), un insieme di strumenti e metodologie di supporto per lo sviluppo di applicazioni basate su piattaforma Microsoft .NET. In questo articolo scopriremo come funziona e quali sono i vantaggi per lo sviluppatore.

di Andrea Ferendeles

TDO è l'acronimo di Typed Data Object ("oggetto dati tipizzato"), un insieme di strumenti e metodologie di supporto per lo sviluppo di applicazioni basate su piattaforma Microsoft .NET. TDO consente agli sviluppatori di applicazioni basate sul Framework Microsoft.NET (v. 1.0 e v. 1.1) di produrre, in modo completamente automatico, tutto il codice sorgente necessario (in C#), per la gestione di una base dati Microsoft SQL Server 7.0 e 2000. TDO è basato su ADO.NET ed è un generatore di codice sorgente Visual C# .NET.

"Anziché scrivere a priori codice generico per una generica base dati ed a posteriori codice specifico per la base dati specifica (operazione che richiede tempi non indifferenti rispetto all'intero ciclo di vita del software), dotiamoci a priori di uno strumento che scriva all'occorrenza il codice per quella base dati specifica". Questo si fa con TDOCodeGenerator.exe.

Inoltre è facile osservare come, nonostante ogni base dati abbia la propria struttura e le proprie caratteristiche, su tutte sia possibile compiere le stesse operazioni in modo parametrico (ricerca, inserimento, ecc…).

Se in aggiunta riusciamo ad ottimizzare tale codice ed a renderlo robusto, grazie anche alle caratteristiche intrinseche dei prodotti specifici utilizzati (MS SQL Server 2000 e .NET), allora possiamo portare a fattor comune tali accorgimenti e "usufruirne" ogni volta dal codice scritto automaticamente per la nostra base dati specifica, tramite System.Data.TDO.dll.

Andrea Ferendeles *ha iniziato ad appassionarsi di informatica dai tempi del VIC20. Ha sviluppato in BASIC, Logo, Turbo Pascal, C, C++, Visual Basic, fino ad arrivare ai linguaggi .NET, quali VB.NET e C#. È certificato MCP, MCAD, MSF, MCSD (.NET), MCDBA, MCT. Può essere contattato via email: aferendeles@infomedia.it.*

Da dove nasce l'idea TDO

"Ogni qual volta si deve realizzare un'applicazione a n livelli, che fa uso di una base dati, occorre scrivere codice per utilizzare questi dati."

Tale codice, in un'ottica Object Oriented e in applicazioni ad n strati, viene modellato in classi e incapsulato nello strato Data e successivamente utilizzato nello strato Business. Chiameremo d'ora in avanti "Data Object" il codice necessario alle operazioni di accesso ai dati.

Dalle osservazioni fatte durante il processo di sviluppo di una applicazione, si osserva che:

- il codice nello strato Business è sempre il frutto della logica delle esigenze del Cliente (Business Requirements), e tali esigenze sono diverse di volta in volta;
- il codice nello strato Data, è sempre il frutto delle operazioni che si compiono sui dati, e tali operazioni sono sempre concettualmente le stesse (ricerca, inserimento, modifica, ecc…);
- la struttura della base dati viene invece creata ad hoc per

supportare indirettamente le esigenze del Cliente e direttamente le esigenze dell'applicazione e quindi anch'essa cambia di volta in volta;

Se focalizziamo ora l'attenzione sullo strato dati, si osserva che tutto il codice che si renderà necessario scrivere, per consentire l'accesso a questi dati (Data Object) non risulta più essere legato direttamente alle esigenze del Cliente ma bensì risulta essere profondamente legato alla struttura della base dati stessa.Tale legame ci impone ogni volta di riscrivere il Data Object, per accedere a quella particolare base dati anziché ad un'altra.

Per ridurre i tempi di sviluppo, ci si è dunque dotati nel tempo di "facilitatori" e/o porzioni di codice "riusabile", che comunque necessitano di volta in volta di una inevitabile personalizzazione, dovuta proprio alla "tipicità" della base dati.

La nostra modesta esperienza ci ha insegnato negli anni che tali porzioni di codice riusabile, risultano spesso essere "asettiche", a volte poco performanti oppure molto astratte e dunque poco usabili (perché devono tenere in considerazioni diverse possibilità di utilizzo in futuro).

Di cosa si compone TDO

TDO è composto da 2 elementi:

System.Data.TDO.dll: una libreria, contenente Interfacce, Classi astratte, struct, eccetera, che inglobano le caratteristiche a fattor comune di accesso di una base dati MS SQL Server 7.0 e 2000. Tale libreria implementa al suo interno, tutte quelle ottimizzazioni e accorgimenti che vanno necessariamente tenuti in considerazioni ogni volta che si accede ai dati, come ad esempio problematiche di SQL Injection, ottimizzazione delle query, performance e scalabilità.

TDOCodeGenerator.exe: un'applicazione, utilizzabile sia a linea di comando che in modalità grafica, per la scrittura di un file C#, contenente codice sorgente specifico per una base dati MS SQL Server 2000.

Una volta eseguito TDOCodeGenerator su di una base dati specifica, viene prodotto un unico file di testo, contente codice C# con le seguenti caratteristiche:

- tutto il DataBase viene esposto, secondo il modello Object Oriented, attraverso una classe (tdohelperclass): il database è un oggetto che quindi può avere attributi ed operazioni;

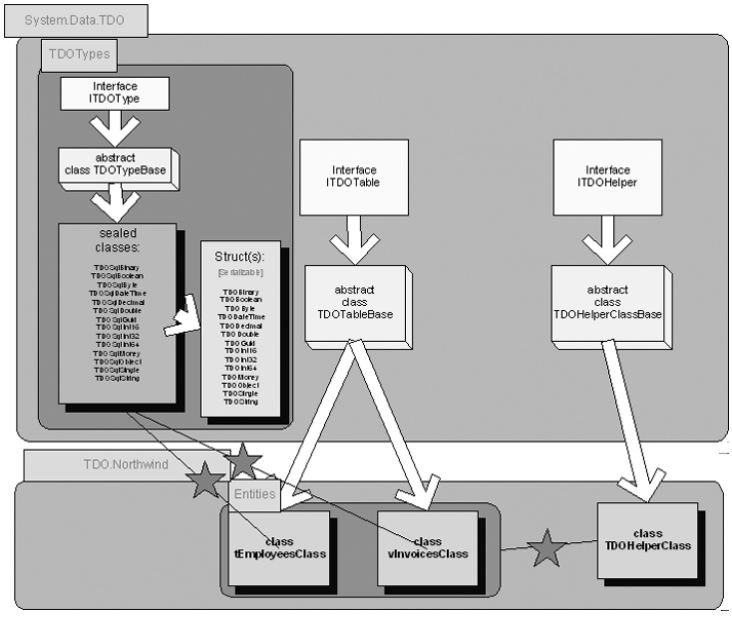


Figura 1 Architettura System.Data.TDO.dll

• attributi: ogni entità del database (Tabella, Vista) viene esposta come una classe C# che estende una classe base implementata nella libreria System.Data.TDO.dll; ogni entità avrà a sua volta attributi ed operazioni; attributi: ogni campo della tabella/vista viene esposto attraverso una property tipizzata; operazioni: ogni operazione possibile sull'entità (select, insert, update, delete) viene esposta come metodo.

• operazioni: ogni stored procedure, user function, table function e in-line table function viene esposta come una operazione (funzione C#) della classe tdohelperclass.

I Punti di forza di TDO

Tutto è tipizzato: tutti i campi delle tabelle e viste e tutti i parametri delle funzioni e stored procedure hanno nella base dati un proprio "tipo" (int, char, binary, ecc…).

Tali tipi vengono riesposti da TDO negli equivalenti tipi disponibili nel Framework .NET (int, string, byte[]).

Riduzione degli errori: essendo gli attributi del database tipizzati nella classe tdohelperclass, (e quindi non "object") i controlli sui tipi vengono fatti a compile-time dal compilatore .NET del linguaggio in uso.

Ciò consente una scoperta degli errori ed una conseguente eliminazione già in fase di stesura del codice e non più a run-time. Ad esempio, se dobbiamo modificare un campo di tipo "char" di un record in una tabella del database, potremo assegnare a tale campo solo un valore di tipo "string"; se dobbiamo richiamare una stored procedure che si aspetta un parametro di tipo "tinyint", potremo passare a tale metodo della classe tdohelperclass solo un valore di tipo "byte", ecc…

Facilità e velocità d'uso: TDO è facile ed estremamente veloce da usare anche per chi lo utilizza per la prima volta, perché ha una struttura chiara ed inequivocabile. Inoltre rispetto alla canonica stesura di codice che utilizza solo ADO.NET, si ha un risparmio di circa il 70% in termini di codice da scrivere con conseguente risparmio sui costi e tempi di sviluppo.

Ad esempio, tutte le classi che espongono tabelle del database iniziano per "t"; tutte le classi che espongono viste iniziano per "v"; tutti i metodi che espongono user-function iniziano per "f"; tutti i metodi che espongono SP iniziano per "p"; se dobbiamo invocare una SP con 1 parametro, in ADO.NET dovrei scrivere circa 8 righe di codice (dichiarazione del Command, CommandType, CommandText, Connection utilizzata, Nome del Parametro, Tipo del Parametro, Direzione e Valore) mentre in TDO basta una riga di codice per richiamare la funzione adibita alla SP (nella classe tdohelperclass) passando come parametro (della funzione) il parametro della SP.

Tutte le operazioni necessarie all'esecuzione della SP sono già state implementate da TDOCodeGenerator nella funzione della classe tdohelperclass.

Un solo linguaggio: Gli statement T-SQL si possono scrivere in C# (o VB.NET o altro linguaggio), evitando così allo sviluppatore di utilizzare un ulteriore linguaggio durante la stesura del codice:

```
WHERE (Firstname = 'Nancy' AND LastName LIKE '%davolio%')
```

con TDO si scrive

```
Clause.Where
(tdo.tEmployees.Firstname=="Nancy" & tdo.tEmployees.
                                    Lastname%"%davolio%")
```

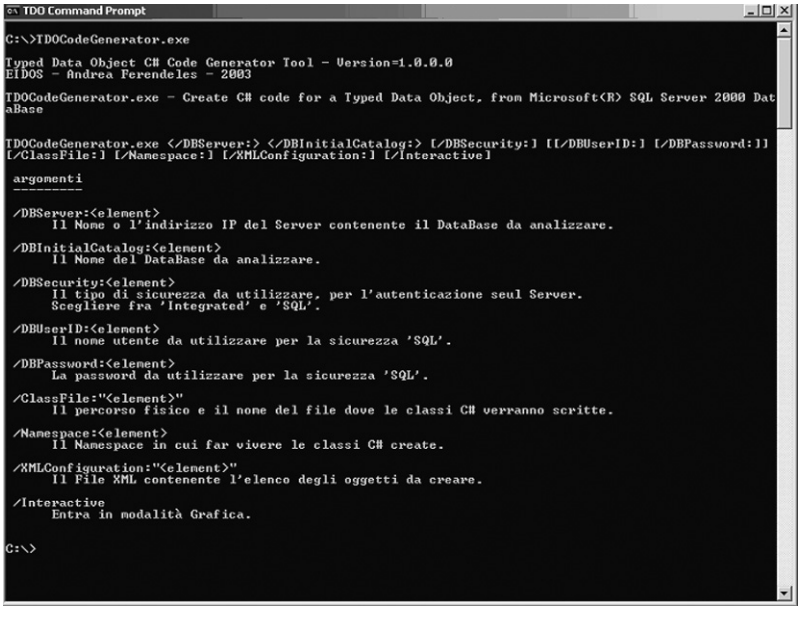


Figura 2 I parametri che è possibile passare a TDOCodeGenerator.exe

Figura 3 TDOCodeGenerator.exe in azione sul Database Northwind

Documentazione automatica del codice: tutta la documentazione "tecnica" delle classi prodotte da TDOCodeGenerator è automaticamente scritta attraverso XML Documentation, per ogni classe, property, function, ecc… Questa caratteristica unitamente alle funzionalità di VS.NET, consente allo sviluppatore di produrre automaticamente tutta la documentazione tecnica necessaria (in formato HTML) per il Data Object.

Basato su ADO.NET connesso e/o disconnesso: TDO può essere utilizzato sia in modalità connessa che in modalità disconnessa, proprio perché utilizza e si integra appieno con la parte connessa (Connection, Command, ecc…) e la parte disconnessa (DataSet) di ADO.NET.

Performance: le performance di TDO sono del 45% migliori rispetto alla parte disconnessa di ADO.NET (DataAdapter); del 5% rispetto a statement T-SQL non parametrici della parte connessa (Command).

Tali performance sono ottenute adottando alcuni accorgimenti noti per l'ottimizzazione delle query su MS SQL Server 2000 (es. caching execution plan).

Scalabilità: TDO risulta molto scalabile in quanto tutti gli statement lanciati dallo sviluppatore attraverso TDO vengono resi parametrici prima di essere inviati a SQL Server 2000, sfruttando dunque la scalabilità delle query parametriche di SQL Server 2000. Inoltre TDO offre una caratteristica di occupazione della memoria RAM "al primo utilizzo", consentendo così l'uso selettivo delle classi TDO.

Adattamento ai cambiamenti: a fronte di cambiamenti nella struttura della base dati (cambiamenti della base dati in corso d'opera/ manutenzione evolutiva) è sufficiente rieseguire TDOCodeGenerator.exe sul database ed in pochi secondi tutte le classi sono rigenerate. Ad esempio, l'esecuzione di TDOCodeGenerator sul database Northwind ha generato 16.590 righe di codice C#, per un totale di 720 Kb di codice sorgente, 192 Kb di codice compilato MSIL e 25 Kb di spazio occupato in RAM; il tutto in soli 8 secondi.

Bindable: TDO può essere utilizzato per operazioni di data binding (simple e complex); inoltre implementando l'interfaccia IComponent può essere utilizzato come componente .NET.

Serializable: tutte le classi TDO sono marcate con l'attributo [Serializable] e sono dunque serializzabili sia in formato binario che in formato XML; in virtù di questa feature TDO non è stato costruito con il namespace System.Data.SqlTypes, che contiene tutte classi non Serializzabili.

COM Interoperability: TDO può essere utilizzato da applicazioni basate su COM (ad esempio applicazioni scritte in Microsoft Visual Basic 6.0).

Esempi di utilizzo

Creare una istanza della classe TDOHelperClass di TDO:

- Creare un nuovo Progetto in C#.NET (qualsiasi template)
- Aggiungere un riferimento a System.Data.TDO.dll
- Aggiungere il file di classi generato da TDO per il Vs. Database MS SQL Server 7.0/2000

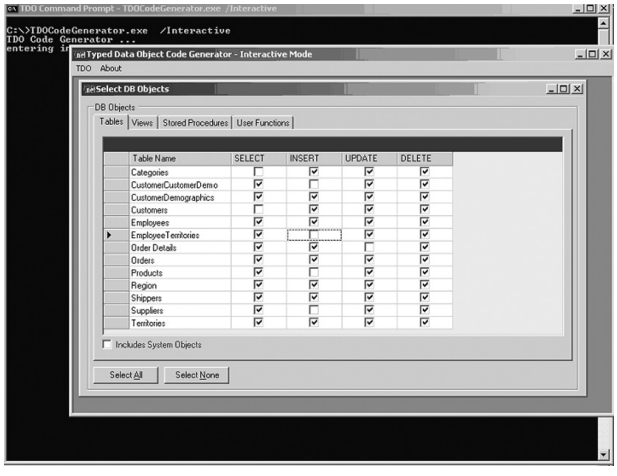


Figura 2 TDOCodeGenerator.exe /Interactive – La gestione diventa più semplice tramite una finestra Windows. In modalità Interactive è addirittura possibile scegliere per quali operazioni TDO dovrà generare codice per ogni db object

- Aggiungere le clausole using System.Data.TDO e using TDO.MyDatabase
- Creare un Istanza della class TDOHelperClass

```
using System.Data.TDO;
using TDO.MyDatabase;
...
private void button1_Click(object sender, System.
                                        EventArgs e)
{
string connectionstring="Data Source=(local);Initial
        Catalog=MyDatabase;userid=sa;password=password;";
TDOHelperClass tdo=new TDOHelperClass(connectionstring);
tdo.OpenConnection();
//do something with TDO ...
tdo.CloseConnection();
}
```

Effettuare una select da una Tabella del Database:

- Aprire la connessione al Database (opzionale)
- Utilizzare il metodo SelectDataTable più appropriato. Tutte le Tabelle hanno come prefisso 't' (se la connessione è chiusa il metodo SelectDataTable la apre automaticamente)
- Chiudere la connessione

```
DataTable dt;
tdo.OpenConnection();
//select * from Employees
dt = tdo.tEmployees.SelectDataTable();
tdo.CloseConnection();
```

Oppure …

```
DataTable dt;
tdo.OpenConnection();
//select * from Employees
// where
// Lastname='Davolio'
// AND
// FirstName='Nancy'
dt=tdo.tEmployees.SelectDataTable
(
Clause.Where
(
tdo.tEmployees.Lastname=="Davolio"
&&
tdo.tEmployees.Firstname=="Nancy"
)
);
tdo.CloseConnection();
```

Oppure …

```
DataTable dt;
tdo.OpenConnection();
//select EmployeeID, BirthDate
// from Employees
// where
// Lastname='Davolio'
// AND
// FirstName='Nancy'
dt=tdo.tEmployees.SelectDataTable
(
Clause.Where
(
tdo.tEmployees.Lastname=="Davolio"
&&
tdo.tEmployees.Firstname=="Nancy"
),
tdo.tEmployees.Employeeid,
tdo.tEmployees.Birthdate,
);
tdo.CloseConnection();
```

Oppure …

```
DataTable dt;
tdo.OpenConnection();
//select EmployeeID, BirthDate from Employees
// where
// Lastname LIKE '%avol%'
// order by
```

```
// BirthDate ASC
dt=tdo.tEmployees.SelectDataTable
(
Clause.Where(tdo.tEmployees.Lastname%"%avol%")
+
Clause.OrderBy(tdo.tEmployees.Birthdate,Clause.Asc)
,
tdo.tEmployees.Employeeid,
tdo.tEmployees.Birthdate
);
tdo.CloseConnection();
```

Inserire un Record in una Tabella del Database:

- Aprire la connessione al database
- Utilizzare il metodo BeginEdit() della Tabella sulla quale si vuole inserire il Record
- Assegnare i valori alle proprietà della Tabella (.Value)
- Invocare il metodo EndEdit() della Tabella
- Invocare il metodo Insert()
- Chiudere la connessione al database

```
//INSERT INTO Employees (FirstName, LastName) VALUES
                                        ('John', 'Doe')
tdo.OpenConnection();

tdo.tEmployees.BeginEdit();
tdo.tEmployees.Firstname.Value="John";
tdo.tEmployees.Lastname.Value="Doe";
tdo.tEmployees.EndEdit();
tdo.tEmployees.Insert();
//Se si devono inserire più Record sequenzialmente
//utilizzare Insert(true ) solo al primo inserimento e
                                        successivamente
//Insert(false ). In questo modo la query di Insert viene
                                "Preparata" al primo utilizzo
//e riutilizzata nei successivi inserimento, aumentando
                                        così le performance.
tdo.CloseConnection();
```

In alternativa è possibile utilizzare direttamente il metodo Insert(....), passando direttamente i valori dei campi

```
//INSERT INTO Employees (FirstName, LastName) VALUES
                                        ('John', 'Doe')
tdo.OpenConnection();
DBNull n=DBNull.Value;
tdo.tEmployees.Insert("John","Doe",n,n,n,n,n,n,n,n,n,n,n,
                                        n,n,n,n,true );
//Se si devono inserire più Record sequenzialmente
//utilizzare Insert(...,true ) solo al primo inserimento
                                        e successivamente
//Insert(...,false ). In questo modo la query di Insert
                                viene "Preparata" al primo utilizzo
//e riutilizzata nei successivi inserimento, aumentando
                                        così le performance.
tdo.CloseConnection();
```

Modificare un Record in una Tabella del Database:

- Aprire la connessione al database
- Utilizzare il metodo BeginEdit() della Tabella sulla quale si vuole modificare il Record
- Assegnare i valori alle proprietà della Tabella (.Value)
- Invocare il metodo EndEdit() della Tabella
- Invocare il metodo Update()
- Chiudere la connessione al database

```
//UPDATE Employees SET FirstName="John", LastName="Doe"
// WHERE EmployeeID=1
tdo.OpenConnection();
tdo.tEmployees.BeginEdit();
tdo.tEmployees.Firstname.Value="John";
tdo.tEmployees.Lastname.Value="Doe";
tdo.tEmployees.EndEdit();
tdo.tEmployees.Update(Clause.Where(tdo.tEmployees.
                                        EmployeeID==1));
tdo.CloseConnection();
```

Eliminare un Record da una Tabella del Database:

- Aprire la connessione al Database
- Utilizzare il metodo Delete() della Tabella dalla quale si vuole eliminare il/i Record(s)
- Chiudere la connessione al Database

```
//DELETE From Employees WHERE Photo ISNULL
tdo.OpenConnection();
tdo.tEmployees.Delete(Clause.Where(tdo.tEmployees.Photo==
                                        DBNull.Value));
tdo.CloseConnection();
```

Eseguire una Stored Procedure:

- Aprire la connessione al database
- Tutte le Stored Procedure hanno come prefisso 'p'. Utilizzare il metodo pNomeStoredProcedure passando direttamente i parametri tipizzati
- Il metodo pNomeStoredProcedure restituisce sempre un oggetto System.Data.DataTable
- Chiudere la connessione al database

```
//EXEC Employee Sales by Country('01-01-1999', '12-31-
                                        1999')
```

```
tdo.OpenConnection();
DataTable myresults=tdo.tEmployees.pEmployeesalesbycountry(
      new DateTime(1999,01,01), new DateTime(1999,12,31));
tdo.CloseConnection();
```

Effettuare una Select da una Vista del database:

- Aprire la connessione al database (opzionale)
- Utilizzare il metodo SelectDataTable più appropriato. Tutte le Viste hanno come prefisso 'v' (se la connessione è chiusa il metodo SelectDataTable la apre automaticamente)
- Chiudere la connessione al database
- Sulle viste non sono presenti i metodi di Insert(), Update() e Delete().

```
//Select * FROM [Alphabetical List of Products]
tdo.OpenConnection(); //Opzionale
DataTable myproducts=tdo.vAlphabeticallistofproducts.
                                    SelectDataTable();
tdo.CloseConnection(); //Opzionale
```

Eseguire una User/Table Function:

- Aprire la connessione al database
- Utilizzare il metodo fNomeFunzione più appropriato. Tutte le Functions hanno come prefisso 'f'.
- Chiudere la connessione al database
- Le User Functions restiuiscono sempre il loro tipo dichiarato, Le Table Functions un oggetto System.Data.DataTable

```
// create this function on Northwind DB
// ******************************
// CREATE FUNCTION dbo.MySum(@a INT, @b INT)
// RETURNS INT as
// BEGIN
// RETURN (@a, @b)
// END
// ******************************
tdo.OpenConnection();
int results=tdo.fMysum(3,5); // results=8
tdo.CloseConnection();
```

Effettuare una Select con la clausola WHERE ... IN:

- Aprire la connessione al Database
- Utilizzare il metodo SelectXXXXX più appropriato.
- Nella Clause.Where utilizzare la classe TDOWhereSetOfValues.

```
//Select * FROM Employees WHERE EmployeeID IN (1,3,5)
tdo.OpenConnection();
DataTable myEmployees=tdo.tEmployees.SelectDataTable(
Clause.Where(tdo.tEmployees.Employeeid==new TDOWhereSetO
                                    fValues(1,3,5)));
tdo.CloseConnection();
```

Effettuare una Select con la clausola WHERE ... BETWEEN:

- Aprire la connessione al Database
- Utilizzare il metodo SelectXXXXX più appropriato.
- Nella Clause.Where utilizzare la classe TDOWhereRange.

```
//Select * FROM Employees WHERE
     EmployeeID BETWEEN 2 AND 6
tdo.OpenConnection();
DataTable myEmployees=tdo.
   tEmployees.SelectDataTable(
Clause.Where(tdo.tEmployees.
  Employeeid==new TDOWhereRange
                          (2,6)));
tdo.CloseConnection();
```

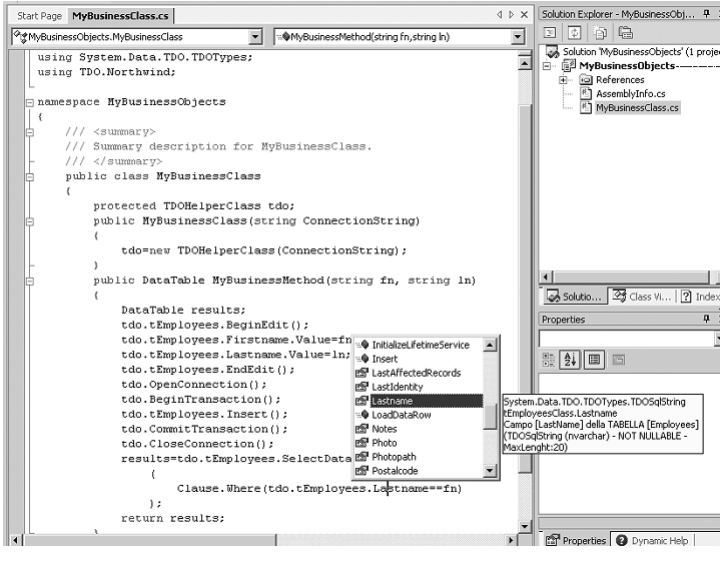


Figura 5 Ecco come la classe TDOHelperClass per Northwind si presenta in Visual Studio .NET

Effettuare operazioni di Aggregazione Select COUNT, GROUP BY, ecc.:

La filosofia di TDO è molto semplice: qualsiasi operazione di Aggregazione deve essere intermediata da Viste, Stored Procedure e User/Table Function e non può essere effettuata direttamente dalla classe TDOHelperClass.

Nelle classe TDOHelperClass, vengono comunque forniti 4 metodi accessori (Fill, ExecuteReader, ExecuteNonQuery, ExecuteScalar) per eseguire statement T-SQL "liberi". Questi metodi sono marcati come [Obsolete] in modo che il compilatore generi un Warning a compile-time. Ciò ne consente una rapida individuazione in caso di necessità. Se si dovesse rendere necessario l'uso di questi metodi è comunque possibile adottare alcuni accorgimenti, per impattare minimamente sulla manutenibilità del codice. Tale scelta progettuale, spinge l'utente a fare uso di strumenti più consoni e performanti ad un DB relazionale. Attraverso lo strumento TDOCodeGenerator (esposto attraverso TDO Service on-line), l'utilizzo di Viste, SP e UF create nel database è immediato.

È sufficiente infatti creare tali Oggetti sul database e rilanciare il generatore di codice. In pochi secondi la classe TDOHelperClass prodotta riespone tutti i nuovi oggetti.

```
//Metodi [Obsolete]
tdo.OpenConnection();
DataTable myDataTable=tdo.Fill("select count("+
      tdo.tEmployees.FirstName.ColumnName +") TOT From " +
 tdo.tEmployees.TableName + " Group By " + tdo.tEmployees.
                                  FirstName.ColumnName);
tdo.CloseConnection();
```

Modificare il codice TDO per implementare alcune funzionalità "Custom":

Anche in questo caso la filosofia di TDO è molto semplice. Ragioniamo in OOP: se avete necessità di aggiungere nuove feature alle vostre classi TDO, estendete le classi esistenti, tramite meccanismi di ereditarietà. Il codice sorgente TDO non dovrebbe essere mai modificato, in quanto una rigenerazione (a fronte di modifiche sul database), farebbe perdere tali modifiche. Se invece avete esteso le classi TDO e concentrato qui l'aggiunta di nuove classi/metodi, tali modifiche verranno ovviamente preservate. Qui di seguito è riportato un esempio di estensione sia della classe TDOHelperClass che della classe tEmployeesClass (che espone metodi ed attributi della Tabella Employees):

```
using System;
using System.Data.SqlClient;
using System.Data.TDO;
using System.Data.TDO.TDOTypes;
using TDO.Northwind;
using TDO.Northwind.Entities;

namespace TDONorthwindExtended
{
namespace Entities
{
public class tEmployeesExtendedClass : tEmployeesClass,
                                                  ITDOTable
{
public tEmployeesExtendedClass(TDOHelperClass tdohelper-
                                    class) : base(tdohelperclass)
{ }
public string MyNewEmployeesMethod()
{
//Do something
//...
return "MyNewEmployeesMethod of tEmployeesExtendedClass";
}
public new void Insert(bool preparethiscommand)
{
base.Insert(preparethiscommand);
//Do something else
//...
}
}
}
public class TDONortwindExtendedClass : TDOHelperClass,
                                                  ITDOHelper
{
public new Entities.tEmployeesExtendedClass tEmployees;
public TDONortwindExtendedClass(string connectionstring)
                                    : base(connectionstring)
{
this.tEmployees=new TDONorthwindExtended.Entities.
                                tEmployeesExtendedClass(this);
}
public string MyNewMethod()
{
// Do something
return "MyNewMethod of TDONorthwindExtendedClass";
}
}
}
```

Riferimenti

[1] TDO Evaluation copy – www.eidosis.com/tdo

SOA e il .NET Framework

Aspetti pratici della Service Oriented Architecture per gli sviluppatori .NET

di Paolo Pialorsi

Seconda puntata

Nel precedente articolo a proposito della Service Oriented Architecture abbiamo visto quali sono gli aspetti chiave e gli obiettivi di questa nuova architettura del software. Affrontiamo ora alcuni aspetti pratici dello sviluppo SOA, rivolgendo la nostra attenzione agli strumenti e alle tecniche da adottare per lavorare con il .NET Framework e con Visual Studio .NET 2003.

Strumenti per disegnare i messaggi

Come abbiamo già visto, una delle fasi più importanti di un progetto SOA è il disegno dei messaggi, ad oggi sotto forma di documenti XSD. Benché la sintassi di XML Schema non sia eccessivamente complessa, consentendoci di prendere in considerazione un qualsiasi editor testuale (persino il Blocco Note di Windows), conviene comunque appoggiarsi a qualche software che ci aiuti nella definizione dei messaggi. Sicuramente uno strumento comodo e che tutti noi dovremmo avere a disposizione è Visual Studio .NET. Con esso possiamo infatti creare dei documenti XSD sia utilizzando un designer grafico (**Figura 1**), che sfruttando un comodo editor testuale, dotato di intellisense sulla grammatica di XSD (**Figura 2**). Personalmente utilizzo nella maggioranza dei casi l'editor testuale, perchè ritengo che avendo un minimo di padronanza del linguaggio XSD si riesca ad essere molto più rapidi, quindi anche più produttivi, che non lavorando con il mouse e il designer grafico. Quest'ultimo lo utilizzo generalmente quando devo definire lo schema di una struttura dati che estraggo da un database. Visual Studio .NET consente infatti di trascinare (drag'n'drop), all'interno dell'area di disegno di un XSD, le tabelle di una base dati, convertendo in schema la loro struttura.

Esistono poi sul mercato decine di programmi in grado di fornire editor evoluti di XSD. Il più famoso di questi è certamente XMLSpy della Altova [2]. A prescindere dal tool che utilizziamo per definire lo schema XSD, avremo l'esigenza di trattarlo da codice. Se stiamo lavorando con il Framework .NET possiamo utilizzare un ulteriore tool, questa volta a riga di comando, di nome XSD.EXE, che è in grado di leggere un XSD e crearne una rappresentazione equivalente sotto forma di linguaggio .NET (C#, VB.NET, J#).

Pensiamo quindi a questo semplice schema:

XSD

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

<xsd:element name="product">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="description"
                    type="xsd:string" />
```

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.*

```
        <xsd:any minOccurs="0" maxOccurs="unbounded"
                                    processContents="lax" />
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:int"
                                    use="required" />
    <xsd:attribute name="euroPrice" type="xsd:decimal"
                                    use="required" />
    <xsd:anyAttribute processContents="lax" />
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Ipotizzando che il file XSD si chiami product.xsd, possiamo invocare XSD.EXE nel modo seguente:

Command Line

```
XSD.EXE /c /n:DevLeap.Entities product.xsd
```

Dove il parametro /c indica che vogliamo generare una o più classi, corrispondenti alle entità descritte nello schema. Il parametro /n comunica invece quale deve essere il namespace del codice autogenerato. Il terzo parametro è ovviamente il file XSD sorgente.

Il risultato sarà simile al seguente:

C#

```
using System.Xml.Serialization;
namespace DevLeap.Entities {
[XmlTypeAttribute(Namespace="http://schemas.devleap.com/
                                                Product")]
[XmlRootAttribute(Namespace="http://schemas.devleap.com/
                                Product", IsNullable=false)]
public class product {

 public string description;

 [XmlAnyElementAttribute()]
 public System.Xml.XmlElement[] Any;

 [XmlAttributeAttribute()]
 public int id;

 [XmlAttributeAttribute()]
 public System.Decimal euroPrice;

 [XmlAnyAttributeAttribute()]
 public System.Xml.XmlAttribute[] AnyAttr;
}
}
```

Siamo a questo punto pronti per includere il file di codice nei nostri progetti e utilizzarlo. Ciò che per noi sarà un'istanza della classe product, per chi si interfaccerà con noi e per i nostri stessi servizi sarà un messaggio strutturato come definito in product.xsd. Chi ci garantisce tutto questo è la presenza nel codice di una serie di attributi, relativi al namespace System.Xml.Serialization che sono finalizzati proprio a dire a XmlSerializer, il motore di serializzazione dei messaggi, che tipo di struttura XML deve generare.

Per esempio XmlAttributeAttribute dice al serializzatore di generare un attributo XML al cui interno inserire il valore della proprietà id.

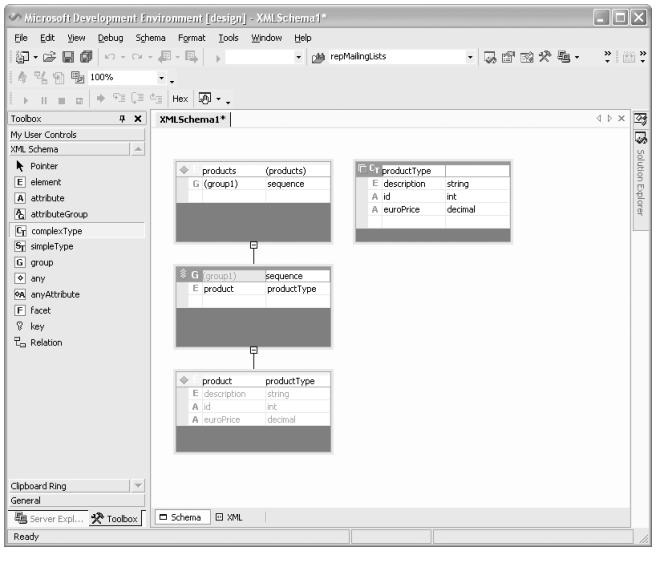


Figura 1 Designer di uno schema XSD in Visual Studio .NET

L'approccio "Contract First"

Ipotizziamo di aver definito tutti i messaggi da utilizzare nei nostri servizi. Ora dobbiamo definire i contratti di quei servizi, proprio secondo un approccio "Contract First", cioè definendo prima i contratti e poi i servizi che li implementano e non il contrario. I contratti abbiamo già visto essere dei file WSDL, cioè dei file XML che descrivono: servizi, binding, porte, messaggi, tipi. Vediamoli e definiamoli uno per uno. La sezione relativa ai servizi descrive l'esistenza di uno o più servizi, la URL alla quale sono raggiungibili e il binding per utilizzarli:

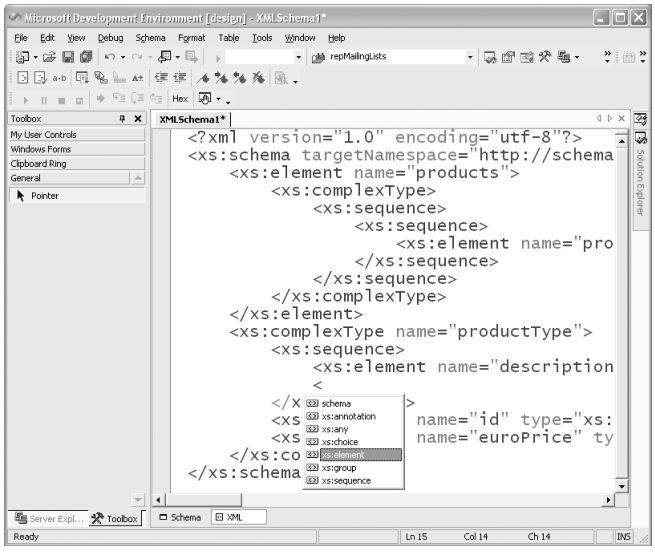


Figura 2 Editor di uno schema XSD in Visual Studio .NET

WSDL

```
<service name="productService">
  <port name="productService" binding="svc:productService
                                                Binding">
    <soap:address location="http://localhost/
                                    ProductService.asmx" />
  </port>
</service>
```

Generalmente è sufficiente scegliere un nome per il servizio, per la relativa porta e per il binding al protocollo, quindi indicare una URL. La sezione relativi ai servizi non è obbligatoria in un approccio "Contract First", in quanto potremmo decidere a posteriori come e dove implementare il contratto.

Dobbiamo invece definire il binding al protocollo:

WSDL

```
<binding name="productServiceBinding" type="svc:product
                                              ServiceSoap">
  <soap:binding style="document" transport="http://
                                schemas.xmlsoap.org/soap/http" />
  <operation name="GetProduct">
    <soap:operation soapAction="urn:GetProduct" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

In questa sezione è importante notare che, per essere compatibili con le regole di WS-I [5], dobbiamo descrivere i messaggi con formato document/literal [2].

Serve inoltre dichiarare le singole operazioni che vogliamo esporre con il servizio.

Ogni operazione potrà essere costituita da un messaggio in ingresso (input) e un messaggio in uscita (output).

I messaggi passeranno attraverso una porta:

WSDL

```
<portType name="productServiceSoap">
  <operation name="GetProduct">
    <input message="svc:GetProductSoapIn" />
    <output message="svc:GetProductSoapOut" />
  </operation>
</portType>
```

La porta dichiarerà il proprio nome e, per ciascuna operazione, la corrispondenza tra i messaggi e l'operazione stessa. Per definire i messaggi dovremo utilizzare l'apposita sezione del WSDL:

WSDL

```
<message name="GetProductSoapIn">
  <part name="parameter" element="svc:getProduct" />
</message>
<message name="GetProductSoapOut">
  <part name="parameter" element="p:product" />
</message>
```

Si noti che ogni messaggio può essere fatto di N parti. Dal punto di vista della specifica WS-I Basic Profile 1.1, che detta le regole fondamentali per la interoperabilità dei servizi SOAP, un messaggio può però essere costituito al più da una parte, quindi o zero o una parte.

Le singole parti poi referenziano la sezione dei tipi di WSDL, per descrivere la loro struttura:

WSDL

```
<types>
  <xsd:schema elementFormDefault="qualified"
    targetNamespace="http://schemas.devleap.com/
                                    ProductService">
```

```
    <xsd:import namespace="http://schemas.devleap.com/
                                                Product"
    schemaLocation="http://localhost/Product.xsd" />
    <xsd:element name="getProduct" type="xsd:string" />
  </xsd:schema>
</types>
```

In questo caso definiamo come è strutturato il messaggio getProduct, mentre importiamo da un file esterno (il nostro product.xsd) la definizione dello schema del prodotto.

Si noti che l'importazione dello schema esterno, per essere ancora una volta compatibili con WS-I BP1.1, deve avvenire all'interno della sezione dei tipi, cioè nello schema, e non può essere eseguita a livello di intero contratto WSDL.

La specifica di WSDL 1.1 prevede infatti un tag wsdl:import che però dovrebbe essere utilizzato solo per importare altri file WSDL.

Purtroppo anche il .NET Framework, quando definite dei Web Service che restituiscono dei typed DataSet [3] inserisce nel WSDL autogenerato (quello che ottenete chiamando nel browser servizio.asmx?wsdl) un tag wsdl:import che importa lo schema del typed DataSet a livello di WSDL, anzichè a livello di schema XSD, rendendo il risultato non compatibile con WS-I BP1.1. In ogni caso noi stiamo costruendo manualmente il nostro documento WSDL proprio per essere sicuri che il risultato sia corretto e compatibile con tutte le specifiche oggi esistenti. Noi ragioniamo con l'approccio "Contract First"!

L'attività di disegno del WSDL può essere svolta utilizzando Notepad, Visual Studio .NET o altri tool di terze parti. Per esempio anche in questo caso XmlSpy fornisce un designer grafico per la definizione di WSDL. Se per il disegno di un XSD trovo superfluo appoggiarsi a designer esterni, per la definizione di un WSDL può invece risultare molto più comodo sfruttare strumenti come XmlSpy. Consiglio altrimenti di conservare un file WSDL di base, semplice ma completo, da utilizzare sempre come punto di partenza per qualsiasi WSDL che si debba definire.

Implementare i servizi server-side

Ora che abbiamo definito il contratto sotto forma di WSDL possiamo innanzitutto "sottoscriverlo" con i nostri partner, cioè verificare che risponda ai requisiti, per poi fornirlo a chi si deve interfacciare con i servizi che descrive.

Sia noi, che sviluppiamo il servizio, che i nostri clienti/partner, cioè i "consumer" del servizio, possiamo basarci sul WSDL, che prescinde dalla piattaforma di sviluppo, dedicandoci allo sviluppo reale del servizio e/o del client, detto anche proxy, con un qualsiasi ambiente di sviluppo.

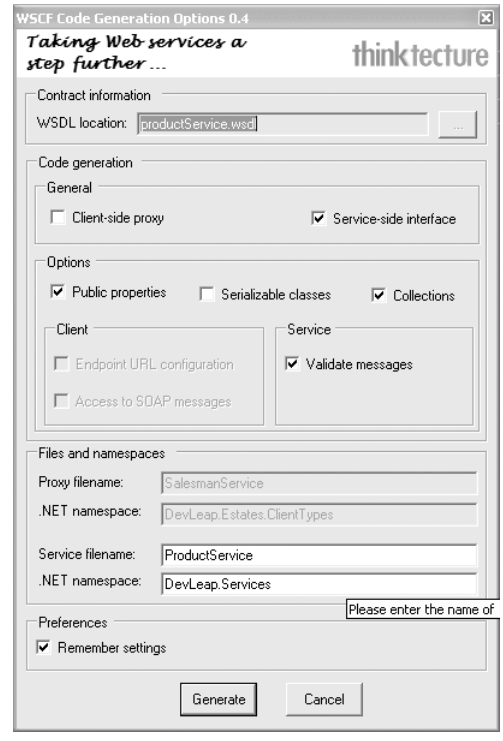


Figura 3 WSContractFirst

Nei più diffusi ambienti di sviluppo (.NET e Java in primis) esistono strumenti per la generazione automatica del codice lato client e lato server, rispetto alla struttura di un documento WSDL. Con il .NET Framework questo si ottiene utilizzando il programma WSDL.EXE nel modo seguente:

Command Line

```
WSDL.EXE /server /n:DevLeap.Services productService.WSDL
```

Dove /server indica che vogliamo generare il codice lato server, il parametro /n al solito indica il namespace .NET del codice da generare e l'ultimo parametro sarà la URL del nostro file WSDL, quello che abbiamo creato manualmente.

Il codice generato conterrà la definizione di una classe astratta che rappresenta le caratteristiche del WSDL sotto forma di Web Service ASP.NET. Nel nostro caso avremo, in forma abbreviata:

C#

```
// Definizione della classe astratta del servizio
 [WebServiceBindingAttribute(Name="productServiceBinding",
   Namespace="http://schemas.devleap.com/ProductService")]
public abstract class productService : System.Web.Services.
                                                WebService {

// Definizione della operazione GetProduct come WebMethod
                    [System.Web.Services.WebMethodAttribute()]
[System.Web.Services.Protocols.SoapDocumentMethodAttribute
    ("urn:GetProduct", Use=System.Web.Services.Description.
        SoapBindingUse.Literal, ParameterStyle=System.Web.
                Services.Protocols.SoapParameterStyle.Bare)]
[return: System.Xml.Serialization.XmlElementAttribute
("product", Namespace="http://schemas.devleap.com/Product")]
public abstract product GetProduct(
[System.Xml.Serialization.XmlElementAttribute(Namespace=
      "http://schemas.devleap.com/ProductService")] string
                                                getProduct);
}
```

Si può subito notare che la classe generata è molto ricca di meta-informazioni, sotto forma di attributi .NET. Si tratta di attributi che comunicano al motore dei Web Service di ASP.NET come saranno strutturati i messaggi SOAP in ingresso e in uscita. Per utilizzare questa classe dovremo creare una nostra classe, da essa derivata, copiare tutti gli attributi .NET presenti sulla classe base e sui suoi metodi, perchè gli attributi .NET non vengono derivati dalle classi figlie e quindi implementare il codice dei singoli metodi.

Nel nostro caso dovremo implementare solo il metodo GetProduct, che restituisce un oggetto di tipo product, definito secondo lo schema XSD da noi costruito. Negli esempi associati all'articolo trovate un esempio completo di tutto questo procedimento.

Esiste un add-in per Visual Studio .NET di nome WSContractFirst (**Figura 3**), creato da Christian Weyer [8], che consente di svolgere queste operazioni senza abbandonare l'ambiente di sviluppo di Visual Studio .NET.

Inoltre sempre questo tool genera già la classe derivata, pronta da implementare e consente di aggiungere alcune caratteristiche interessanti al codice autogenerato. Per esempio si può chiedere a WSContractFirst di convertire eventuali array di oggetti in collezioni, per poterli meglio gestire da codice. Inoltre consente di attivare lato server una SOAP Extension [9] che esegue la validazione XSD dei messaggi, per verificare che in input ai nostri servizi arrivino solo documenti validi, rispetto agli schema creati.

Verificare la bontà del risultato

Leggendo queste due puntate sullo sviluppo SOA dovrebbe essere emerso in modo chiaro il fatto che sviluppare in architettura SOA oggi non è banale e non è a costo zero.

Dovrebbe però anche essere chiaro che sviluppare architetture SOA, quando effettivamente è necessario essere interoperabili, indipendenti dalla piattaforma e disaccoppiati, è la scelta corretta. Sarebbe però paradossale, oltre che svilente, dedicare tempo e risorse allo sviluppo di un'architettura SOA, per poi scoprire che non siamo realmente interoperabili con le altre piattaforme, nel momento in cui dovesse servirci realmente di esserlo. La garanzia di interoperabilità vera oggi non c'è. Abbiamo però una garanzia di interoperabilità ad oggi sicuramente teorica e andando avanti nel tempo anche pratica e per tutte le piattaforme: WS-I Basic Profile 1.1.

Per avere la certezza del fatto che i nostri servizi sono compatibili con WS-I BP1.1 possiamo utilizzare degli strumenti che il consorzio WS-Interoperability stesso ci mette a disposizione come download dal sito di WS-I [5].

Mi riferisco ai WS-I Test Tools che prevedono un Monitor e un Analyzer che rispettivamente tracciano e analizzano il traffico SOAP, verificando se i messaggi, gli schema e i contratti corrispondono alle regole di BP1.1 o meno.

Se i nostri servizi non dovessero essere compatibili con BP1.1 l'Analyzer ci dirà non solo che il test è fallito, ma ci segnalerà anche puntualmente i "difetti" del servizio, in modo da poterli correggere tempestivamente.

Il mio consiglio è di utilizzare sempre questi tool per verificare la correttezza delle implementazioni SOA.

Bibliografia e Riferimenti

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

[2] P. Pialorsi – "XML & Web Services .NET Full Contact", Mondadori Informatica, 2003

[3] P. Pialorsi – "Accesso ai dati tramite Web Service", Visual Basic Journal n° 55

[4] http://www.altova.com/

[5] http://www.ws-i.org/

[6] http://msdn.microsoft.com/architecture/soa/

[7] http://www.w3.org/TR/xml-infoset/

[8] http://www.thinktecture.com/Resources/Software/WSContractFirst/

[9] http://www.devleap.com/SchedaArticolo.aspx?IdArticolo=10450

I design pattern più famosi
implementati in VB .NET
Prima puntata

In questa breve serie di articoli vedremo alcuni esempi di implementazione dei design pattern più noti

di Lorenzo Vandoni

Il movimento dei pattern, nato nel 1993 a seguito della pubblicazione del libro "Design Pattern" di Gamma, Helm, Johnson e Vlissides (soprannominati *gang of four*, o *GoF*), costituisce una delle più significative novità dell'ultimo decennio nel campo dell'ingegneria del software.

Un pattern è una soluzione generalizzata per un determinato problema, valida nell'ambito di un contesto.

Per capire di cosa si tratta, basta fare un parallelo con una ricetta di cucina. Il problema è il piatto da preparare, la soluzione è la ricetta, e il contesto sono gli ingredienti disponibili.

Un pattern, di per sé, è quindi semplicemente un approccio alla risoluzione di problemi, del tutto interdisciplinare.

Quando si parla di "design pattern", invece, si intende l'applicazione di questo approccio alla progettazione del software.

In questa serie di articoli verranno mostrati degli esempi di implementazione di alcuni pattern, utilizzando il linguaggio Visual Basic.NET. Parlare di "implementazione" di un design pattern è di per sé scorretto, perché la soluzione di un pattern di questo tipo è di tipo progettuale, e può essere implementata in molti modi diversi.

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.*

È più corretto quindi parlare di esempi, anche se nessuno vi vieterà di utilizzare il codice proposto come base per un'analoga implementazione all'interno di una vostra applicazione.

Descrizione di un design pattern

La descrizione di un pattern è principalmente testuale, ma deve essere strutturata in modo da evidenziare alcune proprietà che hanno lo scopo di descrivere separatamente il problema, i vincoli e la soluzione, nonché evidenziare i possibili legami di questo con altri pattern.

Queste proprietà non sono esattamente identiche in tutte le pubblicazioni. Utilizzando come riferimento il libro "Design Pattern", possiamo citare, tra le proprietà principali:

- *Intent*, una breve descrizione dell'*obiettivo* del pattern;
- *Motivation*, un'ampia descrizione delle motivazioni che stanno alla base dell'utilizzo del pattern. Questa proprietà è la più importante, perché espone il proble-

ma che il pattern intende risolvere ed eventuali vincoli da prendere in considerazione;

- *Applicability*, alcuni possibili usi concreti del pattern;
- *Structure*, il diagramma UML delle classi coinvolte;
- *Participants*, una descrizione più dettagliata delle responsabilità delle classi coinvolte;
- *Collaborations*, eventuali possibili interazioni con altri pattern;
- *Consequences*, possibili conseguenze pratiche relative alla soluzione adottata;
- *Known uses*, un elenco di usi noti del pattern. Va notato che questi possono essere antecedenti alla pubblicazione dello stesso pattern. In questo caso il pattern costituisce semplicemente la formalizzazione di un procedimento già noto e applicato in vari contesti.
  Non per questo ne viene diminuita l'utilità. Anzi, descrivere in modo preciso una buona pratica di programmazione può essere il modo migliore per incentivarne l'utilizzo anche in altri progetti.

Listato 1 Implementazione del pattern Iterator in VB.NET

```
Namespace Iterator
    '''''''''''''''''''''''''''''
    ''' Interfaccia Iterator '''
    '''''''''''''''''''''''''''''
    Public Interface Iterator
        Function FirstItem() As Boolean
        Function NextItem() As Boolean
        Function CurrentItem() As Object
    End Interface

    '''''''''''''''''''''''''''''
    ''' Interfaccia Collection '''
    '''''''''''''''''''''''''''''
    Public Interface Collection
        Function CreateIterator() As Iterator
        Function AddItem(ByVal oItem As Object)
                                    As Boolean
        Function Count() As Integer
    End Interface

    '''''''''''''''''''''''''''''''
    ''' Classe ConcreteIterator '''
    '''''''''''''''''''''''''''''''
    Public Class ConcreteIterator
        Implements Iterator
        Private miCurrent As Integer = 0
        Private moCollection As ConcreteCollection
        Public Sub New(ByVal oCollection As Collection)
            moCollection = oCollection
        End Sub
        Public Function CurrentItem() As Object
Implements Iterator.CurrentItem
            Return moCollection.moItems(miCurrent)
        End Function
        Public Function FirstItem() As Boolean
Implements Iterator.FirstItem
            miCurrent = 0
            Return IIf(moCollection.Count > 0,
                                    True, False)
        End Function
        Public Function NextItem() As Boolean
Implements Iterator.NextItem
            NextItem = False
            If miCurrent < moCollection.Count
Then
                miCurrent = miCurrent + 1
                NextItem = True
            End If
        End Function
    End Class

    ''''''''''''''''''''''''''''''''''
    ''' Classe ConcreteCollection '''
    ''''''''''''''''''''''''''''''''''
    Public Class ConcreteCollection
        Implements Collection
        Friend moItems(10) As Object
        Private miCount As Integer = 0
        Public Function CreateIterator() As
                                        Iterator
            Implements Collection.CreateIterator
            Return New ConcreteIterator(Me)
        End Function
        Public Function AddItem(ByVal oItem As
                                Object) As Boolean
                Implements Collection.AddItem
            AddItem = False
            If miCount < 10 Then
                moItems(miCount) = oItem
                miCount = miCount + 1
                AddItem = True
            End If
        End Function
        Public Function Count() As Integer
                    Implements Collection.Count
            Return miCount
        End Function
    End Class
End Namespace
```

Al di là di queste proprietà, è piuttosto comune che venga fornita anche una implementazione di esempio, utilizzando principalmente linguaggi object-oriented come C++ o Java.

Uso di un design pattern

Il compito più difficile per un progettista che voglia applicare un determinato pattern ad un proprio progetto, sta nella scelta di quale pattern utilizzare. Questa affermazione può sembrare paradossale, ma è estremamente realistica.

Il problema, anzi, può essere ancora più a monte. La ricerca di un pattern, infatti, implica perlomeno l'avere maturato la sensazione di avere un problema di progettazione.

Spesso il pattern non viene nemmeno ricercato, perché il problema non viene riconosciuto come tale. Nei casi più sfortunati, il problema si manifesterà poi quando lo sviluppo del software sarà già stato intrapreso o addirittura terminato, e la scelta di progettazione sbagliata potrà avere conseguenze anche pesanti in termini di manutenzione o estendibilità del software. Anche ammettendo di avere più o meno individuato il proprio problema, però, la scelta del pattern da applicare non è banale, perché il "modello mentale" che il progettista si è fatto del proprio problema può anche essere molto distante dalla descrizione della *motivation* del pattern. La difficoltà che si incontra è quindi quella di capire che il proprio problema può essere modellato ricorrendo ad un determinato pattern.

Una volta risolto questo problema, il compito di applicare la soluzione indicata sarà, nella maggior parte dei casi, molto più semplice.

Il pattern Iterator

Iterator è uno dei pattern più noti ed utilizzati, tra quelli descritti nel libro della GoF, e credo sia il miglior candidato per un introduzione all'argomento. Il pattern è applicato anche in molte librerie commerciali. Ad esempio, qualsiasi cursore utilizzato per accedere ai dati di un recordset SQL può essere considerato un iteratore. Chi preferisca un paragone più concreto può pensare alla classe *DataReader* di ADO.NET, che implementa un cursore forward-only sui dati sottostanti. Vediamo quindi come sia possibile descrivere questo pattern utilizzando le proprietà precedentemente menzionate.

- *Intent*. L'obiettivo è quello di permettere di accedere agli elementi di una collezione senza far conoscere alle applicazioni client la struttura interna della stessa.
- *Motivation*. L'idea di separare l'implementazione di una collezione, come ad esempio una lista o uno stack, dall'implementazione degli algoritmi necessari per accedere ai suoi elementi, e di includerli all'interno di uno o più oggetti *iterator*, porta con sé diversi vantaggi: anzitutto, come già detto, permette di non esporre la struttura interna della lista; inoltre, apre la possibilità di accedere agli elementi della collezione in vari modi (sequenzialmente, con accesso diretto, in modo ordinato, ecc…), semplicemente definendo per ognuna di queste modalità un diverso tipo di iteratore; infine, consente di limitare il numero di metodi definiti nella collezione, e di sfruttare lo stesso tipo di iteratore, per gestire collezioni di tipo diverso, semplificando il codice delle applicazioni *client*.
- *Applicabilità*. L'applicabilità del pattern è di fatto definita nelle motivazioni. Il pattern può essere usato in tutti i casi in cui si debba definire una classe costituita da una collezione di oggetti più semplici, indipendentemente da come la collezione è strutturata.
- *Structure*. La struttura della soluzione è mostrata in **Figura 1**. Una generica collezione definisce un metodo *CreateIterator* che permette di creare un iteratore compatibile con la sua struttura. L'iteratore fornisce i metodi necessari per accedere agli elementi dell'interfaccia. Le due classi *ConcreteCollection* e *ConcreteIterator* costituiscono implementazioni concrete delle interfacce.
- *Participants*. Più in dettaglio, queste sono le responsabilità delle varie classi ed interfacce:

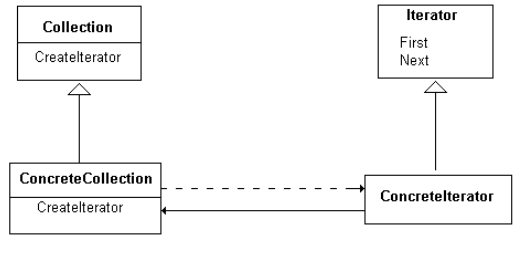


Figura 1 Architettura del pattern Iterator

- l'interfaccia *Iterator* definisce i metodi che dovranno essere implementati da qualsiasi iteratore concreto per permettere alle applicazioni client di spostarsi tra i vari elementi della collezione;
- la classe *ConcreteIterator* implementa l'interfaccia *Iterator* per una specifica collezione;
- l'interfaccia *Collection* definisce i metodi che dovranno essere implementati da una collezione che debba permettere l'utilizzo di iteratori;
- la classe *ConcreteCollection* implementa l'interfaccia *Collection*.

- *Collaborations*. Nessuna.

- *Consequences*. Le conseguenze derivanti dall'utilizzo del pattern sono la possibilità di definire diversi modi per accedere agli elementi della collezione, una semplificazione dell'interfaccia pubblica delle classi *Collection*, che non dovranno dotarsi dei metodi di navigazione, e la possibilità di avere diversi iteratori contemporaneamente attivi sulla stessa collezione, che quindi non sarà limitata ad avere un unico elemento "corrente" o selezionato.

- *Known uses*. Gli iteratori sono comunemente usati in molte librerie e framework ad oggetti, sia specializzati, come quelli per l'accesso a database relazionali, sia generici, come ad esempio la Standard Template Library (STL) di C++.

Implementazione del pattern in VB.NET

Il codice di esempio allegato all'articolo riporta l'implementazione delle due interfacce e delle due classi tratteggiate nel paragrafo precedente. Il numero di metodi definiti nelle interfacce è stato volutamente contenuto in modo da evidenziare meglio le caratteristiche progettuali del pattern.

L'interfaccia *Iterator* contiene due metodi per spostare la posizione del record corrente, e un metodo *CurrentItem* che permette di accedere all'elemento corrente secondo la posizione dell'iteratore. L'interfaccia *Collection*, da parte sua, contiene i metodi necessari per aggiungere nuovi elementi e per contarne il numero.

Per definire il tipo degli elementi mantenuti all'interno della collezione sarebbe corretto utilizzare la programmazione generica, ma questo sarà possibile solo con la prossima versione di .NET.

Per questo motivo, ci si accontenterà di porre il tipo degli elementi della collezione uguale a *Object*. In questo modo, diventa possibile utilizzare la collezione per gestire elementi di qualsiasi tipo, anche se non ci sarà nessun controllo sul fatto che elementi di tipo diverso vengano inseriti nella stessa collezione. Le due interfacce e le due classi, in questo esempio, sono state per comodità implementate all'interno dello stesso file di codice.

Più probabilmente, un'implementazione reale vedrebbe separate le due interfacce dalle due classi.

Le due classi dovranno però essere in qualche modo collegate, perché è ovvio che il *ConcreteIterator* dovrà avere un accesso privilegiato alla *ConcreteCollection*, in modo da poterne manipolare gli elementi senza che questa sia necessariamente costretta a definire dei metodi pubblici per farlo (il che, tra l'altro, renderebbe quasi inutile l'iteratore).

Questa modalità di accesso privilegiato tra classi viene resa disponibile con diverse modalità da quasi tutti i linguaggi di programmazione orientati ad oggetti.

Con Visual Basic.NET si può utilizzare la parola chiave *Friend*, che indica che un particolare metodo o proprietà di una classe deve risultare accessibile da tutte le altre classi dello stesso progetto, ma invisibile all'esterno dello stesso, esattamente come se fosse un membro *Private*.

L'iteratore utilizza questo privilegio per accedere alla struttura della collezione, implementata in questo esempio tramite un semplice array, e tenere traccia della posizione corrente.

L'applicazione di esempio è costituita da una form che sfrutta il principio dell'iteratore per permettere di navigare all'interno della collezione in due modi diversi e indipendenti.

Conclusioni

La conoscenza dei principali pattern di progettazione permette in molti casi concreti di evitare di "reinventare la ruota", ricorrendo invece a soluzioni già note e consolidate.

Spesso, capita di ritrovare all'interno di un pattern la formalizzazione di un ragionamento che ha portato a una particolare implementazione. In questi casi, il pattern può aiutare a comprendere meglio implicazioni e limitazioni della soluzione adottata.

L'applicazione di pattern, in particolare, può evitare l'adozione di soluzioni troppo rigide od orientate alla semplice risoluzione di problemi di programmazione, che alla lunga potrebbero provocare seri problemi di manutenzione.

NOVITÀ

euro 5,00

Un testo pratico e semplice per l'apprendimento di C#, il nuovo e rivoluzionario linguaggio di programmazione, concepito da Microsoft, per l'ambiente .NET. Con spiegazioni abilmente concepite, suggerimenti nati dalla profonda esperienza accumulata negli anni, ed esempi semplici ed efficaci, Baccan prende in esame gli aspetti salienti di C#, sia della versione 1.1 che della **nuovissima versione 2.0.**

corso di C#

Questo testo è l'evoluzione, riveduta e potenziata, del corso a puntate tenuto per rivista DEV di Infomedia. Sono state riscritte per intero delle parti, per renderle attuali, è stato fatto un lavoro di ricontrollo di tutti gli esempi e soprattutto è stata inserita la spiegazione di tutti i nuovi costrutti di C# 2.0.

pagg. 96
ISBN 8881500167
€ 5.00

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

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

Oracoli di Turing e Sistemi Logicamente Aperti

In questo articolo, anteprima assoluta, parliamo di Nuove Frontiere della Computazione: "frontiere" che hanno a che fare con il concetto stesso di computazione oltre il limite di Turing. In particolare sono illustrate due teorie della computazione, unite da un mix di idee nuove, tra cui la ricerca sui "sistemi logicamente aperti".

di Ignazio Licata

Gran parte dei progressi dell'informatica sono rivolti ad ampliare le risorse dei sistemi di calcolo in termini di tempo/velocità di elaborazione e capacità di memoria. In particolare ci si è concentrati sul "collo di bottiglia" di Von Neumann – computazione *step by step* –, ideando architetture massivamente parallele che distribuiscono l'elaborazione tra più nodi di calcolo/memoria interconnessi. In alcuni casi si è fatto ricorso anche a materiali diversi per realizzare le porte logiche, in alternativa agli attuali chip di silicio; è il caso dei bio-chip o del calcolatore ottico. In nessuno di questi casi però la teoria fondamentale della computazione che deriva dal formidabile lavoro di Alan Turing, *On computable numbers* (1936), viene messa radicalmente in discussione. In questa sede vogliamo proporre alcune considerazioni altamente speculative sulla possibilità di definire nuovi modelli di computazione, equivalenti o addirittura più potenti di quello offerto dalla Turing-computabilità.

Il limite di Turing

Alan Turing mise a punto la sua famosa Macchina di Turing (MT) considerando ciò che un essere umano faceva effettivamente quando eseguiva un calcolo. Diversamente da quello che si sarebbe portati a pensare oggi, il problema non aveva una vocazione puramente applicativa, ma era legato piuttosto al programma di *assiomatizzazione* di D. Hilbert ed ai limiti messi in luce nel 1931 da K. Gödel con i suoi famosi *teoremi di indecidibilità ed incompletezza*: non esiste un sistema inferenziale deduttivo in grado di rispondere della verità o falsità di ogni proposizione costruibile tramite *funzioni ricorsive enumerabili*. In pratica una MT fornisce il contesto matematico corretto ed universale per la definizione generale di algoritmo, in accordo con la *Tesi di Church-Turing*: Le MT sono una classe di automi computazionalmente universali.

Al di là dei vari formalismi (*macchine a registri, sistemi di produzione di Post,Lambda-Calcolo di Church,Grammatiche di Chomsky,* ecc.), la MT resta il modello concettuale più brillante e chiaro della computazione, e può essere descritta come un nastro infinito sul

***Ignazio Licata** è un fisico teorico. Si occupa di sistemi complessi e processi cognitivi. Nel 1998 ha fondato l'Isti–tuto di Cibernetica Non–Lineare per lo Studio dei Sistemi Complessi. Può essere contattato tramite e–mail all'indirizzo ilicata@infomeda.it*

quale un automa a stati finiti può leggere, scrivere e manipolare un numero finito di simboli operando secondo un set di regole applicate in tempo discreto, passo dopo passo. Queste poche osservazioni suggeriscono che la nozione di Turing-computabilità (T-comp) nasce in un ambito altamente formalizzato ed è mirata all'analisi sintattica dell'informazione. In realtà il concetto d'informazione ha una portata molto più ampia, che possiamo comprendere rifacendoci ai sistemi fisici. In natura non esiste informazione che non sia associata a qualche forma di materia-energia discreta e/o continua; ogni sistema fisico o biologico può essere considerato un elaboratore di informazione in cui il valore di alcune grandezze in entrata viene trasformato in uscita dalla dinamica interna del sistema. In questo senso si può dire che ogni sistema fisico "computa", e bisogna prendere in considerazione altri aspetti dell'informazione: la dinamica (la distribuzione spazio-temporale dell'informazione) e la semantica (il valore che l'informazione ha per il sistema, questione particolarmente importante nello studio dei sistemi biologici). Possiamo dunque chiederci se i nuovi scenari relativi allo studio dei sistemi caotici, quantistici e biologici suggeriscono nuovi modelli di computazione.

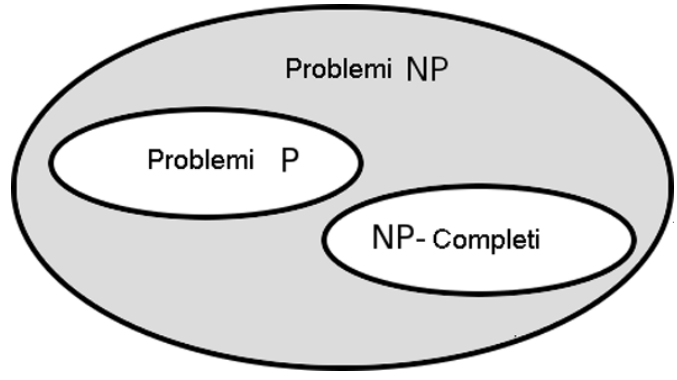


Figura 1 Classi di complessità P ed NP. La "trattabilità" computazionale di un problema conduce ad identificare delle classi di complessità in relazione al tempo richiesto di Turing–computabilità. I problemi P richiedono in genere una quantità polinomiale di tempo di lavoro di un algoritmo deterministico. Gli NP sono risolvibili attraverso algoritmi non–determistici che incorporano elementi random e necessitano di un tempo esponenziale, ma la loro soluzione può essere verificata in tempo finito.

Le MT sono caratterizzate da alcune limitazioni fondamentali strettamente connesse ai teoremi di K. Gödel, e globalmente indicate come *limite di Turing.* È possibile dimostrare che le MT sono un insieme *infinito numerabile* (cioè possono essere contate), mentre l'insieme di tutte le funzioni che si possono costruire ha la *potenza del continuo*, ossia sono un insieme infinito non numerabile. È piuttosto intuitivo comprendere che questo risultato è collegato all'aspetto discreto della T-comp, ed al modo di enumerare le MT. In altre parole, il grado di "infinità" delle MT è più basso di quello dell'insieme delle funzioni costruibili, e dunque esistono funzioni non computabili secondo Turing (*funzioni non-ricorsive*). Un esempio classico è dato dal problema della fermata (*halting problem*): dato un algoritmo ed una stringa è possibile dimostrare che non esiste alcuna MT in grado di decidere se il processo di computazione avrà termine o no, ossia se l'algoritmo produrrà o meno quella stringa. I computer non sanno mai quando è il momento di smettere! Può essere utile vedere il risultato in termini di linguaggi formali: con gli strumenti del linguaggio è impossibile stabilire se una certa espressione appartiene al linguaggio o meno, dunque: esistono linguaggi ricorsivamente enumerabili ma non ricorsivi (teorema sui linguaggi ricorsivi).

Intuitivamente è possibile connettere questa limitazione ad una "rigidità" delle MT, dei linguaggi e dei sistemi formali. Infatti è possibile mostrare che per superare un singolo problema di indecidibilità – ma ne resteranno sempre infiniti, in accordo all'universalità dei risultati di Gödel-Turing! – nuovi assiomi devono essere aggiunti al sistema, ossia è necessario modificare la struttura del sistema. Siamo ormai talmente abituati a convivere con l'ovvietà di questo risultato da non riuscire ad apprezzare il gioco d'implicazioni a cui condurrebbe la violazione di tale limitazione.

Oracoli e sistemi logicamente aperti

Consideriamo una struttura ibrida del tipo (MT, O), dove con il simbolo O indichiamo un dispositivo ideale chiamato *oracolo*. Nella sua accezione più ampia un oracolo può essere considerato come un insieme di procedure di decisione in grado di vincolare l'attività della MT. Potrebbe esse-

Tabella 1 Le caratteristiche significative della computazione naturale rispetto alle Macchine di Turing

Turing Computabilità	Computazione naturale-Sistemi formali continui
Programmabilità; il processo termina dopo un numero finito di passi; problemi di indecidibilità, halting–problem	Capacità evolutive auto–organizzanti; il processo non termina, modifica i suoi obiettivi in relazione al contesto
Informazione discreta; logica binaria; distribuzione sintattica dell'informazione	Informazione continua; logica a valori multipli; pat–tern continui nello spazio–tempo
Processo deterministico; emergenza computazionale	Elementi random; emergenza intrinseca
Dati esatti; output definito	Dati "fuzzy"; strategie di ottimizzazione adattiva
Seriale/Lineare	Parallela/Non–Lineare

re una tabella di valori, oppure una "black-box". Ciò che qui conta è rendersi conto del tipo di limitazioni di una MT. Per definizione, un oracolo è dunque in grado di superare i limiti di una MT, ad esempio fornendo l'algoritmo di un insieme non ricorsivo rispetto alla MT, e risolvendo così l'halting problem. In pratica, l'accesso all'oracolo renderebbe una macchina ibrida capace di affrontare classi di problemi che nessun sistema algoritmico può risolvere. Un altro campo in cui gli oracoli hanno mostrato una funzione teorica è nella classificazione dei problemi "trattabili" o "intrattabili", risolvibili cioè in tempo polinomiale o esponenziale in relazione alla dimensione n del problema. Lo studio delle classi P ed NP è uno dei grandi problemi aperti della matematica, ma alcune convincenti argomentazioni hanno messo in luce che i problemi del tipo "NP-completo" (come i problemi di ottimizzazione, dal "commesso viaggiatore" al *folding protein*), possono essere ridotti a problemi "trattabili" con algoritmi in tempo polinomiale con l'ausilio di oracoli. La stessa linea di ragionamento ha mostrato però che il tipo di soluzione dipende strettamente dalle caratteristiche dell'oracolo adottato, e dunque non ha un carattere universale ma piuttosto relativo.

È possibile gettare nuova luce sulle caratteristiche delle MT ed ibride utilizzando la teoria dei *sistemi logicamente aperti* (LOS). Com'è noto, tutti i sistemi fisici possono essere classificati utilizzando tre categorie principali: 1) i sistemi conservatori di informazione, che conservano l'energia; 2) i sistemi distruttori di informazione, soggetti a dissipazione; 3) i sistemi amplificatori di informazione, dove si hanno fenomeni di *emergenza* e di *auto-organizzazione* in tempo polinomiale o esponenziale. In questa terza classe rientrano i sistemi logicamente aperti, descritti tramite *un modello formale logicamente aperto, ad esempio un sistema non Turing-Computabile*, per il quale è impossibile stabilire una procedura effettiva per stabilire a priori quale informazione è rilevante per la descrizione del sistema. Questo perché l'informazione viene fornita al sistema in modo non-algoritmico (*random*) e soprattutto perché il gioco di interrelazioni tra sistema ed ambiente è particolarmente complesso e porta ad una continua riorganizzazione strutturale del sistema che non può essere esplicitata tramite una semplice procedura ricorsiva. In questa classe rientrano i sistemi *non-lineari* e *context-dependent*, come gli sciami o esseri collettivi, e le reti neurali. Questi sistemi presentano un tipo di emergenza detta *intrinseca* o *osservazionale*, non riconducibile ad una produzione computazionale. È possibile ordinare in una gerarchia di ordini di complessità i LOS, e dimostrare che è impossibile descrivere un sistema logicamente aperto per mezzo di un singolo modello formale, ossia una MT, e che un sistema ad alto grado di apertura logica può essere soltanto approssimato da un altro con apertura di grado n, con n interpretabile come numero di vincoli che formalizzano le relazioni tra sistema ed ambiente. D'altra parte, una MT può essere identificata con un tipico *sistema logicamente chiuso* (LCS), la cui architettura globale non cambia, o comunque può sempre essere ricavata da una procedura effettiva, ragione essenziale degli scacchi storici dell'IA "forte". Arriviamo dunque al punto essenziale:

Una MT è un sistema logicamente chiuso, un oracolo è un sistema logicamente aperto.

Anche se un oracolo ideale non esiste, ed è frutto del genio speculativo di A.Turing, è possibile disporre di dispositivi con capacità "oracolari" che corrispondono a vari gradi di apertura logica, come le reti neurali o gli agenti autonomi dell'IA distribuita. Lo studio di questi sistemi suggerisce un nuovo paradigma computazionale analogico.

Sistemi super & sub Turing

Lo studio dei sistemi logicamente aperti con capacità "oracolari" trova in un gran numero di sistemi fisici la sua ispirazione. Questo corrisponde all'ovvia considerazione che in natura non esistono i problemi di "incompletezza" ed "indecidibilità" che affliggono i nostri sistemi formali. La T-comp è soltanto uno dei modi possibili che possiamo adottare per descrivere il flusso di informazione nei processi naturali e artificiali. Una caratteristica che va rimarcata, oltre alla non-linearità, alla sensibilità al contesto, alle capacità evolutive ed auto-organizzative multi-livello, è la continuità, che situa il "grado di infinità" dei sistemi oracolari alla potenza del continuo. Ad esempio le configurazioni di una rete neurale possono essere parametrizzate tramite un parametro reale connesso ai pesi della rete. In generale, lo studio dei sistemi formali continui, ancora allo stato germinale, ha permesso di individuare le caratteristiche significative della computazione naturale rispetto alle MT (vedi **Tabella 1**).

Ma in pratica, quali sistemi fisici possono essere effettivamente utilizzati per realizzare dispositivi dalle capacità super-Turing? Esistono in questo senso degli "indizi" interessanti, tutti piuttosto recenti, che in questa sede possiamo soltanto menzionare. Esistono classi di sistemi dinamici che possono risolvere formulazioni particolari dell'halting-problem. Le *reti neurali ricorrenti*(RNR) a modificazione sincrona e con pesi reali possono ottenere ottime performance su problemi NP-completi, come i sistemi DAI asincroni, ed anche alcuni algoritmi quantistici mostrano di poter risolvere in tempi polinomiali problemi di difficoltà esponenziale per sistemi classici di computazione. Stesse indicazioni provengono dal DNA-*computing*, dove le informazioni opportunamente "incise" sul DNA vengono elaborate da processi enzimatici. Infine, anche il *chaos computing*, erede delle tecniche di controllo dei sistemi caotici, sta entrando in una fase matura, ricca di grandi possibilità su problemi di ottimizzazione.

È possibile dunque immaginare una tecnologia di oracoli neuro-fuzzy, caotici e quantistici. Il prezzo da pagare è l'universalità. In tutti i casi citati infatti è possibile fornire dei contro-esempi in cui la prestazione viene compromessa da piccole variazioni dei parametri in gioco, in corrispondenza di ciò che si delineano essere come *soglie critiche* della capacità computante. Al di là dei casi particolari, esiste una considerazione intuitiva che può aiutarci a comprendere questa convivenza di capacità super & sub Turing. I sistemi logicamente aperti presentano fenomeni di emergenza intrinseca e dunque sono soggetti ad un forte grado di impredicibilità logica. La computazione "oracolare" appare dunque come una forma di computazione "dedicata", costruita sul particolare problema e sulle caratteristiche del sistema. La cosa non dovrebbe sorprenderci, perché conosciamo bene un dispositivo di questo tipo, messo a punto dall'evoluzione in millenni di "prove ed errori": la mente umana.

Bibliografia:

[1] M. Davis, "Il calcolatore universale", Adepti, Milano, 2003;

[2] M. Davis, "Computability and Unsolvability", Dover, NY, 1982;

[3] J.W.Dawson Jr., "Dilemmi logici. La vita e l'opera di Kurt Gödel", Bollati-Boringhieri, Torino, 2001;

[4] A. Hodges, "Storia di un enigma.Vita di Alan Turing 1912-1954", Bollati-Boringhieri, Torino,1991.

[5] G. Chaitin, "Information, Randomness and Incompletness", World Scientific, Singapore, 1990;

[6] J.L.Casti & A.Karlqvist (eds.), "Boundaries and Barriers", Addison-Wesley,1996;

[7] I. Licata, "Verso una teoria generale dei sistemi intelligenti", neuroscienze.net, vol.1, n.1, 2004;

[8] C. Hewitt, "The Challenge of Open Systems", in Byte, 10, 1985;

[9] C. Hewitt, "Open Information Systems: Semantics for Distributed Artificial Intelligence", Artificial Intelligence, 47, 1991;

[10] G. Minati, M.P. Penna, E.Pessa, "Thermodynamical and Logical Openness in General Systems", Systems Research and Behavioral Science, 15, 1998;

[11] H.T.Siegelmann, "Computation Beyond the Turing Limit", Science, 268,1995;

[12] H.T. Siegelmann, "Neural Networks and Analog Computation.Beyond the Turing Limit", Birkhäuser, Boston, 1999;

[13] B. MacLennan, "Transcending Turing Computability", Tech. Rep. UT-CS-01-473, Dept. Comp. Science, Univ. Of Tennessee, Knoxville, 2001;

[14] D. Deutsch e R. Jozsa, "Rapid Solution of Problems by Quantum Computation", Proc. Royal Soc., A439, 1992;

[15] A. Berthiaume e G. Brassard, "Oracle Quantum Computing", Jour. of Modern Optics, XLI, 1994.

ULTIME USCITE!

Cosa c'è nei capitoli di questo libro? Il percorso segue una logica progressiva. Si parte spiegando cos'è una tecnica di Intelligenza Artificiale (IA) e come si sviluppi per gradi di complessità, astrazione e generalizzazione. Si prosegue con la Bioinformatica, disciplina che consente di manipolare stringhe di DNA. Questa automazione consente di capire la scienza degli Algoritmi Genetici e di codificare le informazioni in modo differente.
Il passo successivo, è una introduzione all'IA simbolica, per la rappresentazione della conoscenza e la risoluzione dei problemi. Allontanandosi volutamente da certe elaborazioni "classiche", il libro ricorre al supporto degli algoritmi genetici (AG) per la risoluzione di varie tipologie di problemi: dalla computazione evolutiva, alla predizione dei sistemi caotici, alla rappresentazione della conoscenza.
Infine, si affronta l'esplorazione del mondo dell'IA connessionista: attraverso le reti neurali è possibile metter a punto modelli di apprendimento e strutture di riconoscimento.
La panoramica è completa e abbraccia le tematiche più importanti ed essenziali dell'IA. Il tutto corredato con esempi pratici e listati di codice in Visual Basic, linguaggio che per diffusione e semplicità consente sia ad un pubblico di professionisti sia di amatori di sperimentare in pratica i temi affrontati nel testo.

pagg. 112
ISBN 8881500183
€ 12.00

disponibile SUBITO

disponibile SUBITO

pagg. 240
ISBN 8881500175
€ 18.00

Questo libro è concepito non solo come integrazione dei corsi di laurea in Ingegneria Informatica o Scienze dell'Informazione (con particolare riguardo all'ingegneria del software ed ai sistemi informativi), ma anche come valido strumento per tutti coloro, professionisti o appassionati, che desiderano acquisire maggiori conoscenze sulla progettazione del software attraverso UML. Il testo descrive un case-study didattico di progettazione e prototipazione di CRM (Customer Relationship Management) attraverso UML: lo scenario da modellare è una moderna azienda di import/export. Dopo un studio di fattibilità preliminare, si procede con l'analisi dei settori Sales Force Automation, Marketing Automation, Advanced Sales Configuration, Partner Relationship Management, Multichannel Contact Center & Field Services, Storefront e Self Service Application. Nella parte finale viene descritto un prototipo in ASP e C++, implementando la base dati relazionale in Access. Tuttavia, non mancano considerazioni sugli sviluppi possibili in ambito aziendale, utilizzando ulteriori prodotti e tecnologie, tra cui Oracle, Corba e COM. A corredo del libro tutti i diagrammi, i codici sorgenti e gli applicativi sviluppati possono essere prelevati dal sito web.

GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

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

Un cluster in 15 clic

Specialità del giorno: soluzioni pratiche invece della teoria

di Ingo Rammer

Nelle ultime puntate di questa rubrica, ho scritto abbastanza di design e di sviluppo di applicazioni scalabili. Qualcosa che vi devo per completezza è una dimostrazione pratica di come installare un cluster e quali sono i vantaggi che se ne ricevono.

In effetti non è in realtà un classico argomento di architettura del software. Ho comunque deciso di mostrarvi questo aspetto pratico, poiché ho osservato che molti sviluppatori e architetti non sanno come sia facile creare un cluster "load balancing" con Windows.

Se avete seguito i numeri precedenti della rubrica Architect's Corner, avrete certamente notato che sono un fautore delle applicazioni scalabili, applicazioni che possono essere distribuite su un cluster di macchine per utilizzarne l'ulteriore potenza di elaborazione. Ma un attimo! Utilizzare un cluster può non solo migliorare il throughput dell'applicazione ma permette di ottenere anche dei sostanziali vantaggi, anche se a lavorare con l'applicazione sono solo alcuni utenti.

Una macchina va giù... e nessuno lo nota

Provate a immaginare che i componenti di servizio (web service, remoting e anche siti web) siano in esecuzione su due macchine in parallelo.

Immaginate anche che entrambe le macchine si comportino in modo che dall'esterno si veda un singolo indirizzo IP. In questo scenario, si può non solo gestire un carico di lavoro superiore, ma si è anche protetti dalle interruzioni di servizio del sistema: se una delle due macchine va in errore, l'altra può gestire semplicemente il carico ulteriore.

L'utente noterebbe questa "avaria" solo perché le prestazioni dell'applicazione si degradano di una certa percentuale.

Tuttavia, l'applicazione non andrà in errore e continuerà a funzionare.

Qualcosa di analogo sarà vero anche per l'installazione dei service pack e degli aggiornamenti del sistema operativo: si possono rimuovere semplicemente singoli nodi dal cluster, applicare l'aggiornamento a questi nodi, eseguire il reboot se necessario, e riunirli al cluster in seguito.

Possono anche esistere cluster in cui vi è un mix di sistemi operativi, dove alcune macchine utilizzano Windows NT 4.0, alcune Windows 2000, e altre che sono già state aggiornate a Windows Server 2003.

Fantasticherie? No. È necessario dell'hardware speciale? No. È costoso? No! La funzionalità che ho descritto è già compresa nel sistema operativo Windows Server 2003 (e anche di Windows Server 2000, ma con meno Wizard GUI

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.*

fantasiosi per effettuare la configurazione). Tutto ciò che è necessario per implementare Windows Network Load Balancing (NLB) sono due server (sino a 32) standard.

Non devono essere identici e non necessitano di hardware speciale: a scopo dimostrativo, si potrebbero anche utilizzare due notebook.

Elementi di NLB

Contrariamente ad altre soluzioni di load balancing (come le soluzioni dedicate Cisco), Windows NLB non richiede un sistema dedicato per eseguire il dispatching delle richieste in ingresso. Invece, tutte le macchine in un cluster comunicano tra di loro per capire chi deve gestire la prossima richiesta in ingresso.

Ciò riduce il potenziale degli errori critici (poiché rimuove il cosiddetto single-point-of-failure) e minimizza l'investimento, poiché non è necessario acquistare ulteriore hardware.

Per illustrare questa situazione, supponiamo di avere due macchine server differenti (SERVER01 e SERVER02) con i seguenti indirizzi IP:

SERVER01: 192.168.0.41

SERVER02: 192.168.0.42

Queste due macchine devono essere connesse a un cluster con l'indirizzo IP virtuale (VIP) 192.168.0.40, in modo che i web service, i componenti del remoting e i siti web possano essere distribuiti in modo "load balanced".

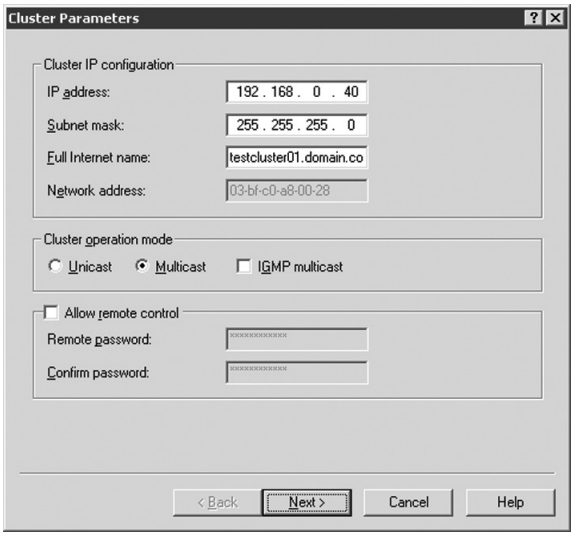


Figura 1 Creazione di un nuovo cluster

Mondi in connessione

Prima di mostrare passo per passo come si può creare un cluster con circa 15 clic del mouse, permettetemi una nota di cautela: leggete la documentazione (almeno in seguito)! Specialmente se si pensa di utilizzare un cluster NLB in un ambiente di produzione, si dovranno fare diverse scelte che dipendono dalla configurazione della rete.

Queste scelte influenzeranno l'ottimalità della scalabilità del cluster, ma la discussione di questi argomenti è un po' troppo per un articolo come questo.

La documentazione online di Windows NLB contiene alcune ottime liste di controllo di ciò che si deve considerare prima di creare un cluster. Ciò è importante perché una configurazione non ottimale può anche influenzare in modo negativo le prestazioni, la scalabilità, l'affidabilità e la disponibilità dell'applicazione.

Per iniziare a creare il cluster, bisogna loggarsi su una macchina Windows Server 2003 (questa macchina non deve necessariamente far parte del cluster) e avviare il tool NLB manager, raggiungibile da Start → Administrative Tools → Network Load Balancing Manager. In questa applicazione, si scelga Cluster → New, in modo che appaia il dialogo in **Figura 1**.

In questo dialogo, si deve specificare il nuovo indirizzo IP virtuale (VIP) del futuro cluster, la maschera di sottorete, il nome completo del DNS e la modalità operativa. Nella documentazione online di NLB, è presente una lunga e dettagliata discussione di diverse pagine su queste modalità, che possono essere riassunte approssimativamente in queste indicazioni: provare per prima la modalità *Multicast* (senza IGMP, a meno che non si sia sicuri al 100% che la rete lo supporti).

Se questa non funziona nella vostra rete, passate a *Unicast*. Unicast è un po' più lento e non permette una comunicazione distinta tra i nodi (tuttavia tutti i nodi sono sempre raggiungibili dall'esterno del cluster).

Dopo aver premuto il pulsante *Next*, si ha la possibilità di specificare ulteriori indirizzi VIP, che nel nostro caso non sono necessari.

Il successivo dialogo è più interessante: permette di specificare le cosiddette *Port Rules*, che configurano come vengono distribuite le richieste in ingresso tra le macchine del cluster.

Con un doppio clic sulla prima regola di default già esistente, si possono modificare i dettagli come mostrato nella **Figura 2**.

A meno che non si utilizzi lo stato di sessione in-memory per le applicazioni web, si deve sempre modificare le impostazioni di affinità da "Single" a "None" per permettere il miglior *failover trasparente* possibile (ossia, in caso di interruzione dell'operatività di un computer, interviene automaticamente un'altra macchina ad assorbire il carico delle richieste. La transizione è del tutto trasparente all'utente. NdT).

Se non si modifica questa impostazione a "None", ma la si lascia a "Single" o a "Class C", ciò inciderà sulla distribuzione delle richieste in ingresso. Dopo che la prima richiesta in ingresso da un singolo host (o da una intera sottorete di classe C) è stata assegnata in modo random, questa impostazione specifica che le successive richieste che provengono dalla stessa origine (host o sottorete) verranno gestite dallo stesso nodo del cluster.

Ciò è ad esempio necessario se si esegue il deployment di una applicazione ASP vecchio stile (pre .NET) che utilizza lo stato di sessione in-memory. In questo caso, la sessione verrà direttamente associata a una singola macchina del cluster e sarà disponibile solo a questa macchina. È perciò necessario che tutte le richieste in ingresso provenienti dallo stesso utente siano gestite dallo stesso nodo.

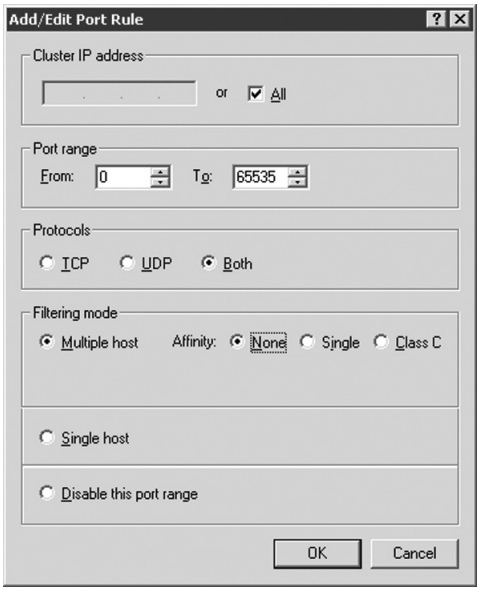


Figura 2 Modifica dell'affinità di sessione del cluster

Per le applicazioni scalabili che sono state sviluppate in modalità cluster-compatibile, questa impostazione ha delle conseguenze piuttosto negative, pertanto di solito si dovrebbe scegliere "None".

Dopo aver completato questo dialogo, si deve specificare l'indirizzo IP del primo nodo del cluster. Per ciascun nodo, si deve anche specificare un ID univoco nel cluster (un valore da 1 a 32). Essendo il primo nodo, si deve selezionare 1 come ID. Completata la configurazione del primo nodo, NLB manager contatterà la macchina specificata utilizzando WMI (Windows Management Instrumentation) per applicare alcune modifiche di configurazione direttamente a ciascun nodo. Nel frattempo, si può continuare ad aggiungere ulteriori nodi nella schermata principale di NLB manager, come mostrato nella **Figura 3**.

Nota: NLB è compatibile con una ampia varietà di hardware di rete. In passato, ho comunque sperimentato dei problemi con le schede di rete on-board, al punto di aver dovuto inserire in uno slot della macchina una seconda scheda di rete per supportare NLB.

Internet Information Server in un cluster

Prima di poter pubblicare i propri Web Service o componenti Remoting in un cluster, si deve eseguire una ulteriore modifica alla configurazione. Per inciso: NLB funziona a livello di connessioni TCP/IP ed è distribuito solo con le richieste di connessione iniziale. Non appena la connessione tra un client e un nodo del cluster è stata stabilita, NLB non prenderà più parte in alcuna ulteriore comunicazione. Naturalmente ciò non dovrebbe comportare alcun problema, se non fosse per il fatto che i Web Service ASP.NET e i componenti .NET Remoting riutilizzano le connessioni già stabilite per lunghi periodi di tempo. Nelle specifiche HTTP 1.1, ciò è detto "Keep-Alive". Ciò significa che una connessione HTTP già stabilita verrà disconnessa solo dopo due minuti di inattività. Ciò ha molto senso per i siti web classici con una gran quantità di elementi <IMG>, perché può migliorare le prestazioni, altrimenti dovrebbe essere creata una nuova connessione TCP/IP per ogni <IMG>.

Nelle architetture application-server ad alta disponibilità, il keep-alive è tuttavia alquanto negativo. Se fallisce un nodo, tutte le macchine che hanno una connessione persistente (keep-alive) ad essa riceverebbero una eccezione. E ciò accade anche se volessero solamente inviare una ulte-

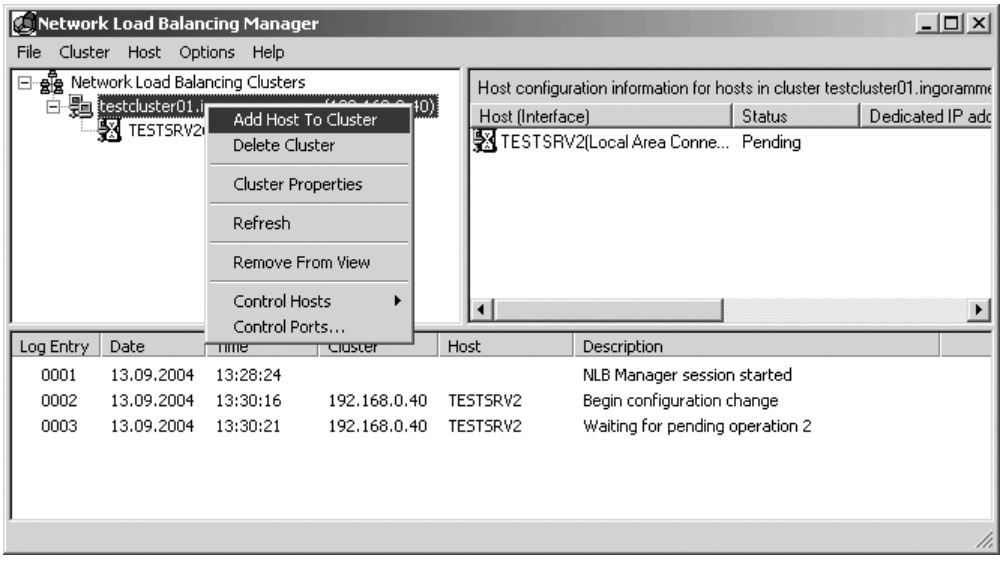


Figura 3 Aggiunta di ulteriori nodi

riore richiesta un po' di tempo *dopo* che la macchina è andata giù. Sostanzialmente, ciò significa che non si ha la possibilità di un fail-over trasparente. Per modificare questo comportamento, si deve aprire lo snapin IIS MMC (Start → Administrative Tools → Internet Information Services). In questo tool, si passa al sito web site che contiene l'applicazione in cluster (questa impostazione è sempre valida per un sito completo e non per una sola directory). Nella prima scheda ("Web Site"), si potrà disattivare la checkbox "HTTP Keep-Alives Enabled". Dopodiché, ogni singola richiesta HTTP sarà soggetta al load balancing. Il costo di ciò in termini di prestazioni è di circa soli 2 millisecondi per ciascuna richiesta in una LAN, e nella gran parte dei casi il potenziale del failover trasparente è ben più prezioso di questa piccolissima degradazione delle prestazioni.

Aggiornamenti a caldo

Per rimuovere un nodo da un cluster in modo controllato (ad esempio per applicare un aggiornamento o un service pack), non va semplicemente disconnesso dalla rete. Altrimenti, tutte le richieste correntemente in esecuzione dovrebbero essere nuovamente inviate dopo che i client ricevono una eccezione. Invece, si deve aprire il tool NLB manager, spostarsi sul nodo da rimuovere, e con un clic destro del mouse selezionare Control Hosts → Drainstop. Questa selezione farà sì che il server termini l'elaborazione delle connessioni esistenti senza accettare alcuna nuova richiesta di connessione. Dopo che questa modalità "drain" è stata terminata, il nodo sarà temporaneamente rimosso dal cluster. Si può quindi installare il service pack ed eseguire il reboot se necessario. Per ricongiungere il nodo al cluster, basta scegliere Control Hosts → Start.

Riassumendo

Come si è visto, la creazione di cluster basato su sistemi Windows non è affatto complessa. Richiede solo alcuni clic del mouse e nessun hardware ulteriore per congiungere due o più nodi in cluster. Tuttavia, NLB distribuisce solo le richieste di rete in ingresso, ma non gestisce le applicazioni installate. È perciò estremamente importante essere certi che tutti i nodi contengano sempre le stesse identiche versioni delle applicazioni. Grazie al deployment XCOPY, di solito questo non è un grosso problema, poiché può essere risolto semplicemente scrivendo alcuni script a riga di comando. In alternativa, si possono utilizzare prodotti come Microsoft Application Center 2000, che si appoggia a NLB, e offre ulteriori tool di gestione per il cluster. Ma, come ho già detto, ciò è del tutto opzionale. Il cluster funzionerà senza questo ulteriore software. Distribuirà le richieste in ingresso tra più nodi e noi trarremo vantaggio dal maggiore "uptime" dell'applicazione, dalla maggiore disponibilità e dal supporto degli *aggiornamenti a caldo*. Cosa state aspettando?

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

Verso la sanità elettronica

di Alberto Rosotti

L'automazione delle procedure cliniche, amministrative e di supporto alle decisioni nelle moderne strutture sanitarie si sta manifestando in maniera sempre più evidente, portando notevoli vantaggi per la salute dei pazienti ed al tempo stesso ponendo sfide sempre più ardue agli operatori del settore: ne è un buon esempio il workflow nei reparti di radiologia.

I vantaggi dell'introduzione dell'informatica e dell'integrazione dei reparti sia clinici che amministrativi sono noti a tutti: si pensi alla possibilità di prenotare un esame tramite Internet o call center, all'uso di diagnostiche complesse come TAC o PET che archiviano gli esami nei fileserver insieme ai referti firmati elettronicamente, alla Telemedicina o alla robotica operatoria.

Mentre in Inghilterra si sta lavorando al progetto di un PACS[1] distribuito, nel nostro paese sono stati stanziati *"44 milioni di euro con l'obiettivo di conseguire una riduzione della crescita della spesa complessiva della Sanità italiana, un innalzamento dei livelli essenziali di assistenza, un incremento della qualità dei servizi percepita dai cittadini, una riduzione ed un maggior controllo dei tempi di attesa delle prestazioni e della degenza ospedaliera",* come affermato dal Ministro per l'innovazione e le tecnologie Lucio Stanca. Fin dall'inizio del nuovo millennio l'imperativo dei CIO[2] sanitari è stato quello di risparmiare, perchè la coperta della spesa pubblica era diventata corta per tutti.

La teoria dello sviluppo sostenibile richiede sacrifici: o si diminuisce la qualità dei servizi, o si diminuisce il numero delle prestazioni, o si allungano le liste di attesa, o si privatizza il servizio pubblico, o si aumentano i ticket o si razionalizzano le prestazioni per abbattere i costi. Le vie più promettenti per risparmiare mantenendo alti i LEA[3] sono l'automazione del flusso di lavoro (workflow) e l'applicazione delle best practice[4].

Ciò comporta, al di là delle possibilità offerte dalle tecnologie emergenti, la formazione degli operatori sanitari e la reingegnerizzazione del workflow, tanto nelle corsie come nei reparti e nei laboratori.

La sanità elettronica

Il sistema sanitario italiano, pubblico per il 65% circa, raggiunge a fatica l'efficienza dei percorsi terapeutici a causa della scarsità di personale, di risorse ed organizzazione. Se da un lato dobbiamo tutti adoperarci per difendere il diritto alla salute così come sancito dall'articolo 32 della Costituzione, è pur vero che i fondi a disposizione sono limitati e vanno trovati nella eliminazione degli sprechi e nella razionalizzazione dei costi.

Partendo da un'analisi della californiana Kaiser Foundation, il Ministro Stanca ha quantificato un risparmio del 2% annuo nella spesa sanitaria nazionale, pari ad almeno 1,6 miliardi di euro, derivante dall'adozione di tecnologie informatiche inserite nell'iniziativa "Sanità elettronica", varata il 16 marzo 2004. Questa iniziativa è stata

Alberto Rosotti *è laureato in ingegnria elettronica. È responsabile sistemi informativi della ditta FBL di Pesaro. Può essere contattato tramite e–mail all'indirizzo arosotti@infomeda.it*

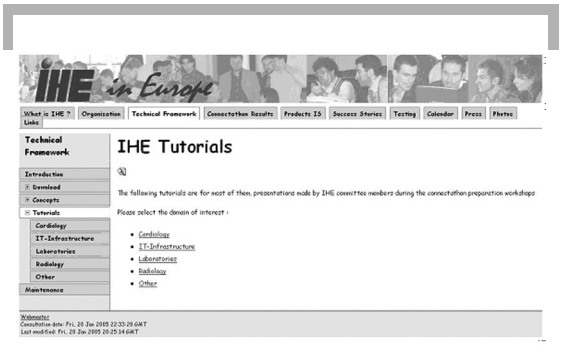


Figura 1 Il sito Internet www.ihe–europe.org

illustrata dal Ministro della Salute Girolamo Sirchia, il quale ha parlato di prevenzione attiva e proattiva, prenotazioni on line integrate a livello nazionale, dati elettronici sulla storia sanitaria dei pazienti e telemedicina. La *prevenzione attiva* è un sistema integrato che consente un ruolo attivo all'azione di prevenzione: la sanità deve andare incontro al malato senza aspettare che sia lui a richiedere cure, intervenendo per esempio in quel 62% di maschi diabetici che non segue alcun trattamento farmacologico. Le *prenotazioni on line integrate a livello nazionale* consentono al cittadino una scelta consapevole dell'offerta sanitaria, fornendo un miglioramento del servizio, una riduzione ed un controllo dei tempi di attesa, oltre alla razionalizzazione dell'offerta che permette di valutare meglio la domanda e ridistribuirla sull'intero territorio. Mira ad informare il cittadino ed a metterlo nelle condizioni di non dover più ricorrere ai "viaggi della speranza". I *dati elettronici sulla storia sanitaria dei pazienti* permettono la condivisione delle informazioni mediche, l'archiviazione ed il retriving sicuro, e fanno riferimento alla realizzazione della cartella clinica automatizzata (CCA), utilissima soprattutto per i pazienti affetti da patologie croniche. La *telemedicina* avvicina l'erogazione di servizi sanitari specialistici ai medici che hanno in carico il paziente, migliorando il servizio e l'assistenza domiciliare e permettendo, ove possibile, la deospedalizzazione.

La tessera sanitaria elettronica

La sanità elettronica prevede l'uso della Tessera Sanitaria Elettronica, promossa dai Ministri dell'Economia, della Salute e delle Finanze, già in distribuzione in Abruzzo, Umbria, Emilia Romagna, Veneto e Lazio a seguito del decreto pubblicato il 4 novembre 2004 sulla Gazzetta Ufficiale. Verrà man mano estesa a tutte le regioni con l'obiettivo di emettere 15 milioni di tessere entro il mese di aprile 2005. *"Sarà così possibile acquisire dati statistici aggregati, ma anche combattere sprechi, abusi e truffe"*, ha dichiarato Stanca, rilevando che grazie alla nuova tessera *"in particolari situazioni geografiche si potranno individuare i casi anomali in cui un cittadino risulta iscritto a medici posti in Asl diverse, rilevare utilizzi abnormi di particolari farmaci o l'emissione di ricette a carico di persone decedute; con pochi interventi mirati si possono acquisire risparmi, razionalizzazione ed efficienza dell'ordine di centinaia di milioni di euro"*. Attualmente sono 800 milioni le ricette che vengono emesse annualmente in Italia, per cui i margini di risparmio sono enormi. I primi dati sembrano dare ragione al Ministro: una Asl, di cui però non è stato reso noto il nome, ha già sperimentato il sistema e con la sola pulizia degli archivi ha risparmiato oltre 200 mila euro. La tessera sanitaria elettronica è simile ad un bancomat e contiene i dati relativi alla salute dell'assistito. Permette l'accesso alle prestazioni ambulatoriali e specialistiche oltre all'acquisto delle medicine necessarie; inoltre può essere utilizzata in tutti i paesi dell'Unione Europea per ricevere l'assistenza sanitaria in sostituzione dell'attuale modello cartaceo E111. La tessera sanitaria riporta i dati anagrafici ed il codice fiscale dell'assistito anche su banda magnetica e codice a barre, oltre ad un codice identificativo unico collegato al medico prescrittore; rappresenta uno strumento per facilitare il contatto dei cittadini con il servizio sanitario e per ricevere più rapidamente sia le cure che i risultati delle visite e degli esami. Sarà facilmente utilizzabile anche dai non vedenti grazie alla presenza di caratteri braille in rilievo e vale come tesserino di codice fiscale. Grazie alla tessera sanitaria elettronica la nuova ricetta medica viene standardizzata a livello nazionale. Se i dati anagrafici riportati sulla tessera fossero errati, il cittadino potrà rivolgersi ad un qualsiasi ufficio dell'Agenzia delle entrate e richiederne la correzione.

Allo stesso modo, in caso di smarrimento si potrà ottenere un duplicato, anche via Internet, accedendo al portale www.agenziaentrate.gov.it; per informazioni è anche disponibile il numero verde 800 030 070.

Per l'introduzione della tessera, la Regione Umbria ha avviato il 10 dicembre 2004 seminari di studio rivolti agli operatori del settore direttamente nel loro luogo di lavoro tramite corsi di formazione a distanza mediante l'uso di CD interattivi e della piattaforma di e-learning accessibile dal sito web www.sistemats.it.

Il progetto IHE

La gestione del flusso di lavoro unitamente alla possibilità di automatizzare alcune procedure, come l'ammissione dei pazienti in ospedale, la gestione delle prenotazioni degli esami, la cartella clinica automatizzata, le schede di dimissione ospedaliera, la compilazione dei DRG[5], le ricerche dei documenti in archivio, può liberare molte risorse medico-infermieristiche che verrebbero convogliate verso la loro primaria attività, quella clinica. D'altro canto l'obiettivo di rendere completamente automatico e digitale il flusso di lavoro all'interno di un ospedale è visto sempre più come un obiettivo ineludibile da parte dell'utente finale, il quale giudica l'assistenza anche da queste caratteristiche ed 'emigra' verso i centri più informatizzati.

Se è vero che le tecnologie sono ormai mature sia dal punto di vista amministrativo che clinico, è altrettanto vero che non sempre si sanno utilizzare al meglio: ciò è comprensibile soprattutto se proviamo a calarci nella realtà di una corsia dove il personale sanitario non ha tempo per risolvere i problemi contingenti, figuriamoci per ottimizzare il workflow.

Per questo è nato IHE (Integrating the Healthcare Enterprise), un'iniziativa sviluppata in America con l'obiettivo di assicurare che nella cura di un paziente, tutte le informazioni richieste per le decisioni mediche siano corrette, immediatamente disponibili ed interpretabili da tutti. IHE è sia un progetto che un forum di discussione (www.ihe.net) che incoraggia l'integrazione delle risorse e l'uso degli standard; un team di ricercatori esegue rigorosi test delle procedure ed organizza sessioni di divulgazione per dimostrare e diffondere i benefici derivanti dall'applicazione delle nuove tecniche. Coinvolge sia medici che produttori di apparecchi medicali con lo scopo di eliminare i gap che ostacolano l'uso di standard quali Dicom ed HL7, spiegando come utilizzarli al meglio per raggiungere gli obiettivi. La sezione europea dell'IHE pubblica manuali scaricabili da internet all'indirizzo www.ihe-europe.org, **Figura 1**, a disposizione di utenti e sviluppatori; tra il 2002 ed il 2003 ne sono stati scritti almeno dieci, che partono da problemi clinici e spiegano dettagliatamente come implementare concretamente le procedure ed ottenere le best practice. Per esempio lo "Schedule Workflow Integration Profile" stabilisce il flusso delle informazioni per la gestione delle immagini digitali del paziente, in merito all'acquisizione, alla consistenza dei dati, allo storage, alla visualizzazione ed alla stampa, mentre "IT Infrastructure Technical Framework" illustra come gestire il dominio di una infrastruttura di rete. Dal 2002 esiste anche la sezione italiana dell'IHE.

Il workflow nel reparto di radiologia

Il reparto di radiologia è tradizionalmente il settore del mondo sanitario in cui la tecnologia e l'informatica sono maggiormente diffuse ed accettate, se escludiamo l'amministrazione dove regnano gli ERP. Modellare il flusso di lavoro in radiologia, cioè schematizzare e sintetizzare il reparto con le sue sfaccettature, umori e varianti umane, difficilmente conduce ad un risultato univoco esprimibile con un software. L'IHE propone un percorso standard che qui presentiamo come un modello di workflow e che impegna diversi attori, vedi **Figura 2**. Il flusso di lavoro comincia con *l'ingresso del paziente in ospedale* e la sua registrazione che avviene tramite un sistema ADT (Ammissione, Dimissione e Trasferimento), un attore che memorizza i dati anagrafici, i recapiti ed altre caratteristiche peculiari. In una successiva ammissione del medesimo paziente sarà sufficiente richiamare i dati già inseriti ed eventualmente aggiornarli. I dati di ammissione dovranno essere disponibili anche ai successivi attori della catena. Dal punto di vista tecnico l'ADT è generalmente realizzato con un'architettura client-server a tre livelli o web-based, che fa perno su un database dalle ottime prestazioni con tempi di risposta immediati.

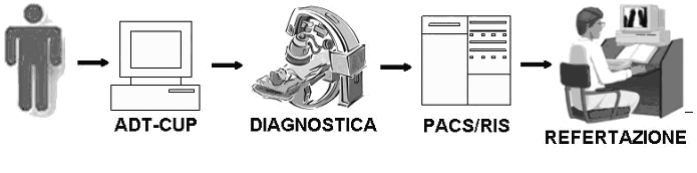


Figura 2 Il percorso del paziente radiologico in ospedale

L'architettura web-based sfrutta le funzionalità dei più comuni browser, come Netscape Navigator o Internet Explorer, quindi risulta facile da gestire ed immediata negli aggiornamenti, sopratutto negli ambienti di lavoro dove sono presenti molti PC; d'altro canto è meno flessibile e più lenta della ormai consolidata architettura client-server.

Successivamente viene *richiesta una prestazione* per il paziente, come per esempio una PET. La prenotazione avviene tramite un sistema CUP (Centro unico di prenotazione) che gestisce sia richieste provenienti dall'esterno dell'ospedale che dall'interno; in questo ultimo caso si opera tramite il sistema informativo ospedaliero. Il CUP individua la miglior data per effettuare l'esame tenendo conto del tecnico disponibile, del parco macchine, degli orari di lavoro e delle necessità del paziente: al temine genera un'agenda di appuntamenti che schedula il workflow e che può essere facilmente consultata, condivisa con l'amministrazione ai fini contabili e modificata. Come l'ADT, anche il CUP dal punto di vista informativo generalmente è di tipo client-server o web based: spesso sfrutta un database proprio, ma è evidente il vantaggio di utilizzare un sistema integrato con il RIS (Radiological Information System) ed il datawarehouse. Alla data prefissata, a meno di imprevisti, *l'esame viene eseguito* sfruttando le funzionalità di un'apparecchiatura diagnostica, comunemente detta "modalità". Il tecnico di radiologia, all'inizio del turno potrà scaricare la lista di lavoro giornaliera, in quanto le modalità attuali sono costituite da workstation (tipicamente Windows o Linux) che si interfacciano da un lato con il RIS ed il CUP e dall'altro con i circuiti di controllo legacy della diagnostica medica. Le modalità di ultima generazione forniscono immagini digitali che vengono immediatamente immagazzinate in un apposito server detto Image Server, sfruttando i sistemi PACS e standard quali Dicom.

Gli Image Server memorizzano le immagini su array di dischi on line e successivamente su supporti più lenti e capienti, come i juke box di nastri capaci di memorizzare petabyte di dati.

A questo punto interviene il medico di radiologia che recupera le immagini dal PACS per *produrre un referto* che sarà firmato digitalmente, archiviato, stampato, consegnato al paziente o ad altri medici e reso disponibile sul web, con tutte le misure di sicurezza per tutelare la privacy, al fine di proseguire nel percorso terapeutico. La refertazione viene eseguita su workstation dotate di due o più monitor ad alta risoluzione per visualizzare molte immagini contemporaneamente, utili nel confronto prima/dopo o per la visione contemporanea da diverse angolazioni, e con l'uso di tool grafici di zoom, contrasto, esaltazione dei contorni, ecc. Inoltre i referti possono essere compilati sfruttando moduli wizard preformattati, oppure mediante dettatura e riconoscimento vocale basandosi sul lessico standard della terminologia medica internazionale ICD9-CM (International Classification of Diseases, versione 9 – Clinical Modification). Grazie alla terminologia ICD9-CM, che classifica più di 15000 patologie e pratiche mediche basandosi su codici numerici di 3 o più cifre (per esempio il codice 001 è assegnato al colera), i sanitari di tutto il mondo possono interpretare correttamente qualsiasi cartella clinica. Se la cartella clinica è automatizzata, dalla sintesi dei suoi dati si può facilmente ottenere la scheda di dimissione ospedaliera sulla base della quale, con un apposito software chiamato "grouper", si assegnano i DRG indispensabili agli ospedali (in Italia) per ricevere i finanziamenti dalle Regioni. Questo giro di informazioni, attualmente a carico di medici e capi sala, se automatizzato con l'adeguato workflow, può fare risparmiare enormi quantità di denaro a vantaggio di tutti, vedi **Figura 3**.

Figura 3 Il flusso dei dati digitali dalla CCA al DRG

Grid computing in ospedale

Quali capacità di calcolo richiede un moderno ospedale completamente informatizzato? La risposta dipende ovviamente da molteplici fattori, quali l'ampiezza delle strutture, il tipo ed il numero di interventi effettuati, l'indice di Case-Mix[6], il numero dei posti letto e del personale impiegato, il livello d'integrazione tecnologica ed ovviamente tanto altro ancora. Il sistema informativo ospedaliero, per il tipo di servizio che deve offrire (disponibilità no-stop e massima sicurezza), potrebbe essere ben servito da un sistema grid computing, cioè un'infrastruttura software adattabile che fa un uso efficiente di server di ridotte dimensioni e soluzioni di storage modulari in grado di bilanciare i carichi di lavoro e garantire una vera capacità on demand. L'amministrazione uni-

Tabella 1 I punti chiave della sanità elettronica

1. La prevenzione attiva
2. Le prenotazioni on line integrate a livello nazionale
3. I dati elettronici sulla storia sanitaria dei pazienti
4. La telemedicina

taria del grid permette di eliminare le inefficienti isole informatiche ottenendo maggiori prestazioni ed affidabilità, trattando tutte le risorse come un servizio virtuale per ottimizzarne l'impiego e la disponibilità. I fattori caratterizzanti il grid computing sono: la virtualizzazione, l'allocazione dinamica e la condivisione delle risorse, l'autoadattabilità, il monitoraggio costante dei sistemi e la gestione centralizzata. Se fino a pochi anni orsono l'imperativo era centralizzare potenza di calcolo e database, tendenza evidente nei data center dei servizi bancari che utilizzano i blade server[7], oggi al contrario si torna a delocalizzare le risorse mantenendo però un controllo centrale e fornendo la disponibilità.

Per far fronte all'imprevedibilità ed all'urgenza in termini di elaborazione, analogamente a come avviene al pronto soccorso quando c'è un'emergenza, i CIO hanno finora fatto ricorso a server sovradimensionati capaci di assorbire i picchi di lavoro; oggi, grazie al paradigma del grid, l'elaborazione viene distribuita dove c'è effettivo bisogno attingendo le risorse computazionali dalle reti disponibili, grandi o piccole, lente o veloci, proprio come un computer attinge la corrente elettrica dalla rete, senza sapere chi la genera o la distribuisce.

Suddividendo il lavoro su più server aumenta la scalabilità, la disponibilità e l'utilizzo delle apparecchiature, ottenendo un ampio risparmio ed economie di scala, in osservanza alle ultime direttive governative. Sebbene la novità intrinseca del grid provenga dall'hardware, la sua potenza risiede nel software e nella capacità dei database e dei server applicativi di gestire la condivisione delle informazioni e della potenza di calcolo. Il grid computing, il computing on demand, l'adaptive computing, l'utility computing, l'organic computing o l'ubiquitous computing sono solo alcuni sostantivi, sinonimi, con i quali il mondo della sanità sta navigando verso un nuovo corso.

Bibliografia & Riferimenti

[1] A. Spada – "L'informatizzazione del Workflow in sanità. Il modello di integrazione IHE", di Andrea Spada, Computer Programming 121
[2] P. Ghedini – "Strutture sanitarie ad alta automazione, governo clinico e standard"
[3] M. Nash – "Oracle 10g: infrastruttura per il grid computing", white paper Oracle, settembre 2003.
[4] www.innovazione.gov.it

Note

[1] **PACS**: Picture Archiving and Communication System. Sistema per la generazione e la gestione delle immagini digitali ospedaliere e delle informazioni ad esse associate.

[2] **CIO**: Chief Information Officer, cioè responsabile del sistema informativo.

[3] **LEA**: Livelli Essenziali di Assistenza, stabiliti nel Piano Sanitario Nazionale dal Ministero della Salute.

[4] **Best practice**: termine usato per intendere il miglior modo di affrontare un problema, derivante dall'esperienza dei maggiori player. Le best practice sono codificate in procedure informative ben definite e trovano ampio spazio nei sistemi integrati di gestione delle risorse, permettendo di raggiungere i migliori risultati.

[5] **DRG**: Diagnosis Related Group. È un metodo per la classificazione dei pazienti dimessi che si basa su raggruppamenti omogenei di diagnosi, utilizzato in Italia come base per il finanziamento delle Aziende Ospedaliere.

[6] **Case-Mix**: indice della complessità degli interventi svolti in ospedale. Esistono vari modi per calcolarlo, uno dei quali consiste nel fare il rapporto tra la somma dei pesi dei DRG ed il numero dei DRG stessi.

[7] **Blade server**: letteralmente server taglienti, affilati, disposti su singole schede indipendenti che contengono essenzialmente processore e memoria RAM, per minimizzare l'ingombro ed ottenere una elevata concentrazione di risorse.

Configurazione e deployment in ASP.NET

Vediamo come ASP.NET risolve molte delle limitazioni che affliggevano ASP

di Dino Esposito

Gran parte del successo di ASP è dovuto alla sua facilità di utilizzo. E in effetti chiunque riesca ad orientarsi nel mondo dello sviluppo per Windows – non importa con quale strumento, da Visual C++ a PowerBuilder passando per Delphi e Visual Basic – non fa soverchia fatica a buttar giù pagine ASP anche relativamente complesse ed articolate.

Lo stesso fatto che per circa quattro anni (un'eternità nel mondo Microsoft) in ASP non sia cambiato quasi nulla dimostra l'assoluta stabilità ed equilibrio raggiunti da ASP.

Il che – non fraintendete – non equivale certo a dire che tutto va ben, madama la marchesa ASP.

Il fatto è che per lungo tempo ad ASP non vi sono state alternative affidabili o praticabili su nessuna piattaforma. E su Windows nemmeno a parlarne (escludo soluzioni proprietarie solitamente alquanto costose e dall'efficacia tutta da dimostrare, per non parlare della documentazione).

Per chi doveva scrivere applicazioni Web (e tantissimi hanno dovuto…), per lungo tempo ASP è stato lo strumento ideale in quanto ottimo compromesso tra prestazioni (mai strutturalmente eccelse), costo e facilità di utilizzo. Di fronte a situazioni in cui non vi sono alternative da invocare, né vi sono precedenti versioni da rimpiangere, il programmatore può solo svolgere al meglio il suo compito istituzionale: scrivere codice e pedalare.

Messi alle strette, tanti programmatori hanno aguzzato l'ingegno e, per i vari problemi "strutturali" di ASP, sono venute fuori soluzioni eleganti e meno eleganti ma tutte allo stesso modo efficaci e risolutive.

Nel corso di questi quattro anni sono state sviluppate un buon numero di best-practice per i maggiori limiti di ASP. Tanto per fare qualche nome: supporto per le Web Farm, form rientranti, oggetti di caching, trasformazioni XSL, trucchi per limitare l'ammasso di roba nella Session, e così via. A tutte queste questioni risponde oggi con indubbia efficacia ASP.NET. E così sia.

Ma la vera tragedia di ASP è il deployment delle applicazioni. Non tanto per ASP stesso, quanto per la strutturale complessità del modello di componenti utilizzato: COM prima e COM+ dopo. Per migliorare la performance e "distribuire" il sistema si è finito per introdurre un discreto numero di componenti che per loro intrinseca natura hanno bisogno di sotto-componenti, interfacce, type library, configurazioni varie e soprattutto del registry e di un buon numero di moduli runtime.

Per far girare poche pagine ASP sono necessari pesanti interventi sul registry e sul metabase di Internet Information Services (IIS) con

Dino Esposito *Si occupa di applicazioni Web–based e lavora come consulente per diverse società. È autore di "Professional Windows Shell Programming" e "Windows Script Host Programmer's Reference" entrambi per Wrox Press. È contributing editor di MSDN Magazine e Windows 2000 Magazine. Può essere contattato tramite e–mail all'indirizzo esposito@infomedia.it.*

creazione e modifica di parametri in vari folder virtuali. Tutta roba da fare, se non a mano, tramite un ottimo strumento di setup. E in ogni caso una serie di operazioni da far tremare i polsi e la memoria. Al punto che di consegna chiavi-in-mano proprio è difficile parlare.

Deployment in ASP.NET

Per fortuna, la consegna chiavi-in-mano di una applicazione ASP.NET è molto più semplice. La differenza però non dipende granché da ciò che rende differente ASP da ASP.NET. A ben guardare, la differenza vera è in ciò che fa da sfondo ad ASP e ASP.NET. E cioé l'impalcatura .NET. Vediamo più in dettaglio perché. I passi fondamentali del deployment di una applicazione ASP.NET sono:

- copia dei file ASPX e quant'altro in un Web folder;
- copia di controlli e servizi nel folder BIN del Web server;
- modifica del file web.config per creare le impostazioni a livello di applicazione e/o di sistema.

Dov'è la differenza? Vediamo più o meno i passi necessari in ASP:

- copia dei file ASP e quant'altro in un Web folder;
- installazione e registrazione di componenti COM;
- creazione delle applicazioni COM+ necessarie;
- tuning delle proprietà di IIS.

Apparentemente non vi è una grandissima differenza. I passi sono più o meno quelli e per le pagine non cambia nulla. La novità si ha a proposito di componenti.

Se il modello è COM(+) allora il registry gioca un ruolo essenziale. Se il modello è .NET allora la *reflection* dà il meglio di sé e per installare e registrare un componente, alla fin fine, basta copiarlo in un percorso pubblico. Il registry è essenziale per individuare nome, ruolo e attributi di un componente COM. I componenti .NET espongono le stesse informazioni da programma tramite le interfacce di reflection: una volta trovato il file DLL il più è fatto. Le informazioni, infatti, sono lì esposte con una serie di interfacce particolari.

Il tuning di IIS è in gran parte manuale e in ogni caso richiede lo stop-and-restart del Web server per ogni blocco di modifiche.

In ASP.NET la stragrande maggioranza delle informazioni sull'ambiente runtime – cioè le informazioni che l'utente programmatore/amministratore può voler modificare – sono memorizzate in un file XML chiamato *web.config*. Esso, oltre ad essere facilmente leggibile e manualmente modificabile, è anche isolato dal resto dell'ambiente, indipendente da crash e affini e non richiede restart in seguito a modifiche – basta che sia possibile rileggerne il contenuto. Cosa che avverrà a partire dalla successiva sessione del successivo utente.

Il file web.config permette di associare e modificare impostazioni in modo dichiarativo invece che procedurale. Può farlo in maniera gerarchica dal momento che le modifiche si estendono dalla directory in cui web.config si trova al sottoalbero. Impostazioni definite nel web.config principale possono essere sovrascritte da web.config trovati a livelli più annidati. Il file web.config principale si trova in una directory sotto winnt\microsoft.net.

Struttura di web.config

Come detto, web.config è un file XML. Tutte le informazioni sono racchiuse da un tag chiamato <*configuration*>. Il contenuto può essere di due tipi: handler e setting. Gli handler sono racchiusi in un blocco centrato in <*configsections*>. I setting hanno nomi di nodi personalizzati e sono tutti diretti figli del nodo radice <configuration>. Ecco lo schema di base:

```
<configuration>
  <configsections>
    <!-- Configuration section handlers -->
  </configsections>
  <!-- Configuration section settings -->
</configuration>
```

I Configuration Section Handlers (CSH) sono dichiarati tramite il tag <add> e fanno riferimento alle classi handler utilizzate per la gestione di un certo tipo di parametri.

Si tratta di classi che implementano l'interfaccia *IConfigSectionHandler*.

Non è necessario dichiarare i section handler in ogni file web.config dal momento che le impostazioni sono ereditate da folder più in alto nella gerarchia del file system. Ecco un esempio di file web.config:

```
<configuration>
  <configsections>
    <add name="sessionstate"
      type="System.Web.SessionState.
```

```
SessionStateModule" />
   <add name="httphandlers"
      type="System.Web.Configuration.
HttpHandlersConfig" />
   <add name="security"
      type="System.Web.Config.
SecurityConfigHandler" />
  </configsections>
</configuration>
```

Per ciascuna delle sezioni di configurazione esiste un apposito sotto-albero con i parametri. Per esempio, le informazioni sul Session Manager sono salvate nel seguente codice:

```
<sessionState
   mode="Off|Inproc|StateServer|SqlServer"
   cookieless="true|false"
   timeout="number of minutes"
   connectionString="server name:port number"
   sqlConnectionString=
"sql connection string" />
```

Il tag SessionState definisce una serie di attributi che permettono di dichiarare come si vuole che funzioni il Session Manager in quella (o quelle) applicazioni ASP.NET.

In particolare, l'attributo mode indica la modalità di funzionamento che può essere in-process (*inproc*), out-of-process (*stateserver*) oppure out-of-process ma basato su SQL Server (*sqlserver*). La quarta opzione (Off) semplicemente disabilita il Session Manager. L'attributo Cookieless indica se l'individuazione delle sessioni client deve avvalersi o meno dei cookie. Timeout è il numero di minuti che dura una sessione in memoria.

Infine, ConnectionString ha senso solo quando la modalità è out-of-process. Esso indica il nome della macchina e la porta dove si trova il server su cui gira il Session Manager.

Questa modalità di funzionamento è particolarmente utile quando i dati di sessione sono molto importanti e non si vuol rischiare di perderli. Le prestazioni, ovviamente, sono un po' peggiori che nel caso default quando la sessione viene mantenuta in memoria.

Se la persistenza delle informazioni di sessione è un aspetto cruciale del software allora si può anche fare in modo di salvare la sessione direttamente su un database SQL Server. In questo caso torna comodo l'attributo SqlConnectionString che specifica la stringa di connessione al database che sarà stato creato in precedenza con tutte le stored-procedure necessarie.

Accesso da programma

Le informazioni di web.config sono accessibili anche da programma tramite un oggetto globale statico che ricorda molto le funzioni Windows 3.1 per accedere ai file INI.

Il metodo da utilizzare si chiama GetConfig e appartiene alla classe statica ConfigurationSettings. Con il codice che segue

```
ConfigurationSettings.GetConfig("sectionname")
```

si riesce a leggere il contenuto della sezione indicata. Ciò che viene restituito non è una stringa o un qualsivoglia altro tipo primitivo. Si tratta invece di una classe SessionStateConfig o una equivalente XXXConfig. Ecco come manipolare da programma le informazioni sullo stato di funzionamento del Session Manager.

```
SessionStateConfig config;
config = (SessionStateConfig) ConfigurationSettings.
                                  GetConfig("sessionState");
if (config.CookieLess == true)
{
   // sessioni senza cookies
}
```

Le informazioni incluse nella sezione <appSettings> che riguardano l'applicazione nel suo insieme sono accessibili tramite la collection globale ConfigurationSettings.AppSettings. ASP.NET rileva automaticamente ogni cambiamento ai file di configurazione e li rilegge e applica le modifiche. Non vi è alcun bisogno che il server venga stoppato e fatto ripartire. ASP.NET protegge i suoi file di configurazione da accessi esterni non desiderati e lo fa impostando IIS in modo da prevenire il direct browsing sui quei file. Ogni tentativo di accedere ad uno dei file di configurazione viene punito con un errore HTTP access error 403 (forbidden).

Conclusioni

In ASP.NET il meccanismo di configurazione, di cui la sicurezza e il session manager sono una parte, utilizza una architettura sostanzialmente gerarchica. Tutte le informazioni sono contenute in uno o più file chiamati web.config, che può trovarsi nella stessa directory dell'applicazione. Le sottodirectory ereditano le impostazioni a meno che non contengano a loro volta un file web.config.

All'interno di questo file, vi sono sezioni con informazioni dettagliate per ciascuna delle principali aree configurabili di ASP.NET.

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

a cura di Davide Mauri
dmauri@infomedia.it

Firefox Web Developer Extensions

Il prodotto in questione, lo anticipo subito, non è strettamente legato a .NET, ma si è ugualmente meritato un posto in queste pagine perché diverrà sicuramente un fidato strumento per tutti coloro che sviluppano applicazioni web, nella fattispecie applicazioni ASP.NET.

Il tool è di fatto una estensione di *Firefox* (c'è anche una versione per *Mozilla*), che permette di analizzare ed applicare modifiche ad una pagina web in tempo reale. Le features disponibili, raggiungibili direttamente tramite una barra di menù aggiuntiva, sono raggruppate in dieci sezioni:

- Disable
- Form
- Information
- Outline
- Tools
- Css
- Images
- Miscellaneous
- Resize
- Other

Nella sezione *Disable Features* sono raggruppate le impostazioni che permettono di disabilitare diverse funzionalità del browser. Si va dai cookies alle immagini, dal javascript agli stili.

Questo permette di vedere come il sito viene visualizzato da un browser non recentissimo, in modo tale da poter intervenire e assicurare una buona fruizione del sito anche a chi usa questo tipo di browser.

In *Form Features* si trovano tutte le opzioni riguardanti i form. Da qui è possibile avere informazioni dettagliate sul form, visualizzare i campi nascosti, convertire azioni GET in POST e viceversa e via dicendo. Tali funzionalità sono molto utili per effettuare debug e test delle applicazioni web.

Tramite le *Information Features* è possibile mostrare tutte le caratteristiche altrimenti non facilmente visibili delle pagine web come gli ancoraggi, le dimensioni reali in pixel dei blocchi *div*, l'*id* dei controlli e simili.

Le *Outline Features* permettono in modo davvero molto semplice di evidenziare la struttura di tabelle, frame e più in generale di qualsiasi porzione della pagina.

Oltre a questo è possibile evidenziare tutti gli elementi deprecati dall'HTML 4.0 così da poterli identificare, ed eventualmente sostituire, in modo piuttosto rapido.

Per quanto riguarda la sezione *Tool Features*, all'interno si trova tutto ciò che concerne la validazione delle pagine web e dei vari elementi collegati.

La sezione più importante e probabilmente più utile è quella delle *Css Features*.

Tramite le funzionalità presenti all'interno della stessa è possibile operare in modo particolareggiato sugli stili utilizzati nella pagina.

È possibile visualizzare, tramite un semplice click del mouse, gli stili applicati su un particolare elemento della pagina. Oltre a questo è possibile modificare in tempo reale gli stili presenti, applicando immediatamente tali cambiamenti alla pagina web.

Questo è reso possibile da un editor di testo interno, che si posiziona nella sezione sinistra del browser, dove viene caricato il codice dei *Css* in uso (**Figura 1**). Ogni modifica apportata ad essi viene istantaneamente applicata alla pagina, rendendo così davvero molto semplice il

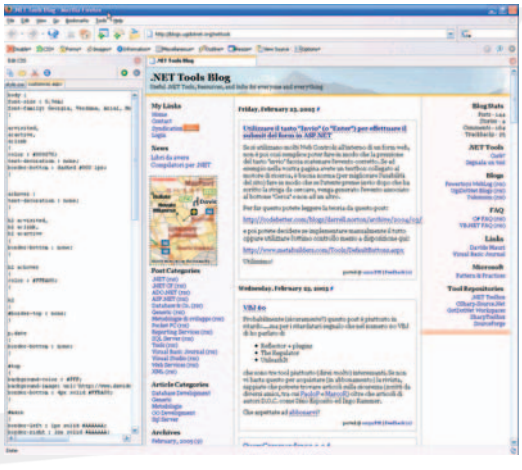


Figura 1 L'editor CSS della Web Developer Extension all'opera

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.*

lavoro di creazione e perfezionamento del layout grafico dell'applicazione web. È importante sottolineare che l'estensione permette di specificare se si desidera rendere i CSS secondo gli standard W3C oppure secondo la metodologia applicata da Internet Explorer.

Non cambia molto tra le due, ma in alcuni casi gli stili vengono applicati in modo diverso. Grazie a questa opzione ci si assicura che gli stili sviluppati vengano ben digeriti da entrambi i browser. Le funzionalità contenute in *Images Features* riguardano le immagini. È possibile, ad esempio, visualizzare tutte le immagini "rotte", oppure quelle senza dimensioni esplicitamente specificate o senza attributo *alt*.

Molto utile la possibilità di evidenziare le immagini che hanno le dimensioni reali diverse da quelle specificate negli attributi dell'elemento *img,* in modo tale da poter ottimizzare il peso della pagina stessa.

Grazie alla sezione *Miscellaneous Features* è possibile rimuovere cookie e cookie di sessione, cache, l'http authentication header e via dicendo: funzioni utili per simulare il comportamento di utenti che non sono mai stati prima sul sito che si sta sviluppando, oppure che non desiderano utilizzare cookies.

Le ultime due sezioni, *Resize* ed *Other Features* permettono rispettivamente di ridimensionare il browser in modo da poter analizzare la visibilità e la fruibilità del sito alle diverse risoluzioni, e di impostare le opzioni dell'estensione di Firefox in modo tale da renderla il più possibile comoda da usare.

In conclusione, un'estensione fondamentale per tutti gli sviluppatori ASP.NET: grazie ad essa assicurarsi che l'interfaccia utente sia visivamente e qualitativamente degna del codice prodotto, sarà molto più facile per tutti.

Prodotto
Web Developer Extension

Url di riferimento
http://www.chrispederick.com/work/firefox/webdeveloper/

Stato Release
0.9.3

Semplicità d'uso ✦✦✦✦✦
Semplicissimo.

Utilità ✦✦✦✦✦
Estrema! Poter applicare modifiche in tempo reale sugli stili delle pagine web significa poter fare un ottimo lavoro in tutta comodità.

Qualità prodotto ✦✦✦✦✦
Ottima.

Qualità documentazione ✦✦✦✧✧
Disponibile solamente online, ma non dovrebbe praticamente servire tanto è semplice l'utilizzo.

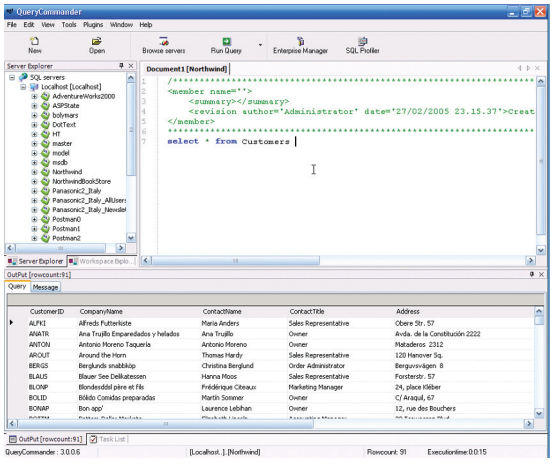


Figura 2 L'ambiente fornito da QueryCommander

QueryCommander

QueryCommander è uno strumento nato per fornire agli utilizzatori di MSDE un ambiente di sviluppo comodo, come quello fornito da Query Analyzer.

Nel tempo si è poi evoluto fino a voler sostituire di fatto il Query Analyzer e non solo, in quanto con la recente versione è stato aggiunto il supporto per database diversi da SQL Server, come Oracle e MySql (**Figura 2**).

L'idea dell'autore è piuttosto ambiziosa dunque (e di strada da fare ce n'è ancora parecchia) ma le premesse sono sicuramente molto buone.

Le funzionalità offerte da questo tool sono davvero numerose, e sono tutte rivolte a rendere più comodo e confortevole lo sviluppo di codice SQL.

Tra le più interessanti c'è sicuramente l'Intellisense, che rende davvero molto più agevole lo sviluppo di script complessi. In tema di script è possibile confrontarne due per evidenziare le differenze che intercorrono tra di essi.

Oltre a queste un'altra funzionalità estremamente utile è la possibilità di poter accedere al codice che definisce un oggetto (ad esempio una stored procedure) dall'interno dello script che ne fa uso; per far ciò è sufficiente posizionarsi sul nome dell'oggetto e scegliere *"Goto Definition"* ed il gioco è fatto.

Un'altra feature comoda per chi sviluppa è la possibilità di inserire degli header di commento. Tali header sono particolari perché contengono elementi XML; grazie a questo, esattamente come accade per Visual Studio, è possibile generare in automatico la documentazione, estraendola dai commenti stessi. Una comodità non indifferente.

Un'idea molto bella è quella della funzionalità *xml2data* che permette di utilizzare un file XML come sorgente di dati, sia per definire la struttura di una tabella, sia per alimentarne il contenuto.

Oltre a queste funzionalità offerte in modo nativo, è possibile sviluppare dei plug-in in modo da estenderne le capacità. Sul sito si trova la documentazione necessaria (poca a dir la verità) e degli esempi da usare come tutorial.

Tra i plugin forniti insieme all'installazione c'è il "*Result to Excel*" che permette di esportare il risultato di una query in formato *xls*: una comodità non da poco, soprattutto nel caso in cui si debba fare un'esportazione ad-hoc o saltuaria.

In conclusione questo è un prodotto che, anche se giovane, è da provare. Personalmente continuo ad utilizzare Query Analyzer per il 99% del lavoro (parlando di SQL Server) visto che ancora quello che mi dà più soddisfazioni (anche se ormai, come ho già detto più volte, mostra tutti i suoi anni), ma se dovessi utilizzare MSDE utilizzerei senza dubbio questo QueryCommander; oltre a questo, inoltre, alcune sue funzionalità sono parecchio comode e ogni tanto mi capita di farne uso ed desiderare di averle integrate nel Query Analyzer… e questo è un ottimo segno.

Prodotto
QueryCommander

Url di riferimento
http://querycommander.rockwolf.com/

Stato Release
3.0.0.6

Semplicità d'uso ✦✦✦✧✧
Abbastanza semplice da usare anche se in alcuni casi un po' di opzioni in più non avrebbero guastato.

Utilità ✦✦✦✧✧
Alta per tutti coloro che usano MSDE o RDMBS privi di strumenti di sviluppo avanzati.

Qualità prodotto ✦✦✦✧✧
Ancora giovane, ma sicuramente promettente. Già a questo livello è comunque più che sufficiente.

Qualità documentazione ✦✦✧✧✧
Non ce n'è traccia. È anche vero che è in pratica un editor di testo mirato al linguaggio SQL quindi non se ne sente molto la mancanza. I nomi dei comandi e delle opzioni sono autoesplicativi.

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

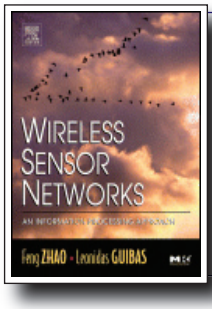
GRUPPO EDITORIALE INFOMEDIA
THE SOFTWARE SIDE OF YOUR MIND

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

IN OFFERTA VBJ 62



Wireless sensor Networks
di F. Zhao e L. Guibas

Ed. Morgan Kaufmann
ISBN 1558609148
376 pp – 72,00 €

Sistemi di basi di dati – Fondamenti 4/E
di R. Elmasri e S. Navathe

Ed. Pearson Italia
ISBN 8871922204
580 pp – 35,00€

Ingegneria del Software 2/E
di C. Ghezzi – D. Mandrioli, M. Jazayeri

Ed. Pearson Italia
ISBN 8871922042
700 pp – 46,00€

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

IN OFFERTA VBJ n. 62

OPPURE inviaci il coupon sottostante al numero di fax

0587/732232

Potrai acquistare i libri qui riportati con uno

SCONTO ECCEZIONALE

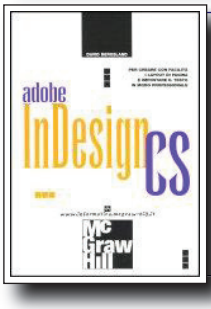
del **10%** anche se acquisti solo **un libro**

OPPURE

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

3ds Max 6 con CDROM
di F. Barrett

Ed. MacGraw–Hill Italia
ISBN 8838643830
450 pp – 36,00€



Adobe InDesignCS
di D. Bergsland

Ed. MacGraw–Hill
ISBN: 88 386 4392–X
512 pp – 31,00€

Aspect–Oriented Database Systems
di Rashid, Awais

Ed. Springer
ISBN 3540009485
176 pp – 60 illus – 3740 €

GRUPPO EDITORIALE INFOMEDIA

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

THE SOFTWARE SIDE OF YOUR MIND — VBJ 62

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 dati 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

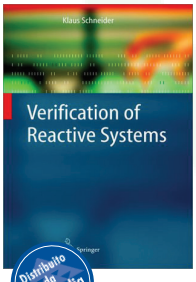
N. _____ SCADENZA _____

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



Verification of Reactive System

Formal Methods and Algorithms

Autore	Klaus Schneider
Editore	Springer
ISBN	3540002960
Lingua	Inglese
Anno	2004
Pagine	600
Prezzo	€ 74,85

Vi sono applicazioni software progettate per sistemi il cui scopo è rispondere a stimoli esterni; questi sistemi sono detti *sistemi reattivi*. Sottoporre i sistemi reattivi ad una serie di test prima del loro rilascio sul mercato è sicuramente necessario ma non sufficiente, per due motivi: il primo è che le possibili condizioni di funzionamento di un sistema del genere sono infinite, mentre il numero di test deve tenere conto delle esigenze di time–to–market. Il secondo motivo è che trovare un difetto di progettazione in fase di test significa trovarlo al termine del processo di sviluppo e dunque implica grossi costi per le modifiche. In questa situazione la matematica viene in aiuto alle buone pratiche ingegneristiche. Se infatti la specifica del prodotto viene espressa in un linguaggio formale non ambiguo, allora si può tentare di dimostrare matematicamente la coerenza interna della specifica e, in certi casi, anche la sua corrispondenza con i requisiti, consentendo così di trovare difetti già nelle prime fasi del processo di sviluppo: questa attività prende il nome di *verifica formale*. Questo volume, unico nel suo genere, si propone di fornire una panoramica completa sulla storia e sullo stato dell'arte di questa materia. Come l'Autore spiega già, con molta onestà, nella prefazione, il testo non è destinato ad un vasto pubblico, neppure tra i tecnici: esso è indirizzato a laureandi, ricercatori e docenti universitari. Nella stessa prefazione, sono anche indicate letture alternative per chi è interessato all'argomento ma ne desidera una trattazione meno formale: per comprendere il contenuto del libro sono infatti necessarie solide basi di logica matematica. Il primo capitolo è dedicato alla storia dei metodi formali. Nel secondo capitolo viene proposto un linguaggio formale di riferimento mentre nei successivi capitoli sono descritti i metodi formali di specifica più usati come gli automi a stati finiti, la logica temporale e quella dei predicati. Ogni metodo formale viene descritto per mezzo di definizioni e teoremi, rendendo quindi il testo adatto allo studio e alla consultazione ma non alla lettura.

Pro

L'Autore propone un esaustivo compendio dei metodi formali di specifica esistenti. Si considera il problema della scrittura di specifiche coerenti e non ambigue, prima ancora di iniziare la fase di sviluppo dell'applicazione.

Contro

La trattazione avviene solamente attraverso definizioni e teoremi; il libro è quindi riservato al pubblico, molto ristretto, dei docenti e dei ricercatori.

Marco Aldrovandi

SMIL 2.0 - Interactive Multimedia

for Web and Mobile Devices

Autore	J. E. Hansen, C. Thomsen
Editore	Springer
ISBN	354020234X
Lingua	Inglese
Anno	2004
Pagine	439
Prezzo	€ 53,45

SMIL è un linguaggio basato su XML, progettato per la realizzazione di applicazioni multimediali su Web e dispositivi "mobili". L'intento degli autori di questo libro è quello di realizzare una guida di riferimento, chiara e al tempo stesso esaustiva, rivolta sia agli sviluppatori che già conoscono SMIL (e magari approfondire la nuova versione o rivisitare determinati argomenti) sia a coloro i quali ne desiderano conoscere le potenzialità. Il testo è suddiviso in quattro parti (una delle quali contenente soprattutto riferimenti); la prima ha uno scopo puramente introduttivo e si occupa di descrivere (mediante esempi) le situazioni in cui è utile impiegare questo linguaggio rispetto ad altre soluzioni. La seconda parte si addentra soprattutto sul discorso dell'architettura su cui poggia SMIL, della sintassi e dei costrutti principali a livello di TAG, mentre nella terza sono approfonditi argomenti di livello più avanzato, come la gestione dei Layout, le tecniche di animazione, la sincronizzazione. Gli autori dimostrano una certa esperienza nel settore soprattutto a livello di approccio alla "materia". Infatti i concetti, pur non essendo di facile livello, sono spiegati in modo chiaro anche grazie alla vasta quantità di immagini (anche di qualità molto elevata) e illustrazioni – presenti in quasi tutte le pagine – con relativi esempi di codice. Il libro mantiene anche una certa impostazione discorsiva, quasi interattiva, ma al tempo stesso schematica; anche il lettore non esperto troverà sempre consigli, ottimizzazioni e spunti interessanti da approfondire a livello pratico. Al termine di ogni capitolo è sempre presente una breve ricapitolazione e vengono pubblicati ulteriori riferimenti presenti su Internet, qualora si intendesse approfondire ulteriormente gli argomenti esposti. Forse la quarta parte poteva anche essere evitata o magari sostituita con ulteriori approfondimenti o esempi, ma comunque alla fine nessuna componente dell'architettura di SMIL viene ignorata. Personalmente questo testo ha soddisfatto le mie aspettative; consiglio di tenerlo sempre a portata di mano durante lo sviluppo.

Pro

Argomenti espressi in modo chiaro grazie anche all'utilizzo di schemi e illustrazioni.

Contro

Nessuno.

Gianluca Masina

Campagna abbonamenti 2005

SI!

MI ABBONO ALLE RIVISTE INFOMEDIA

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	☐	☐	☐	☐
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

CODICE CLIENTE _____

INDIRIZZO DI SPEDIZIONE

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 _____

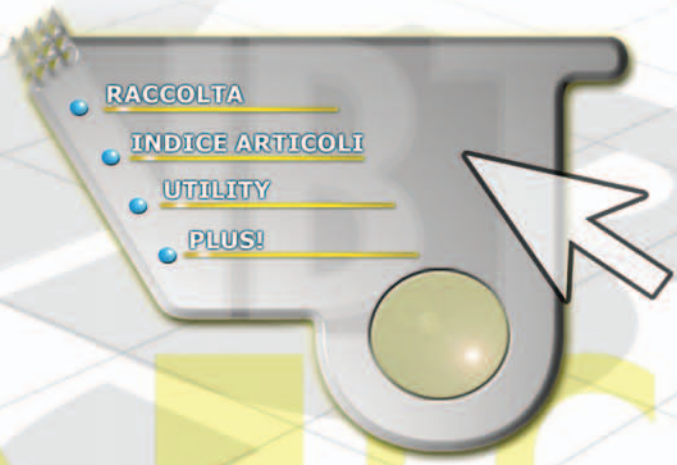
- ☐ Collegati al sito www.infomedia.it dove potrai effettuare il pagamento con carta di credito in modalità sicura (banca SELLA)

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.



VISUALBASIC JOURNAL

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 ☐☐☐☐☐☐☐☐☐☐☐	

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