Magazine di programmazione dedicato a C++, Database, Delphi, Java, Visual Basic, Windows, Linux, .NET...

Computer Programming

ISSN 1123-8526

SETTEMBRE 2007
n. 171 € 5,20

Il primo mensile di programmazione

GRUPPO EDITORIALE INFOMEDIA

Sviluppo col... TURBO!

Gli strumenti Borland/CodeGear per lo sviluppo Windows

e inoltre:

Robotica: Il mattoncino NXT e Java

Low Level: Linux NAPI-compliant network device driver

La programmazione concorrente in **Erlang**

Scrivere un **Web Service** con Eclipse

Java skill: competenze e profili professionali (ultima parte)

Algoritmi di ordinamento: **Quicksort**

OBJECT-ORIENTED DESIGN
Progetto e collaudo di gerarchie d'ereditarietà (2a parte)

OPINIONI
Phishing, è così facile abboccare?

Poste Italiane s.p.a. - Sped. in A.P. - 70% - 171 - Mensile - Decr. Trib. n. 19/1999 - Dir. Resp. Maria Letizia Mari - AUT. DCB/CENTRO1 valida dal 07/07/2005

GRUPPO EDITORIALE
INFOMEDIA

djbdns

pagg. 72
ISBN 8881500205
€ 6,00

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

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

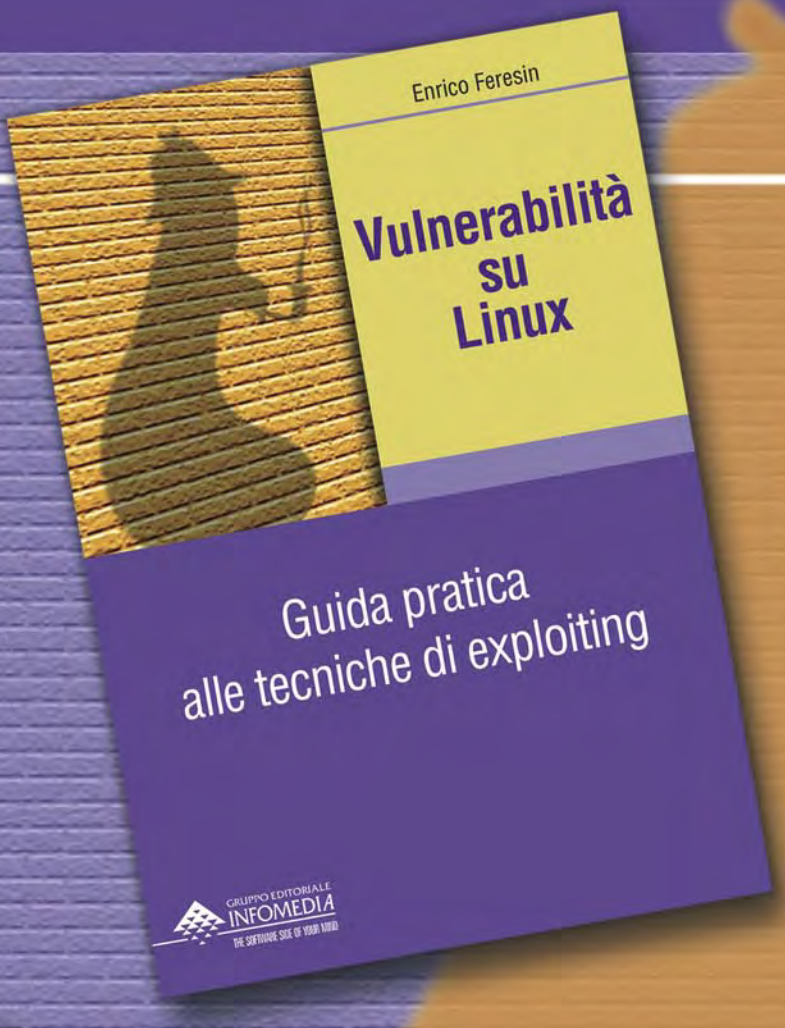
disponibile SUBITO

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



384 pp
ISBN 888150099X
€ 29,00

Vulnerabilità su Linux

Un manuale sullo sfruttamento delle vulnerabilità del software. Un testo che riassume gran parte delle tecniche conosciute di sfruttamento delle vulnerabilità, che permettono l'esecuzione di codice arbitrario.

Queste tipologie di vulnerabilità sono tra le più pericolose e tra le più complicate da sfruttare. È proprio la combinazione di questi due fattori che rende l'argomento così interessante.

La scrittura di exploit per queste vulnerabilità, richiede notevoli conoscenze del sistema, molto intuito e anche fantasia, rappresentando così un'intrigante sfida di programmazione.

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

Software cocciuti

Da anni, man mano che installo applicazioni e tool sui notebook con cui lavoro (tipicamente con una versione di Windows preinstallata), aumenta inevitabilmente il numero di processi o servizi che entrano in azione all'avvio del sistema operativo (e che spesso "insediano" una propria icona nella tray area di sistema). Va da sé, che col passare dei mesi la finestra temporale "post logon" si dilata sempre più, e prima di "riscattare" la proprietà del sistema e avviare il programma con cui intendo lavorare, devo struggermi in paziente attesa per alcuni minuti: il sistema operativo sembra procedere in modalità monotask rasentando quindi un'interattività prossima allo zero. Responsabili di questo "handicap in partenza" sono i molteplici software che all'avvio, per vari motivi, ad esempio contattano la propria "casa madre": da Windows Genuine Advantage della stessa Microsoft, ai vari "update scheduler" e via dicendo. A differenza degli antivirus, che necessitano subito delle firme aggiornate dei virus, questi altri software dovrebbero tener conto che in una tipica giornata lavorativa un sistema resterà sottoutilizzato per diverse ore: perché ostinarsi a eseguire un'azione all'avvio del sistema? Un esempio è Google Updater che si connette al server remoto del servizio (quelle poche volte che lo autorizzo) per informarmi puntualmente che non ci sono nuovi aggiornamenti disponibili. Oppure, da quando mesi or sono ho installato sul mio sistema il software a corredo di una scheda HSDPA, all'avvio parte un processo che serve a rilevare l'eventuale presenza di questa scheda. Non sto a dire che è una scheda che utilizzo solo quando sono fuori sede e in casi di emergenza: per cui non ha senso effettuare il polling in perpetuo di una scheda che inserisco sì e no qualche ora a settimana! Ovviamente, sappiamo tutti benissimo che agendo sulle impostazioni di configurazione o intervenendo nel Registry di sistema si possono escludere e rendere innocui questi software alquanto perniciosi: ma non sempre basta. Ogni volta che elimino dal Registry l'avvio del suddetto monitor della scheda HSDPA, al successivo utilizzo della scheda ci pensa il software di connessione a ripristinare puntualmente l'avvio del processo alla partenza del sistema! Definire questo comportamento "ostinazione" mi sembra davvero poco. E che dire di Windows Update, che dopo l'installazione degli aggiornamenti richiede di poter riavviare il sistema, e se si risponde "Riavvia più tardi" si ripresenta ogni dieci minuti (per default) e ripropone di effettuare il riavvio? Certo, basta intervenire, ad esempio, in *Configurazione Computer* da *Criteri Computer Locale* e reimpostare per Windows Update l'opzione "Rimanda riavvio" per non essere più afflitti (magari a 999.999.999.999.999 minuti). Ma sono io utente che, in assenza di una opzione "Non avvisarmi più", devo insegnare al software le buone maniere mentre sto utilizzando il *mio sistema* per *altre attività*?

Quando scriviamo un software e decidiamo che il processo dovrà partire all'avvio del sistema riflettiamo bene sugli esempi da non imitare; e diamo sempre la possibilità ai nostri utenti di configurare in modo non invasivo il comportamento della nostra applicazione. Soprattutto se a conti fatti siamo i soli a ritenere che si tratti di un'applicazione assolutamente "mission critical", al punto da poter rallentare il sistema all'avvio e anche molestare con costanza l'utente! Dopotutto, si parla spesso di *usabilità del software*.

Marcello Fino

ELENCO INSERZIONISTI

sommario

171

FOCUS

PROGRAMMING

INFORMATICA

sommario

RUBRICHE

All'indirizzo
ftp.infomedia.it/pub/ComputerProgramming/Listati
sono liberamente scaricabili tutti i listati relativi agli articoli e il codice sorgente dell'eventuale progetto software allegato

Computer Programming

n. 171 - settembre 2007

Direttore responsabile
Marialetizia Mari

Direzione Editoriale
Natale Fino (nfino@infomedia.it)

Managing Editor
Renzo Boni (rboni@infomedia.it)

Collaboratori
Ludwig Bargagli
Andrea Baruzzo
Antonino Calderone
Fabrizio Ciacchi
Fabio Fabozzi
Gabriele Fatigati
Federico Feroldi
Giampaolo Marucci
Giuseppe Monterosso
Maria Luigia Nitti
Alessio Saltarin

Grafica e Impaginazione
Valentina Mucci

GRUPPO EDITORIALE INFOMEDIA

Gruppo Editoriale Infomedia S.r.l.
Via Valdera P. 116 - 56038 Ponsacco (PI)
Tel. 0587 736460 (r.a.) - Fax 0587 732232
www.infomedia.it - red-cp@infomedia.it

Ufficio Abbonamenti
Tel. 0587 736460 - Fax 0587 732232
abbonamenti@infomedia.it - http://online.infomedia.it

Direzione
Natale Fino
(nfino@infomedia.it)

Divisione Libri
Lisa Vanni
(book@infomedia.it)

Marketing & Advertising
Tel. 0587 736460
marketing@infomedia.it

Amministrazione
Sara Mattei
(amministrazione@infomedia.it)

Segreteria
Enrica Nassi
(info@infomedia.it)

Stampa
Punto Stampa
Ponsacco (PI)

Si prega di inviare i comunicati stampa e gli inviti stampa per la redazione all'indirizzo:
comunicatistampa@infomedia.it

"Computer Programming"
è una rivista del
Gruppo Editoriale Infomedia S.r.l.
Direzione e Amministrazione:
Via Valdera P. 116 - Ponsacco
Registrazione presso il Tribunale di Pisa n. 19 del 25/11/1999

Questo periodico è associato all'Unione Stampa Periodica Italiana

Borland Turbo C++ 2006: hands on

Mettiamo alla prova il nuovo Turbo C++ con una piccola applicazione CPU intensive

di Alessio Saltarin

Ci eravamo lasciati in Aprile [1] con la prima valutazione di questo nuovo prodotto, il Turbo C++ 2006 di Borland, o meglio di CodeGear – nome della partecipata al 100% che si occupa dello sviluppo degli IDE dall'anno scorso.

Dicemmo allora che si trattava di un prodotto immaturo, ma senz'altro promettente. Soprattutto, vista la dipartita di Microsoft dal settore, il Turbo C++ rimaneva l'unica soluzione RAD (*Rapid Application Development*) per lo sviluppo di eseguibili Windows in codice macchina. Infatti, come sappiamo, il trend seguito da Microsoft è stato invece quello di sostituire i tool che producevano, una volta compilato, codice macchina (*Visual Basic*, *Visual* C++) con strumenti che compilassero in codice MSIL, seguendo quindi i dettami del .NET Framework. Sviluppare oggi un'applicazione per Windows in C++ con interfaccia grafica è possibile solo facendo ricorso al Platform SDK e a un compilatore, andando "a mano" a codificarsi l'interfaccia grafica, oppure con gli strumenti che discendono dal Borland C++ Builder, di cui questo Turbo C++ è l'ennesima incarnazione (la versione 10, per l'esattezza).

Sono passati alcuni mesi e, mentre all'orizzonte appare il nuovo *Borland C++ Builder 2007*, sempre sviluppato da CodeGear, con supporto a Windows Vista, il rilascio della seconda hotfix per il Turbo C++ ci permette di tornare sui nostri passi e riprendere il discorso, magari seguendo passo passo lo sviluppo di una piccola applicazione in C++, che potrete scaricare dal sito ftp di Infomedia.

Alessio Saltarin asaltarin@infomedia.it

È laureato in Ingegneria Gestionale. Si occupa dello sviluppo di applicazioni software enterprise per Vodafone Italy.

Dot Net?

La versione che ho testato e installato si chiama per intero *Turbo C++ 2006 Professional Edition*. Costa circa 400 dollari, ma va detto che ne esistono una versione gratuita e una al costo di 100 dollari destinata agli utenti universitari.

L'installazione richiede la presenza sulla macchina di alcuni prerequisiti, la cui necessità lascia a dir poco perplessi. Si tratta del:

- *Microsoft .NET Framework v1.1*

- *Microsoft .NET Framework v1.1 SP1*
- *Microsoft .NET Framework SDK v1.1*
- *Microsoft J# .NET Redistributable Package*
- *MSXML Parser 4.0 SP2 Parser and SDK*

Anticipo subito che il fatto che, per avere un IDE C++ puro, io mi debba per forza installare il *.NET Framework* rimane il massimo punto a sfavore di questo ambiente. Ne ero scontento mesi fa, continuo a esserlo ora, con tre versioni del Framework sulla mia macchina di sviluppo di cui una (la 1.1) al solo servizio del Turbo C++.

È anche vero che, se vado a cercare quante installazioni ho del runtime di Java sulla stessa macchina, queste sono innumerevoli: il solo client di Oracle ne tiene due per sé. Vi anticipo, in modo che non abbandoniate subito disgustati la lettura, che se la cosa non vi dà eccessivamente fastidio riceverete da questo tool quelle soddisfazioni che già i suoi antenati - a partire dal *Turbo C++ v1.0* per DOS per arrivare agli ultimi *C++ Builder* - avevano saputo regalare.

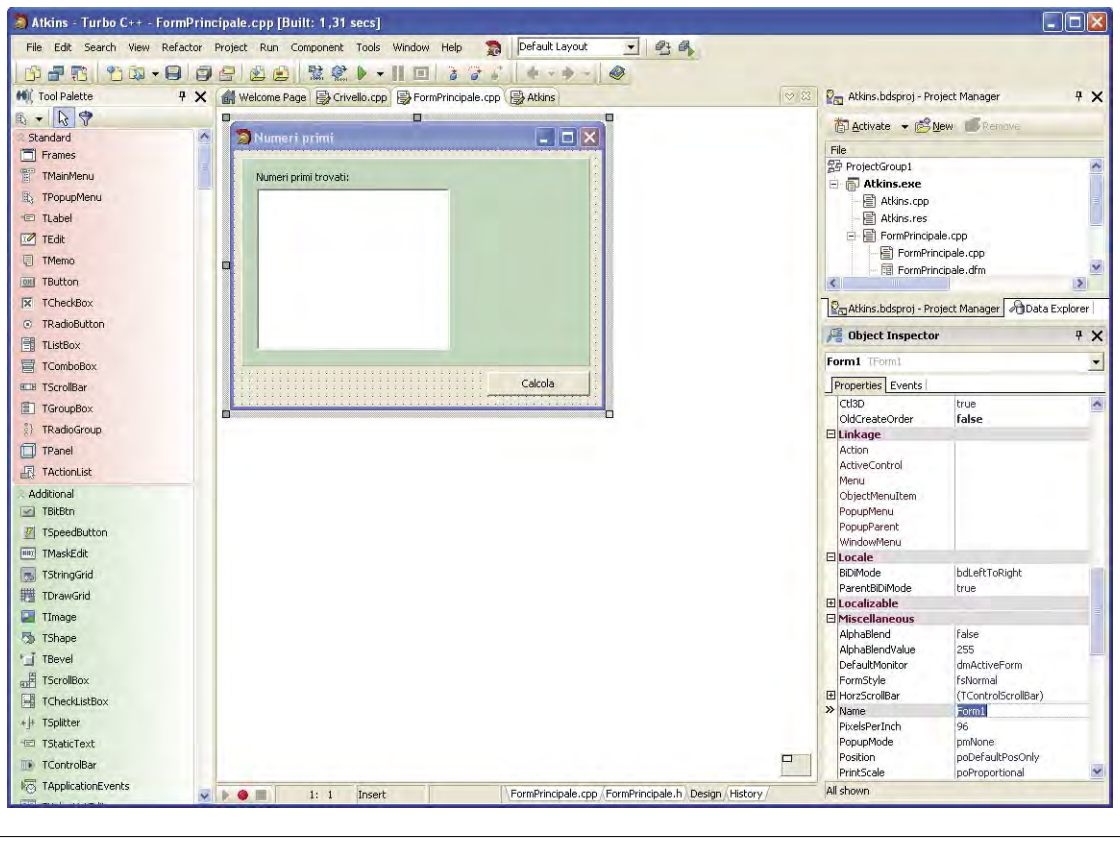
La promessa è che con il Turbo C++ potrete sviluppare un'applicazione Windows 100% nativa per processori x86 in C++. A tutt'oggi dobbiamo dire: scusate se è poco.

La mitica VCL

Il merito della rapidità dello sviluppo e della capacità di questo tool di offrire un ambiente RAD di costruzione delle interfacce alla "Visual Basic", per intenderci, va tutto alle librerie VCL (*Visual Component Library*). Queste librerie, nate per Delphi, incapsulano in maniera ordinata e veramente orientata agli oggetti le API Win32 e pertanto permettono di costruire interfacce grafiche anche complesse al costo di pochi click. Questo grazie ad un potente designer integrato di facile utilizzo e che contiene tutti i controlli di cui possiamo avere bisogno, come si vede dalla **Figura 1**.

È vero che la VCL è rimasta pressoché invariata da molti anni e che, al confronto delle nuove scintillanti interfacce che un'applicazione per .NET può vantare, presenta un look piuttosto *demodé*. Se la cosa però non vi infastidisce più di tanto, sappiate che d'altra parte è piuttosto robusta e soprattutto supportata da un'appassionata schiera di fan, che continua a produrre per essa, a pagamento o come *freeware*, nuovi componenti e soluzioni. Il suo uso è facile e intuitivo anche per chi abbia solo un'infarinatura di C++. Vediamo

FIGURA 1 L'editor grafico di interfacce



Si tratta di un antico procedimento per il calcolo dei numeri primi. Deve il nome al matematico Eratostene di Cirene, che ne fu l'ideatore. È a tutt'oggi utilizzato come algoritmo di calcolo dei numeri primi da molti programmi per computer; pur non essendo un algoritmo straordinariamente efficiente, infatti, è in compenso piuttosto semplice da tradurre in un qualsiasi linguaggio di programmazione.
Il procedimento è il seguente: si scrivono tutti i naturali a partire da 2 fino a n in un elenco detto setaccio o crivello. Poi si cancellano (setacciano) tutti i multipli del primo numero del setaccio (escluso lui stesso). Si prosegue così fino ad arrivare in fondo. I numeri che restano sono i numeri primi minori od uguali a n. È come se si utilizzassero dei setacci a maglie via via più larghe: il primo lascia passare solo i multipli di 2, il secondo solo i multipli di 3, e così via.
Nel caso n = 50, ad esempio, il procedimento di setacciatura si conclude con il numero 7 perché 7 è il massimo intero il cui quadrato non supera 50 e si può provare che il procedimento di setacciatura per ricercare i primi fino ad un certo numero n cessa sempre quando si supera la radice quadrata di n. Infatti ogni numero a del setaccio iniziale, contenente tutti i numeri naturali non superiori ad un dato n, cade dal setaccio che corrisponde al più piccolo dei suoi divisori primi.
Se indichiamo con p il più piccolo divisore primo di a si ha:

$$a = p \cdot r \text{ con } r > p$$

Se ne deduce che

$$a = p \cdot r \geq p \cdot p = p^2$$

da cui **p** è sempre minore o uguale alla radice quadrata di **a**.
(da Wikipedia)

RIQUADRO 1 Il Crivello di Eratostene

perciò di seguito un esempio di sviluppo pratico, "hands-on".

Hands on: impostazioni di progetto

Avete dunque installato il Turbo C++ e volete cominciare a esplorare cosa si può fare con questo strumento.

Scegliendo "*New VCL Project*" dal menù apparirà l'immagine familiare e vagamente rassicurante di una form nuda e di tanti controlli pronti a essere aggiunti secondo necessità. Possiamo costruire la nostra interfaccia col *drag and drop*, cosa che siamo ormai abituati a fare dai tempi del Visual Basic – e che oggi continuiamo a fare, almeno la maggior parte di noi, ad esempio in Visual C# o, per citare un prodotto di Borland, in *JBuilder*. Ma qui si tratta di una vera e propria magia, perché ad ogni colpo di mouse si va formando un codice C++ elegante e completo, diviso in header e sorgente. Il tutto senza che nulla avvenga "di nascosto" e lasciando al programmatore il completo controllo in ogni momento.

L'esempio che voglio proporvi in questo articolo è l'implementazione del *Crivello di Eratostene*, i cui dettagli potete trovare nel **Riquadro 1**. Inseriamo l'algoritmo che trovate nei **Listati 1** e **2**. Per farlo clicchiamo col destro sul nome del progetto nel *Project Manager* e selezioniamo "*Add New...* " "*Unit*". Una unit è l'insieme di due file, l'*header* (.h) e il *sorgente* (.cpp). Diamo loro il nome "Crivello" e cominciamo a digitare l'algoritmo.

Mentre andiamo digitando il codice ci accorgiamo di quanto avanzato sia l'editor, di come svolga molto lavoro per noi suggerendo parentesi, inserendo graffe dove servono, effettuando il syntax coloring e l'highlight delle porzioni di codice che stiamo modificando. Veramente un ottimo editor, secondo forse solo a quello degli ultimi Visual Studio.

Del resto l'interfaccia è spartana, è vero, ma totalmente personalizzabile. Trovo per esempio personalmente molto utile la presenza di un tasto sulla toolbar che compili senza eseguire, tasto che però di default non c'è. È' però molto semplice aggiungerlo, andando a pescare il comando dal menù *Project* per poi averlo sempre disponibile. Una cosa che inoltre appaga il mio senso estetico – e nella quale devo dire i prodotti Borland da sempre si distinguono – è la capacità

LISTATO 1

```
/**
 * Crivello di Eratostene
 * Turbo C++ Hands On
 * Alessio Saltarin 2007
 *
 * File: crivello.cpp
 *
 */

#pragma hdrstop
#include "Crivello.h"
#pragma package(smart_init)

Crivello::Crivello(int maxp)
{
      this->max_prime = maxp;
      this->sieve = new char[this->max_prime];
}

list<int> Crivello::primi()
{
    return this->prime_list;
}

void Crivello::calcola()
{
      int i, currprime;

      // Inizializzazione tabella
      sieve[0] = 0;
      sieve[1] = 0;
      for (i = 2; i < this->max_prime; i++)
      {
               sieve[i] = 1;
      }

      currprime = proxprimo(0);
      while ((currprime * currprime)
                              < this->max_prime)
      {
               for (i = (currprime * currprime);
                           i < this->max_prime;
                                i += currprime)
               {
                        sieve[i] = 0;
      }

               currprime = proxprimo(currprime);
      }

      for (i = 0; i < this->max_prime; i++)
      {
               if (sieve[i] == 1)
               {
                        this->prime_list.push_
                                          back(i);
               }
      }
}

int Crivello::proxprimo(int ultimoprimo)
{
      int i;

      for (i = ultimoprimo + 1; i
                         < this->max_prime; i++)
      {
               if (sieve[i] == 1)
               {
                        return i;
               }
      }

      return 0;
}
```

di personalizzare i colori e i font di ogni singolo componente dell'IDE.

La classe Crivello, illustrata nei **Listati 1** e **2**, è di semplice impostazione. Una lista STL è approntata per contenere gli eventuali numeri primi trovati. Nel costruttore passiamo un intero (*maxp*) che rappresenta il massimo numero entro il quale calcolare i primi. I metodi pubblici *calcola()* e *primi()* fanno il resto, cioè il primo esegue il Crivello di Eratostene sui numeri da 0 a *maxp* e inserisce i numeri primi così trovati nella lista, e il secondo restituisce la lista stessa.

Chiaramente, quando andremo ad utilizzare la classe, invocheremo *calcola()* e poi con un iteratore andremo a stampare i risultati che si troveranno nel metodo *primi()*.

Hands on: disegnare l'interfaccia grafica

Andiamo ora a costruire l'interfaccia grafica. Questa dovrà avere l'aspetto finale che si vede in **Figura 3**. Per costruirla dobbiamo effettuare il *drag and drop* di alcuni controlli sulla form. In primo luogo inseriamo un controllo pulsante (*TButton*) e, attraverso l'*Object Inspector*, andiamo a modificare la sua proprietà *Caption* in "Calcola". Modifichiamo anche le proprietà *Anchors* che stabiliscono il comportamento del controllo durante il ridimensionamento: mettiamo a *true* akRight e akBottom e le altre a *false*. In questo modo il pulsante rimarrà confinato nella parte in basso a destra dell'interfaccia

LISTATO 2

```
/**
 * Crivello di Eratostene
 * Turbo C++ Hands On
 * Alessio Saltarin 2007
 *
 * File: crivello.h
 *
 */


#ifndef CrivelloH
#define CrivelloH

#include <list>

using namespace std;

class Crivello
{
public:
      Crivello(int maxp);
      ~Crivello(void) {}

      void calcola();
      list<int> primi();

private:
      int proxprimo(int ultimoprimo);

      int max_prime;
      char *sieve;
      list<int> prime_list;
};
```

durante un resize.

Inseriamo ora un pannello (*TPanel*) e andiamo a impostare subito tutte le proprietà *Anchor* a true. Possiamo scegliere un colore come sfondo modificando la proprietà *Color*. Io ho scelto *clMoneyGreen*. Azzeriamo *Caption* in modo che non ci sia nessun testo sottostante. Ora, sopra il nuovo pannello posizioniamo una *listbox*, che andrà a stampare i numeri primi via via che il nostro crivello di Eratostene li scoprirà. Per farlo selezioniamo una *TListBox* dall'elenco dei controlli disponibili e posizioniamola a sinistra, con *Anchor* impostati a Top, Left, Right e Bottom.

Infine, per stampare il totale dei numeri primi trovati, immettiamo un controllo *TEdit* a destra della listbox (ma sempre sul pannello).

FIGURA 2 Il codice sorgente dell'applicazione di esempio

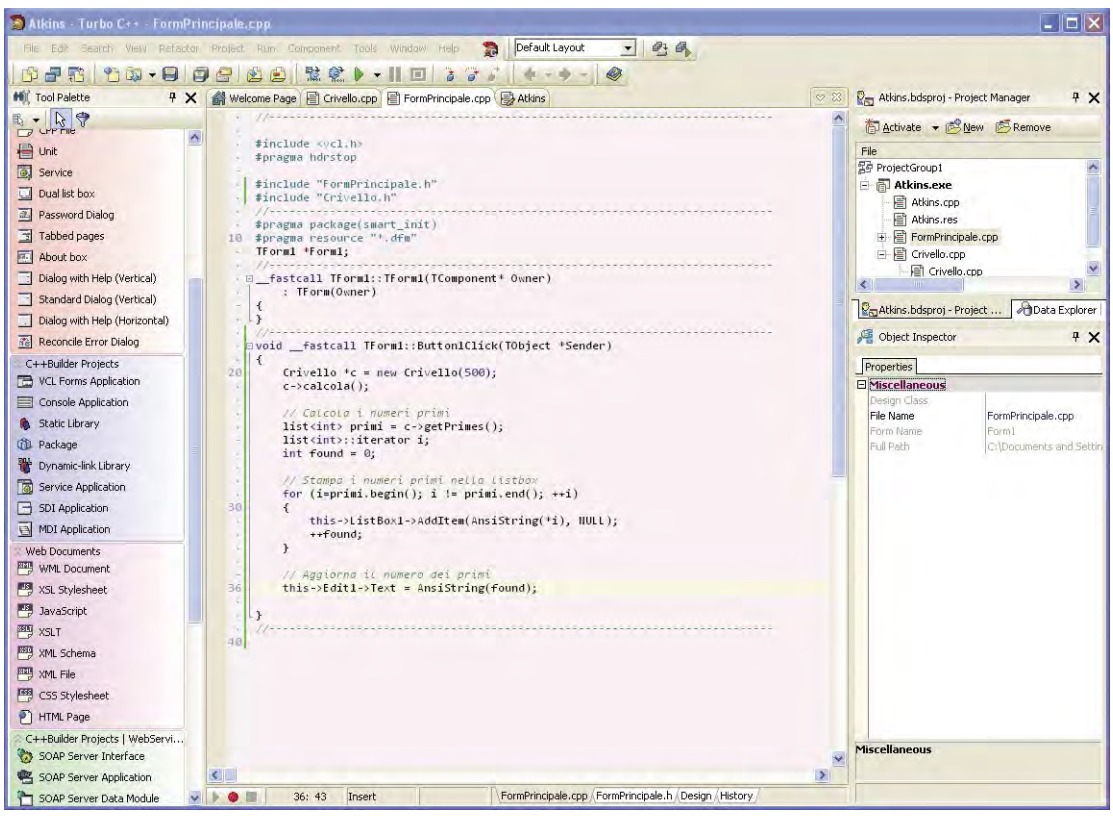


FIGURA 3 L'applicazione finita

Per finire inseriamo l'immagine di un crivello. Come "cos'è un crivello"? È un filtro, un setaccio! Cercate su Google un'immagine che vi piaccia e poi salvatela sul desktop. A questo punto, selezionando il controllo *TImage* (è sotto gli "Additional") potremo caricarla premendo il pulsante "*Load*". Ora, impostando la proprietà *Stretch* a true, l'immagine verrà ridimensionata a seconda della geometria del controllo che andremo a impostare. Anche per questi ultimi due controlli imposteremo l'*Anchor* a Top, Right per il primo e Bottom Right per il secondo, cioè quello in basso.

Hands on: associare codice agli eventi

Ora dovremo collegare l'evento che parte quando l'utente preme il pulsante "Calcola" al codice presente nella classe "Crivello" che abbiamo inserito prima.

Per conoscere proprietà e metodi dei controlli VCL appena usati non dobbiamo fare altro che utilizzare l'ottimo *help in linea* a disposizione. In particolare, noi andremo a cercare nella "Reference" la documentazione "VCL for Win32 (C++)". In questo caso scopriremo che la nostra *TListBox* ha un metodo *AddItem* che prende come parametri una stringa e un oggetto. La stringa è quanto verrà stampato e l'oggetto è l'oggetto associato, nel nostro caso possiamo lasciare *null*, perché ci interessa la sola stringa.

Facciamo bene ad abituarci ad usare *AnsiString* per le stringhe, classe analoga alla ben più famosa *CString* di MFC. Come scoprirete, è facile costruire una *AnsiString* sia a partire da array di caratteri, che da stringhe STL. Inoltre è molto facile convertire: per convertire numeri in stringhe basta chiamare il costruttore di *AnsiString* dandogli come parametro il numero stesso, viceversa, per convertire da AnsiString a numero, basta richiamare un apposito metodo statico.

Facendo doppio click sul pulsante direttamente dal designer dell'interfaccia grafica, il Turbo C++ costruirà un metodo "*ButtonClick*" e lo aggiungerà alla classe *TForm1* che è la classe principale del nostro progetto. In questo metodo inseriamo il codice che potete vedere nel **Listato 3**. Sinteticamente: istanziamo un oggetto di classe Crivello (ho utilizzato un puntatore automatico, di modo che non devo preoccuparmi di rilasciare la memoria), e ne richiamiamo i metodi *calcola()* e *primi()*. A questo punto iteriamo su *primi()* e andiamo a stampare i risultati. Niente di che, e tutto molto facile grazie anche all'ottimo supporto di *code completion* fornito dal Turbo C++.

LISTATO 3

```
/**
 * Crivello di Eratostene
 * Turbo C++ Hands On
 * Alessio Saltarin 2007
 *
 * Parte del file TForm1.cpp
 *
 */

[...]

void __fastcall TForm1::Button1Click
                              (TObject *Sender)
{
      auto_ptr<Crivello>
                        c(new Crivello(5000));
      c->calcola();

      // Calcola i numeri primi
      list<int> primi = c->primi();
      list<int>::iterator i;
      int found = 0;

      // Stampa i numeri primi nella listbox
      for (i=primi.begin(); i
                        != primi.end(); ++i)
      {
            this->ListBox1->AddItem
                  (AnsiString(*i), NULL);
            ++found;
      }

      // Aggiorna il numero dei primi
      this->Edit1->Text = AnsiString(found);

}
```

Se non ci ricordiamo i metodi di una classe basta cominciare a digitare

```
nomeoggetto->
```

perché si apra la finestrella che ci suggerisce nome dei metodi e argomenti. Nulla di sconvolgente o di mai visto, certo, ma qui a differenza di altri IDE semplicemente funziona. E non è poco, credo.

Per produrre ora una versione finale e non di debug del codice, attraverso il comando "*Build Configurations*" del menù "*Project*" selezioniamo la voce "*Release*". Premiamo "*Apply*".

Hands on: compilazione, debugging, deployment

Siamo giunti alla fine della nostra applicazione da "dieci minuti": ci basta cliccare sul pulsantino verde a forma di "Play" e vedremo il nostro codice finalmente eseguito. Giusto per dare qualche numero: nel caso di questo piccolo progetto, il compilatore si macina quasi trecentomila linee di codice e produce un eseguibile, ottimizzato e con link a librerie statiche, di 500 kilobyte. Non male, eh? Avessimo prodotto la stessa cosa in C# probabilmente avremmo ottenuto un eseguibile di pochi kilobyte... peccato, che per farlo funzionare avremmo dovuto distribuirlo insieme alla versione runtime del Framework: oltre ventisei megabyte!

> **Ad ogni colpo di mouse si va formando un codice C++ elegante e completo**

Non ho parlato di debugging perché non c'è molto da dire: *watches*, *locals*, *step in*, *step out*: c'è tutto. Con un po' di esperienza navigheremo il nostro codice via tastiera molto facilmente, ottenendo tutte le informazioni che servono: in **Figura 4**, una sessione di debugging di Eratostene.

Ultima nota: il refactoring consente la sola rinomina dei metodi e delle proprietà: poco, ma assai meglio di niente.

Come effettuare il deployment? Semplice, copiare l'eseguibile su un'altra macchina e lan-

FIGURA 4 Una sessione di debugging

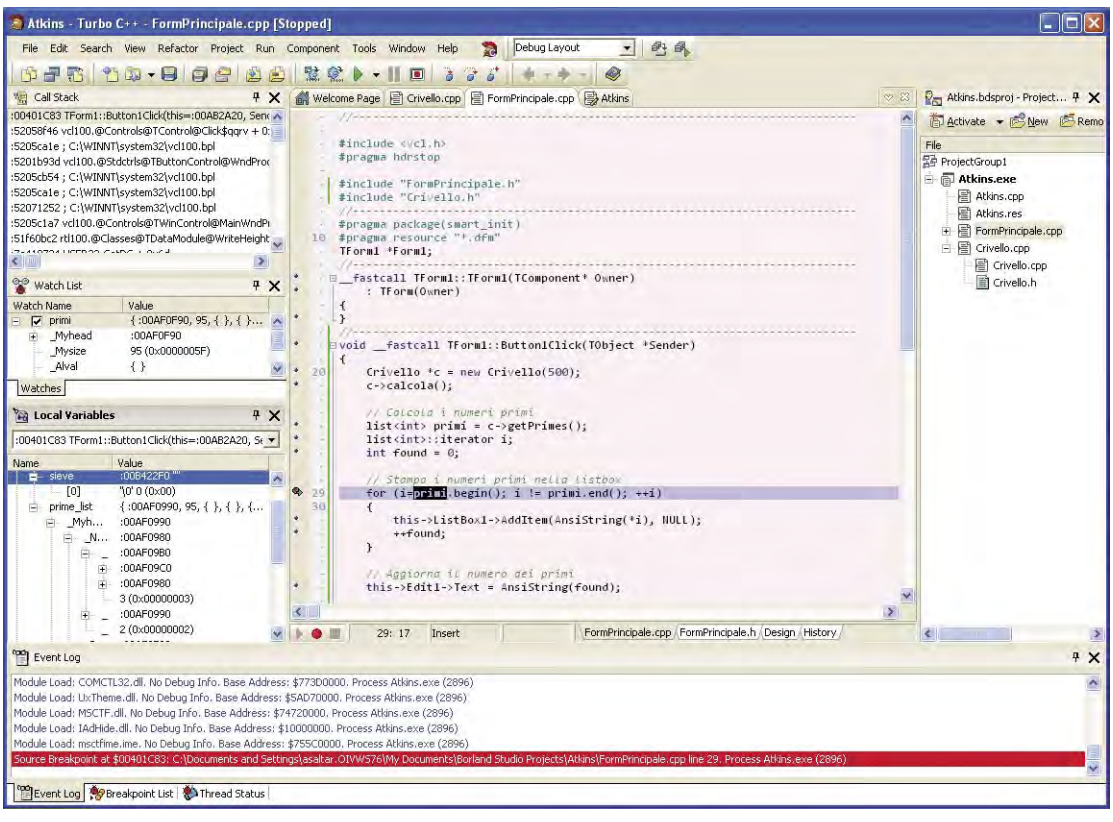
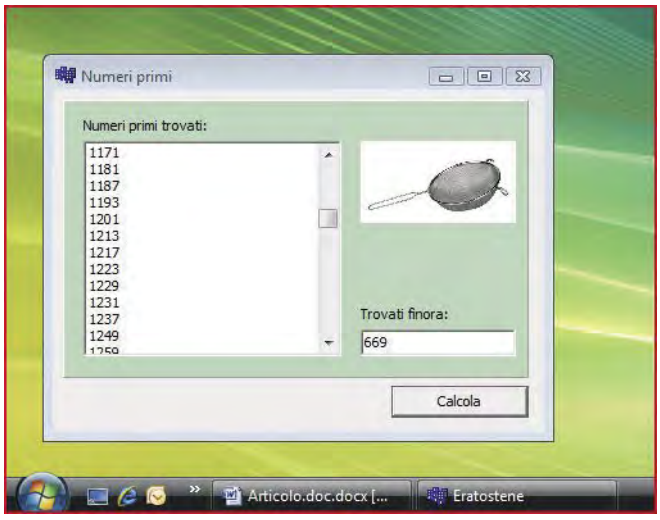


FIGURA 5 Il Crivello su Vista



ciarlo: abbiamo detto che si tratta di puro codice nativo! Sento già che vi lamentate: "*eh, ma io ho provato e mi chiede un certo file vcl100.bpl, Saltarin mi ha ingannato!*" Calma! Occorre prestare attenzione. Di default, il Turbo C++ compila senza link statici alle librerie sue (*borlandmm.bpl*) e della VCL (*vcl100.bpl*). Per forzare a farlo occorre espressamente negare l'uso di RTL dinamico (*Opzioni di progetto/Linker/Linking/Use dynamic RTL*) e negare l'uso di runtime packages (*Opzioni di progetto/Packages/Build with runtime packages*). In **Figura 5**, del resto, potete vedere il nostro programmino girare su un'installazione pulita di Windows Vista.

Conclusioni

La bella notizia è che usando sul serio e a fondo il Turbo C++ non si ha mai la sensazione di essere "abbandonati" dallo strumento, che dà invece una bella impressione di robustezza, stabilità e anche di precisione, per esempio nel disegnare le interfacce grafiche. Devo dire che gran parte del merito va alla serie di *Update* contenuti nell'ultimo *Update Rollfix* disponibile sul sito di CodeGear: la versione di cui parlavo in Aprile era invece piuttosto instabile.

L'altra bella notizia è che ci dovremo aspettare nei prossimi mesi il rilascio del *Builder 2007*, che supporterà – finalmente! – Vista (e, spero, anche le interfacce disegnate con XAML... ma non voglio esagerare).

La domanda perciò rimarrebbe quella già posta in [1]: perché installare la versione gratuita di *Turbo C++ 2006* oppure addirittura comprare *Turbo C++ Professional*? Perché, soprattutto, reinstallare tutto il .NET Framework v1.1 (*run time* e *SDK*) quando vivevamo così felici con il 2.0 e stavamo chiedendoci se passare al 3.0 subito o aspettare l'upgrade a Vista? La risposta è sorprendentemente semplice: Turbo C++ rimane l'unico IDE credibile per sviluppare applicazioni Windows per il desktop in vero C++ compilato. Cioè: naturalmente le si può sviluppare benissimo con un qualsiasi IDE o editor e appoggiarsi al Platform SDK di Microsoft, avendo a disposizione tanto – ma tanto! – tempo e buona volontà, e questa sarebbe effettivamente cosa buona e giusta volendo evitare, come dicevo in apertura, di produrre codice MSIL (tenendo ben presente però, che *Visual C++ Express* fa a pugni col *Platform SDK*, e che bisogna comunque passare alla versione Professional di Visual Studio se si vuole evitare di impazzire).

Turbo C++ è quindi la soluzione che cercavamo: in dieci minuti disegno la mia interfaccia grafica, premo il tasto e si compila, in perfetto Win32 nativo. Il prezzo è l'utilizzo delle antiquate VCL (che non hanno un look propriamente *cool*) e di un IDE che, per quanto robusto e valido, non può competere con Visual Studio che appartiene, direi, ad un'altra categoria. Ma, insomma, la versione Explorer è gratis e quella Pro costa un manciata di dollari. Direi che si può fare, no?

CODICE ALLEGATO

ftp.infomedia.it **TurboC++2**

Il progetto del Crivello di Eratostene

BIBLIOGRAFIA & RIFERIMENTI

[1] Alessio Saltarin – "Turbo C++ ancora una volta", Computer Programming n.167, Aprile 2007
[2] CodeGear from Borland - http://www.codegear.com/

Borland Turbo C# 2006

di Fabio Fabozzi

La Borland è sempre stata uno dei più grandi produttori di compilatori. Prima dell'avvento di framework di programmazione, quali Java o .NET, i compilatori Borland erano molto diffusi tra i professionisti e, altresì, godevano di ottime note di performance. Tra i compilatori più noti, nelle loro varie versioni, quelli più "anziani" tra noi possono ricordare il "Turbo Pascal", il "Turbo C" e nell'epoca delle applicazioni "Window-Based" molti ricorderanno il famosissimo "Delphi" (antagonista di qualità del Visual Basic pre-.NET), che con il passare del tempo, ha sfruttato anch'esso il .NET framework.

Ad ogni modo, oggi la realtà è ben diversa: il tempo dei compilatori stand-alone, che prendevano e rimaneggiavano dei linguaggi standard è passato.

Fabio Fabozzi ffabozzi@infomedia.it

Lavora come Sviluppatore / Architetto Software presso ICM S.p.A. sede di Roma, dove si occupa di sviluppo in .NET, Java, integrazione con SAP e sistemi IBM. È laureando in Scienze dei Processi Cognitivi, presso l'università "La Sapienza" di Roma. È autore del Libro "Intelligenza Artificiale con Visual Basic" e collabora con le riviste del Gruppo Infomedia dal 2001. Tra i suoi principali interessi: Intelligenza Artificiale, Algoritmi Genetici e Modellizzazione Neuro-Cognitiva.

Il mercato domanda (forse giustamente) da molto tempo una compatibilità multipiattaforma, che nel nostro lavoro è di fondamentale importanza. La risposta alla domanda, come molti sanno, è ormai storia nota e passata.

L'esistenza di piattaforme quali Java e .NET ha contribuito a questa sorta di sviluppo, che ha trovato la sua naturale interfaccia nei Web Service, che rappresentano un po' la moda di questi ultimi anni. Si pensi a SAP, ad esempio, che prevede una modalità d'installazione che consente di accedervi via Web Service, oltre al già celebre SAP Connector per Java e .NET.

Il fatto che si possa realizzare, in un'applicazione n-tier, la Business Logic con una tecnologia e/o linguaggio di programmazione ed implementare il Presentation Layer con un'altra, ha aperto molte prospettive per chi "vende" i prodotti informatici.

Il primo contatto

Lasciando alle spalle queste digressioni, passiamo al nostro "Turbo C#". Analizzando uno strumento di un marchio così blasonato, bisogna porre un'estrema cautela ai dettagli presenti in esso: è possibile che qualcuno possa pensare che tra RAD ufficiali (cioè quelli della Microsoft) e quelli "freeware", non ci sia bisogno di un altro

strumento di sviluppo per la piattaforma .NET. A me, personalmente, non dispiace il fatto che si possa scegliere tra una varietà di strumenti e secondo le capacità finanziarie che un'azienda può spendere su molte licenze.

L'installazione

Ho eseguito l'installazione su Windows XP Professional Service Pack 2: Processore Pentium III - 900 Mhz con 128Mb di memoria.

Ho sottostimato la configurazione della macchina, anzi, ne ho montata una vecchia appositamente per la grande occasione.

Ho omesso, appositamente, l'installazione di IIS, tra poco vedremo insieme il perchè.

Ho trovato piuttosto curioso il fatto che uno strumento di sviluppo del 2006, non sia stato dotato del .NET Framework 2.0. In effetti, è presente solo il Framework 1.1 con Service Pack1.

Anche se il Framework 2.0 è stato dotato del Service Pack1, la Borland poteva quantomeno dare la possibilità agli utenti di installare le due versioni di Framework. Ad ogni modo, c'è solo quello (cioè la versione 1.1): andiamo avanti.

Naturalmente, il database che Borland distribuisce con il Turbo C# è Interbase. Tuttavia, se non vi piace Interbase, con ADO.NET potrete collegarvi con il RDBMS che vi piace di più.

Non ho installato sulla macchina anche IIS, perchè mi aspettavo che la Borland distribuisse un qualche Web Server di debug, per testare applicazioni ASP.NET. Infatti c'è e si chiama "CassiniWebServer", un Web Server della Microsoft scritto totalmente in C#. In alcune sue tipologie di distribuzione, vedi "UltiDev Cassini", assume delle connotazioni meno spartane per la configurazione e il rilascio di applicazioni. Non si può pretendere che abbia le potenzialità di IIS, ma è comunque un ottimo strumento per lo sviluppo e il suo impiego come light web server; in sistemi operativi come "XP Home Edition", che non prevedono l'installazione di IIS, può essere un ottima alternativa.

Altra sorpresa, poco gradita da parte mia, è stato il fatto di non trovare un progetto per Windows Mobile, Pocket PC, etc. È un vero peccato, i palmari oggi giorno hanno conquistato una grossa parte dell'utenza nel campo della telefonia e della navigazione satellitare.

La procedura d'installazione è molto semplice e poco impegnativa.

Tuttavia, è osservabile sin da questa fase che all'interno di questa suite vi sono alternative interessanti, ma anche dei punti oscuri.

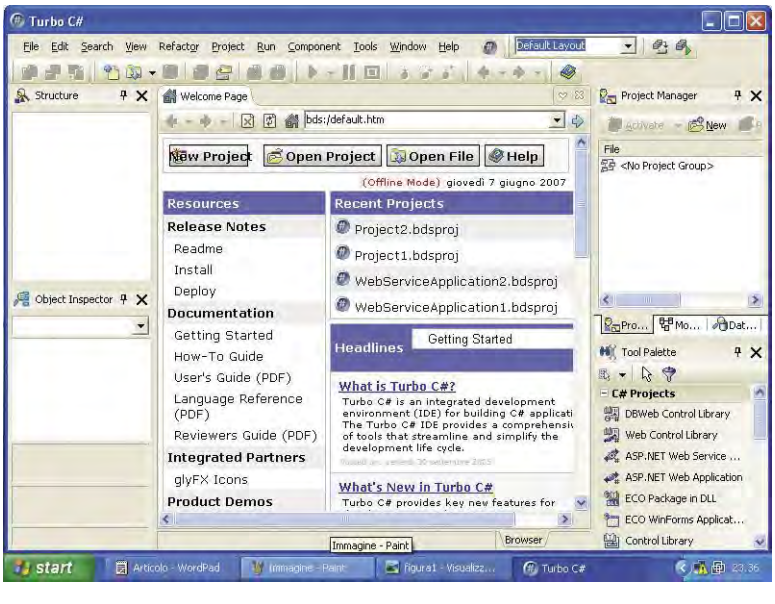
A onor del vero, qualcuno si potrebbe domandare cosa effettivamente questo strumento offra a livello di "features" che si possono trovare nelle versioni "Express" di Microsoft o in RAD come Sharp Develop. Stiamo parlando di modalità di scelta di un tipo di progetto che mediamente è standard: Console Application, Window Form, ASP.NET Web Site, ASP.NET Web Service, Class Library, etc.

Non si tratta, naturalmente, di fare "polemica" verso uno strumento che non è proprietario di un certo framework e/o ambiente di programmazione. Ma in alcuni casi è anche lecito farsi venire dubbi e domande.

All'interno del RAD

All'apertura del RAD, cioè prima di iniziare un nuovo progetto, il Turbo C# mostra una

FIGURA 1



organizzazione delle finestre non usuale in confronto a ciò che gli sviluppatori abituati con Visual Studio sono normalmente abituati a vedere (**Figura 1**).

Immagino che negli ultimi RAD (che per altro non conosco), la Borland abbia adottato questo particolare layout come una sorta di "marchio distintivo".

Ad ogni modo c'è tutto, e non solo, potete anche spostarvi le finestre per riprodurre l'ambiente di MS Visual Studio, se siete abituati all'utilizzo di quello strumento (**Figura 2**).

Per ciò che concerne le tipologie di progetto, si possono trovare quelle più comuni:

- Console Application
- Windows Forms Application
- Class Library
- Control Library
- ASP.NET Application
- ASP.NET Web Service
- Web Control

Con queste tipologie di progetto si può tranquillamente lavorare, dunque. Tuttavia, TurboC# presenta due metodologie di progetto particolari.

Si tratta di una tecnologia chiamata ECO (Enterprise Core Objects) e questa versione di Turbo C#, ne mette a disposizione i seguenti tipi:

- ECO WindowsForms Application
- ECO Package in DLL

La tecnologia ECO si può riassumere in questa sede con poche parole. Si tratta di uno strumento di modellazione della Business Logic, per le applicazioni su cui si va a lavorare. Attraverso questa tecnologia è possibile interagire con una sorta di diagrammi visuali intuitivi che definiscono la struttura delle classi e dei dati che le rappresentano, le modalità con cui viene fornita la persistenza, ossia come i dati vengono collegati ed immessi in un RDBMS.

Naturalmente supporta la modellizzazione attraverso lo standard UML.

FIGURA 2

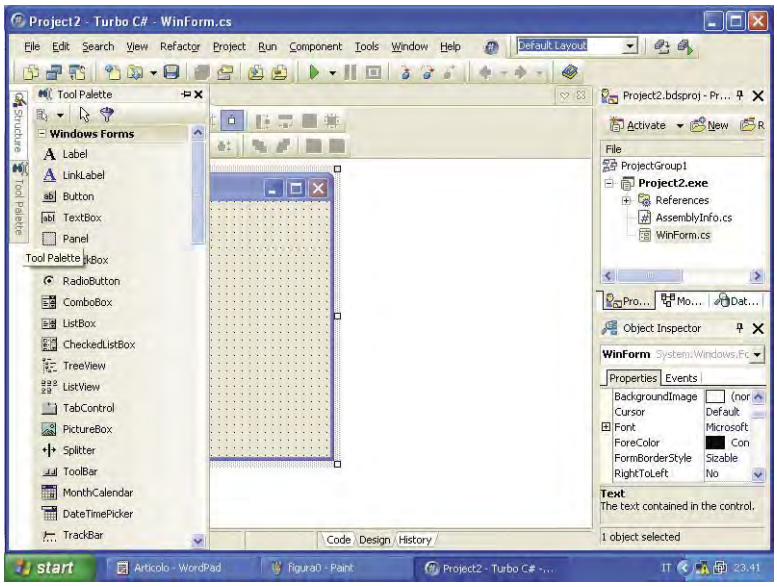
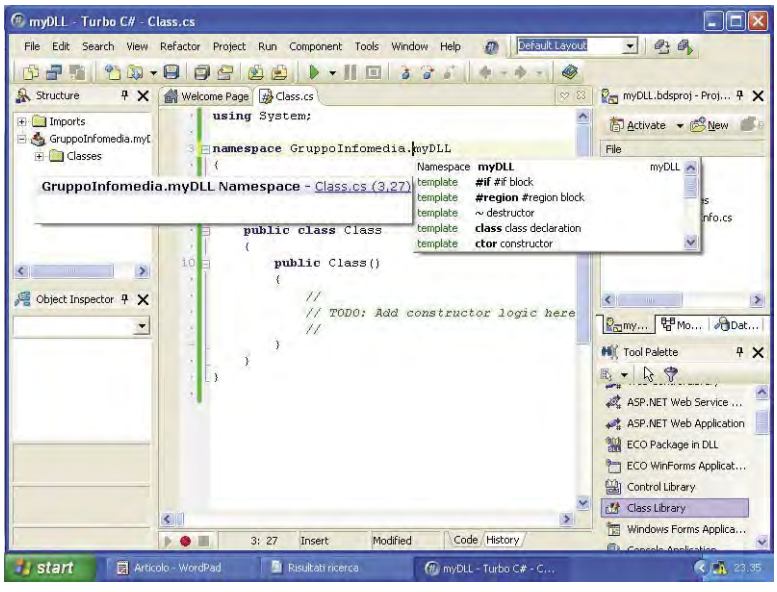


FIGURA 3



Questa tecnologia presenta una peculiarità del mondo Borland, che meriterebbe di essere trattata in altra sede.

Scrivere codice con Turbo C#

Chi conosce l'ultima versione di MS Visual Studio, ma anche strumenti che sono parallelamente concorrenti come "NetBeans" di Sun Microsystem, sa che aiutano e facilitano molto la scrittura del codice.

Il fatto che si provi, in generale, a scrivere un piccolo gruppo di progetti è indice, secondo la mia opinione, di quanta difficoltà e/o familiarità si possa trovare nell'utilizzo di un nuovo RAD.

Proviamo a mettere in piedi un piccolo progetto, per vedere quale sarebbe il grado di difficoltà che si incontra da neofita dell'ambiente e non del linguaggio C#.

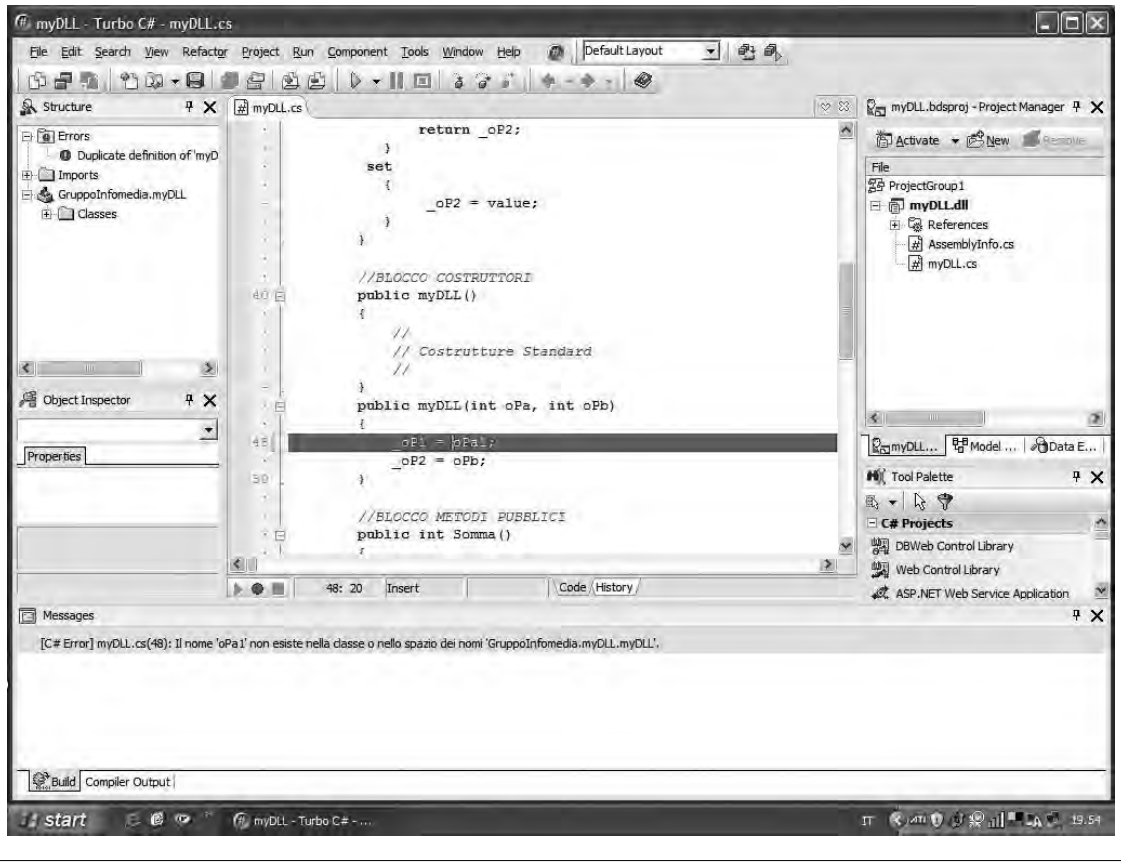
Ciò che voglio fare è molto semplice: una piccola DLL che viene richiamata da un Web Service che espone i metodi per l'accesso alla DLL ed infine, nel Presentation Layer, una applicazione Windows Forms, che consumi il Web Service attraverso una classe Proxy.

Appena ho creato il Progetto "Class Library" ed ho cambiato il Namespace, mi è subito venuto in aiuto il sistema di completamento automatico delle istruzioni: è avanzato quanto gli altri, ma ricorda molto da vicino quello di NetBeans di Sun. Le informazioni su ciò che si vuole "fare" appaiono in modo chiaro e molto esteso (**Figura 3**).

> **Prima dell'avvento di framework di programmazione i compilatori Borland erano molto diffusi tra i professionisti**

Tuttavia, durante la scrittura del codice, MS Visual Studio 2005 (dalla versione Express in su), suggerisce anche il nome della variabile che si vuole utilizzare, una "chicca" in più che non avrebbe fatto male a Turbo C#.

FIGURA 4



LISTATO 1

```
using System;

namespace GruppoInfomedia.myDLL
{
   /// <summary>
   /// Semplice classe di Operazioni tra interi
   /// </summary>
   public class myDLL
   {
      //BLOCCO ATTRIBUTI
      private int _oP1;
      private int _oP2;

      //BLOCCO PROPRIETA'
      public int Operatore1
      {
         get
            {
               return _oP1;
            }
         set
            {
               _oP1 = value;
            }
      }
      public int Operatore2
      {
         get
            {
               return _oP2;
            }
         set
            {
               _oP2 = value;
            }
      }
      //BLOCCO COSTRUTTORI
      public myDLL()
      {
               //
               // Costrutture Standard
               //
      }
      public myDLL(int oPa, int oPb)
      {
               _oP1 = oPa;
               _oP2 = oPb;
      }

      //BLOCCO METODI PUBBLICI
      public int Somma()
      {
               return _oP1 + _oP2;
      }
      public int Molt()
      {
               return _oP1 * _oP2;

      }
      public int Divisione()
      {
               return _oP1 / _oP2;

      }
      public int Sottr()
      {
               return _oP1 - _oP2;
      }

   }
}
```

Ho volutamente fare assolutamente un errore, giusto per vedere come il compilatore segnala gli errori. La segnalazione mi è sembrata abbastanza utile (**Figura 4**).

Il codice della DLL è molto semplice (**Listato 1**).

Per ciò che concerne il Web Service, alla creazione del progetto Turbo C# chiede subito all'utente se si vuole far girare il servizio sotto "CassiniWebServer" o "IIS".

Strutturalmente non è diverso da Web Services creati con altri tool.

A questo punto bisognerà cambiare il namespace e, successivamente, aggiungere il riferimento a myDLL.dll: nel progetto del Web Service, nella finestra di progettazione "File", fare click con il tasto destro su "References" e fare click su "Add Reference", la schermata che comparirà sarà del tutto uguale a quella di Visual Studio.

Per semplicità ho esposto solo un metodo nel Web Service, ho fatto girare il tutto e "CassiniWebServer" mi ha mostrato il mio Web Service senza problemi (**Listato 2**).

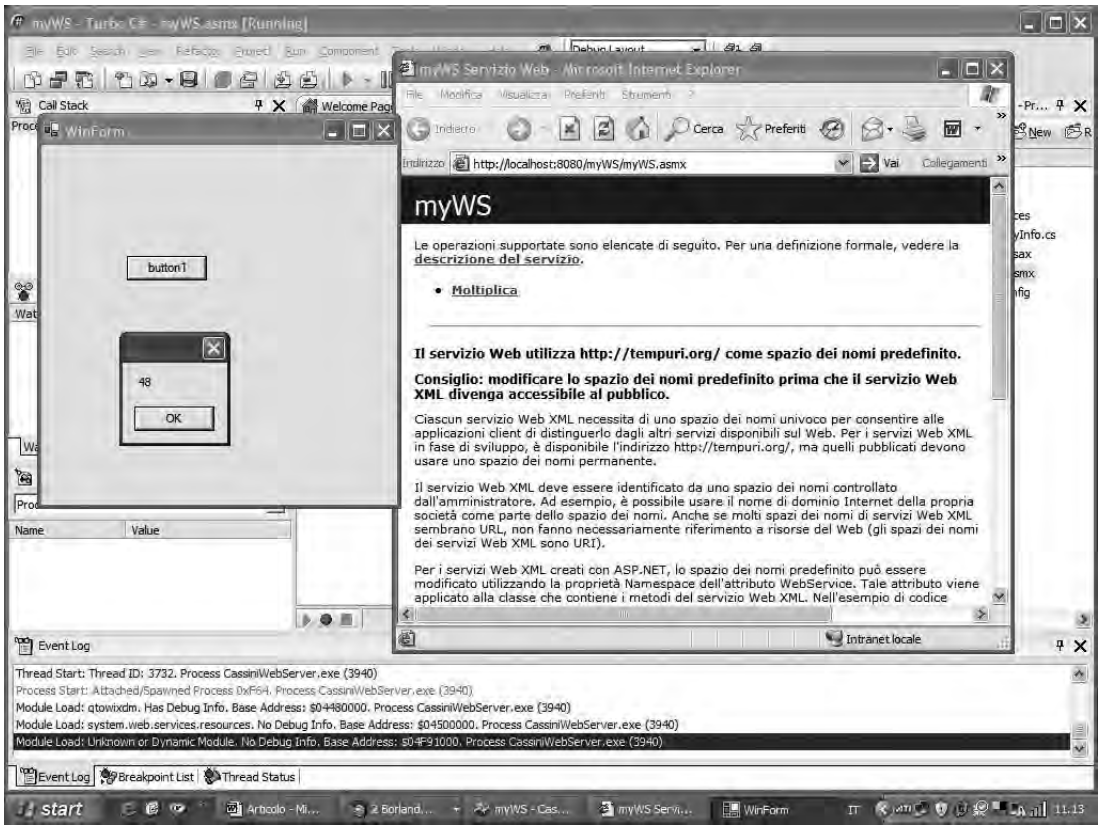
Infine ho consumato il Web Service in una applicazione Windows Forms, in modo molto semplice:

```
private void button1_Click(object sender,
                           System.EventArgs e)
   {
      myWS a = new myWS();
      MessageBox.Show( Convert.ToString
                        (a.Moltiplica(12,4)));
   }
```

LISTATO 2

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using GruppoInfomedia.myDLL;

namespace GruppoInfomedia.myWS
{
	/// <summary>
	/// Summary description for WebService1.
	/// </summary>
	public class myWS: System.Web.Services.WebService
	{
		//private GruppoInfomedia.myDLL iDLL = new GruppoInfomedia.myDLL();
		public myWS()
		{
			//
			// CODEGEN: This call is required by the ASP.NET Web Services Designer
			//
			InitializeComponent();
		}
		#region Web Form Designer generated code
		//Required by the Web Services Designer
		private IContainer components = null;
		/// <summary>
		/// Required method for Designer support - do not modify
		/// the contents of this method with the code editor.
		/// </summary>
		private void InitializeComponent()
		{
		}
		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if(disposing && components != null)
			{
				components.Dispose();
			}
			base.Dispose(disposing);
		}
		#endregion
		// Sample Web Service Methods
		// The following methods are provided to allow for testing a new web service.
		// To test, uncomment these lines, save, and build the project.
		// This code may safely be deleted.
		 [WebMethod]
		 public int Moltiplica(int intero1, int intero2)
	  {
	     GruppoInfomedia.myDLL.myDLL myObj = new GruppoInfomedia.myDLL.myDLL();
			myObj.Operatore1=intero1;
			myObj.Operatore2=intero2;
			return myObj.Molt();
	  }
	}
}
```

FIGURA 5



Il risultato finale è visibile in **Figura 5**.

Qualcuno avrà da argomentare sul fatto che i tre progetti possono essere banali. Ad ogni modo, ciò che non è banale è il fatto che per aver utilizzato la prima volta un nuovo RAD e, naturalmente, conoscendo il linguaggio e le tecnologie sottostanti al framework, sono riuscito a portare a termine il compito in pochissimo tempo: questa non è, naturalmente, una mia prerogativa, ma è ciò che sottende alla tecnologia del Turbo C#, che mi ha consentito di farlo e, sicuramente, lo consentirebbe a chiunque.

Conclusioni

Ho cercato di centrare l'articolo su quelle che sono le esigenze di una certa probabile utenza del Turbo C#: gli sviluppatori .NET che scrivono con il linguaggio C#. Da sviluppatore C# e non solo, sono rimasto colpito dai punti di luce e quelli di ombra innescati da questa suite.

Se da una parte la tecnologia ECO può essere un valido supporto alla progettazione ed andrebbe trattata con grande interesse in un articolo a parte, dall'altra le mie perplessità si sono acuite davanti ad un prodotto che è senz'altro diverso, ma altresì datato già al momento della sua uscita. Non posso considerare il mio giudizio né positivo, né tantomeno negativo. Per altre vie, come credo a qualcuno possa essere venuto il dubbio, mi chiedo se ci fosse bisogno di un prodotto che presenta "features" che sono già pro-

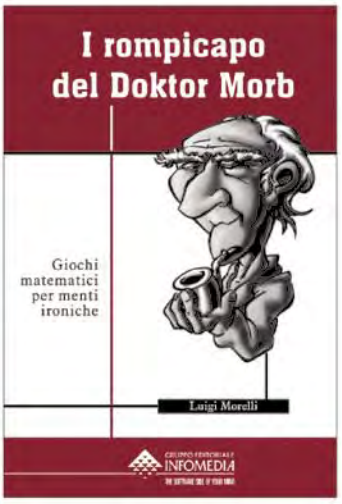
> **Il tempo dei compilatori stand-alone è passato**

prie di prodotto "free" (vedi Sharp Develop e MS Visual Studio Express Edition). Ad ogni modo, ognuno sente proprio il "tuning" dello strumento di sviluppo e quindi vi incoraggio a provarlo. Per gli amanti del mondo "Borland" provenienti da altri linguaggi della stessa casa, il suo utilizzo sarà sicuramente apprezzato, soprattutto se sentite l'esigenza di imparare un nuovo linguaggio di programmazione. Ogni feedback relativo alla vostra esperienza è come sempre molto gradito.

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

Primi passi con il mattoncino NXT e Java

Informatica e Robotica: un primo approccio a un sistema che permette di costruire robot e programmarli.

di Maria Luigia Nitti

Ho visto circa un anno e mezzo fa su Internet per la prima volta i robot Lego ed è stato… un amore a prima vista.

Non sapevo nulla di robotica, non avevo neanche idea di cosa si potesse fare con questi sistemi e in quale maniera. Ho cercato, letto, ho pensato ai miei studenti e alle ricadute positive che avrei potuto avere con loro, ne ho parlato con i miei colleghi e con il Dirigente Scolastico, ed ecco in settembre arrivare i primi kit a scuola. Ho battezzato il primo mattoncino "Cindy" in onore di mia figlia, e sono partita all'esplorazione.

Penso proprio che questi robot siano magici ma non solo per i bambini, i ragazzini e gli adolescenti, ma anche per noi, docenti o genitori o semplicemente curiosi.

Maria Luigia Nitti mlnitti@infomedia.it

È docente di Informatica presso l'ITIS Cannizzaro di Rho (Milano). Si occupa di progettazione di database e sviluppo di software per la gestione di database, collaborando con alcune università italiane; si occupa inoltre di Informatica con le nuove Tecnologie. Da un anno circa è Responsabile del Progetto di Robotica presso l'ITIS Cannizzaro. Sito della scuola: www.itiscannizzaro.it/moodle. Corso:Robotica - LegoMindstorms

Dal primo istante in cui mi sono trovata angosciata davanti una scatola di pezzi che sembrava senza vita, alla costruzione del primo semplice robot, fino ad arrivare a farlo finalmente muovere dopo una settimana di lotta con il computer, la chiavetta bluetooth, il software e il robot, sono passati alcuni mesi. Credo sia utile sapere come iniziare a lavorare senza perdere troppo tempo; il futuro è del pensiero. Cosa decideremo di fare con questo robot? I miei studenti mi hanno subito chiesto se potevamo mandarlo al bar.

Cosa è NXT

Lego MINDSTORMS è un sistema che permette di creare e programmare robot. È utilizzato nel mondo da istituti scolastici di tutti i livelli, dalla scuola primaria di primo grado fino all'Università: quindi anche da bambini di 8 anni. Permette di costruire un sistema integrato, con parti elettroniche ed elettromeccaniche guidate da un computer, combinando il tutto con mattoni Lego e mattoni della serie Technic: ingranaggi, mattoncini forati, aste, eccetera.

Una curiosità: il nome Lego è un'abbreviazione di due parole danesi "leg godt", che significano "gioca bene".

NXT è l'evoluzione del mattoncino RCX,

rilasciato nel 1998. Per chi già possiede RCX con varie attrezzature, esiste la possibilità di utilizzarle anche in connessione con NXT, con cavi speciali.

I robot possono essere equipaggiati con motori e con sensori di luce, di contatto, sonori, di prossimità ad ultrasuoni, forniti nelle confezioni standard; esistono anche altri tipi sensori acquistabili separatamente. Un robot costruito secondo le istruzioni fornite da Lego per eseguire i primi esperimenti ha l'aspetto che potete osservare in **Figura 1**.

FIGURA 1 Il robot equipaggiato con tre sensori

Per programmarli si può utilizzare un linguaggio iconico, molto intuitivo, costituito da blocchi che si accostano uno all'altro, oppure uno dei consueti linguaggi di programmazione. Esistono sulla rete comunità intere che sviluppano API (Application Programming Interface) in vari linguaggi di base, per permettere la programmazione dei robot. È quindi possibile usare i linguaggi standard, come Java, C++, Visual Basic; esistono poi molti linguaggi dedicati.

Nei vari paesi del mondo esistono scuole che insegnano a costruire e programmare gli NXT, e ogni anno si tengono gare di ogni genere, a tema. Il tracciato e lo scopo della gara vengono notificati in anticipo ai partecipanti, che possono quindi allestire e provare il loro robot.

In Italia si è svolto in giugno a Catania il torneo Minirobot della pace, esclusivamente per robot Lego. Si è svolta in Francia EUROBOT 2007 dal 16 al 20 maggio, per partecipanti fino a 30 anni, con qualsiasi tipo di robot. La gara era basata su regole di raccolta di elementi differenziati: pile, bottiglie, e altri oggetti. Il robot doveva prendere la "spazzatura" e metterla nel giusto contenitore. Le regole complete sono molto più dettagliate [3].

Caratteristiche tecniche di NXT

NXT è definito "mattone programmabile": è un vero e proprio computer integrato con display, tastierino, e porte di vario tipo. È ciò che rende vivo il robot MINDSTORMS e gli permette di compiere operazioni diverse [2].

I pulsanti di NXT sono quattro:

- *arancio*: serve per l'accensione; ha il significato del tasto *Invio*
- *due frecce grigio chiaro*: per muoversi avanti e indietro nei vari menù
- *grigio scuro*: serve per tornare indietro nei menù, oppure per cancellare

Nella **Figura 1** si possono vedere chiaramente i pulsanti frontali di NXT.

Le caratteristiche tecniche del mattoncino programmabile NXT sono invece:

- Microprocessore 32-bit classe ARM7, con 256 KB memoria flash, 64 KB RAM
- Microcontrollore AVR 8 bit 8 MHz, con 4k flash e 512 byte RAM
- Interfaccia bluetooth v2.0 (per trasferire il software o per controllare il robot da remoto, con computer, altri NXT, palmari, cellulari)
- Porta USB 2.0 (12Mbit/s)
- 4 porte di input per sensori
- 3 porte di output per motori
- Display grafico LCD bianco e nero da 100 ×64 pixel
- Speaker mono qualità audio 8 KHz
- Alimentazione con 6 batterie AA (1.5V) oppure tramite batteria ricaricabile al litio

Robot e sensori

La parola ROBOT (si pronuncia: ròbot) è stata creata da Karel Capek e dal fratello Josef (deriva dalla parola cecoslovacca "ROBOTA") nel romanzo "Rossum's Universal Robots" nel 1920. In origine la parola ROBOTA significava: lavoro obbligatorio svolto dai servi della gleba.

Un robot è una macchina di qualsiasi tipo, a volte con forma simile a quella umana, in grado di svolgere in modo indipendente alcuni compiti. Per svolgere i suoi compiti, il robot deve poter interagire con l'ambiente che lo circonda; di qui l'importanza dei sensori. Un sensore è un dispositivo che trasforma una grandezza fisica, per esempio un suono o una luce, in un segnale, per esempio elettrico. Il segnale rilevato dal sensore viene trasmesso al computer e può essere manipolato secondo la necessità.

Il sensore di contatto permette di percepire la pressione di un tasto, o la vicinanza di un oggetto; il sensore di luce permette di riconoscere la quantità di luce riflessa da un oggetto, e quindi il suo colore, oppure la luce ambientale; il sensore ad ultrasuoni permette, nel caso di NXT, di riconoscere fino a 8 oggetti a una distanza massima di 2,5 metri e quindi di rilevare ostacoli e percepire il movimento; il sensore di suono permette di riconoscere l'intensità dei suoni. Ad ognuno di questi stimoli il robot reagirà con un comportamento appropriato all'uso che se ne intende fare.

Nella **Figura 1** si possono osservare tre sensori: un sensore di luce, rivolto verso il basso accanto alla linea nera; un sensore di suono, a destra, sulla zona rossa; un sensore ad ultrasuoni, che fornisce al robot l'apparente aspetto umanoide, ed è posto nella parte più alta del robot.

Utilità: apprendimento, creatività, divertimento, socializzazione

La programmazione di un robot perché possa eseguire alcuni compiti particolari è diversa dall'usuale programmazione. Si tratta di interagire inviando comandi ad un meccanismo; a volte l'effetto dei comandi non si esaurisce con la loro esecuzione. Per esempio: inviando un comando di messa in moto del motore A, il motore è avviato e prima della fine del programma deve arrivare al robot un comando di arresto. Se la comunicazione *bluetooth* viene interrotta, il motore, che ha ricevuto il comando di andare avanti, continua a girare, e deve poi essere spento manualmente. Una considerazione interessante riguarda anche l'apprendimento della "taratura". Per certi compiti, è necessario che il sensore di luce possa riconoscere colori diversi, ad esempio bianco, rosso e nero. Il sensore deve essere appoggiato sul foglio del colore analizzato, e si devono eseguire parecchie misurazioni, per individuare un "range" di valori nel quale andrà collocato il valore della lettura. È sicuramente un modo di procedere abbastanza particolare nel mondo della programmazione, in cui di solito gli algoritmi offrono risultati certi. È molto interessante riuscire a "scoprire" cosa si può far compiere al robot, come si può montare, equipaggiare, comandare; lo spazio per la fantasia è sfrenato. È altrettanto coinvolgente e fonte di grande soddisfazione riuscire a realizzare ciò che si desidera. Allo stesso tempo, per realizzare un intento si procede nella conoscenza della pura programmazione: solo perché serve, e non perché sia fine a se stessa. Non solo per gli studenti, ma per chiunque, è molto più piacevole osservare il risultato del proprio sforzo mentale che si traduce in azioni del robot, piuttosto che programmare un algoritmo di ordinamento o altro.

Parte dell'utilità di un gruppo che lavora con un robot è data, almeno per quanto riguarda le esperienze scolastiche, dal realizzarsi di un buono spirito di collaborazione. Ogni studente, lasciato libero di scegliere, trova una sua collocazione nello spazio di lavoro, dedicandosi al compito che più gli interessa.

Cosa si trova in rete: quali linguaggi e dove

Esistono in rete molti linguaggi per programmare NXT. Sono nate comunità intere di svi-

luppatori, che lavorano a questo scopo. NXT è uscito nell'agosto 2006, e i linguaggi sono ancora in versioni sperimentali. Questo elenco non è certamente esaustivo:

- *NBC* (Next Byte Codes): una versione simile all'assembler. Esiste un ambiente integrato di sviluppo, BCC, Bricx Command Center, da installare sul PC. Tramite questo tool, si scrivono i programmi, si compilano e si inviano ad NXT, usando il cavo USB oppure la connessione bluetooth. Il programma può essere eseguito direttamente da NXT, oppure premendo un pulsante di *run* direttamente dal Bricx Center. La documentazione, in inglese, si trova sul sito [4]; la traduzione in italiano di un tutorial, che parla sia di NBC che di NXC, si trova sul sito [5].
- *NXC* (Not Exactly C): un linguaggio ad alto livello, simile al C; usa lo stesso ambiente di sviluppo BricxCC. La guida al linguaggio e il tutorial, in inglese, si trovano sul sito [4].
- *Visual Basic*: Esiste in rete un tutorial che spiega come interfacciare Visual Basic con NXT [6].
- *iCommand*: un package Java, per controllare NXT con una connessione bluetooth; usa il firmware standard Lego NXT per ricevere i comandi direttamente dal computer. Esiste una versione per Windows ed una per Linux. L'ultima versione è la 0.5 dell'ottobre 2006; si attende l'uscita della nuova versione [7].
- *Lejos NXJ*: una Java virtual machine; per utilizzarla, bisogna cambiare il firmware originale di NXT. È uscita la versione Alpha 0.2 nel marzo 2007, sia per Windows che per Linux [7].

FIGURA 2 Selezione della periferica Bluetooth

Il linguaggio scelto per le implementazioni: Java con iCommand

Java esiste in rete come linguaggio open source, scaricabile dal sito della Sun. È un linguaggio ad oggetti molto usato anche in ambito scolastico. Per usare il package iCommand, è necessario avere prima installato Java sul proprio computer.

Dal sito della Sun [8] bisogna scaricare *Java SE Development Kit (JDK)*; le prove di cui tratta questo articolo sono state fatte con la versione *JDK 5.0 update* 6, su un sistema operativo XP.

Il mio consiglio è di eseguire l'installazione offline, scaricando quindi un eseguibile che per Windows avrà un nome del tipo "jdk-1_5_0_6-windows-i586-p.exe". L'eseguibile è self-extracting; basta lanciarlo e l'installazione avviene senza problemi.

Se non vengono modificate le impostazioni all'installazione di Java, sarà creata in "C:\Programmi" una cartella del tipo *\jdk1.5.0_06*, contenente tra le altre le cartelle: *bin*, *jre*, *lib*.

Per proseguire è necessario scaricare da [7] il package iCommand; il file che si ottiene è *icommand-0.5.zip*. Tra i file estratti, interessano principalmente: la libreria *icommand.jar*, e il file *icommand.properties*.

La libreria va messa nella directory *lib* della home directory di Java. Il file *properties* servirà in seguito ad indicare la porta seriale di collegamento con la chiavetta bluetooth (vedi paragrafo successivo).

Sono necessarie quindi le librerie che gestiscono la comunicazione [8]. Scaricando, dalla sezione *download*, il file *rxtx-2.1-7-bins-r2.zip (Final)* e scompattandolo, si otterranno il file *RXTXcomm.jar* e le due librerie *rxtxSerial.dll, rxtxParallel.dll*. L'archivio "RXRXcomm.jar" deve essere copiato nella cartella: *\home_java\jre\lib\ext*.

La chiavetta bluetooth e il collegamento con NXT

A questo punto si deve installare il software necessario a gestire la chiavetta bluetooth. Non tutte le chiavette sono compatibili con

FIGURA 3 Selezione del servizio Bluetooth

NXT; esiste una sezione della documentazione Lego dedicata al problema di compatibilità [10]. La chiavetta utilizzata per queste prove è Belkin, modello F8T013 ver. 2.

Nel corso dell'installazione, vedrete comparire la caratteristica icona bluetooth sulla barra delle applicazioni. L'icona è bianca, quando la chiavetta è installata ma non connessa ad alcun dispositivo; rossa, se la chiavetta non è inserita nella posta USB, oppure verde, se la chiavetta è connessa ad un dispositivo. Installando la chiavetta, si arriva ad un punto in cui può essere effettuata la rilevazione delle periferiche **(Figura 2)**.

Accendete il vostro NXT premendo il pulsante *arancio*.

Selezionando *Cindy*, è attivato il tasto *Avanti*. NXT emette un suono e mostra la password "1234"; premuto il tasto *arancio* su NXT, per accettare la password, bisogna immettere nella videata dell'installazione il codice di sicurezza "1234" (il codice potrebbe naturalmente essere modificato), quindi premere "Abbina ora". Se compare "Errore di pairing" è indispensabile ripetere l'operazione.

Si seleziona *DevB* **(Figura 3)**, e si preme *Configura*. Compare una finestra che mostra il nome della porta seriale; questo nome è importantissimo per il funzionamento successivo del software.

A questo punto, il computer è connesso a NXT; l'icona bluetooth sulla barra delle applicazioni di Windows è diventata verde.

Trovato il nome della porta seriale, che per esempio può essere COM5 come nella **Figura 4**, si va a modificare il file "icommand.properties", utilizzando un editor di testo senza caratteri di formattazione, come per esempio il *Blocco note*.

Il file che avete scaricato contiene due righe di commento standard, quindi la definizione della porta seriale:

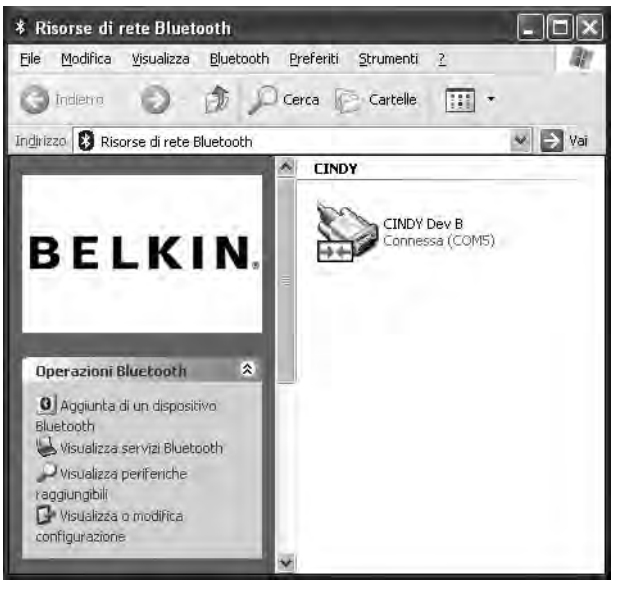
```
# Set the value of the nxtcomm
      property to the value of -
            >your<- serial port,
# e.g. COM4 (on Windows),
     dev/tty.NXT_1 (on Mac OSX),
           /dev/rfcomm0 (on Linux)
nxtcomm=COM5
```

Questo file deve essere situato nella directory home dell'utente.

Un editor per la scrittura e la prova di programmi: JCreator

JCreator è uno tra i più semplici IDE (Integrated Development Environment) per Java. Esiste in

FIGURA 4 Connessione Bluetooth effettuata



versione free [11] ed in una versione a pagamento con alcune funzioni aggiuntive, tra cui quella, molto utile, di completamento del codice. Questo strumento è molto comodo per poter utilizzare Java ad un livello di conoscenza molto basso; altri tool, come per esempio Eclipse, richiedono una discreta conoscenza di Java.

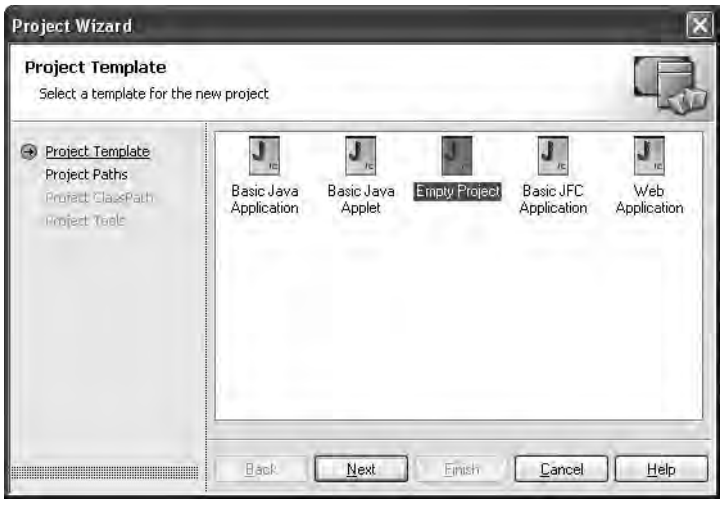
Usando la tipologia *Project*, JCreator raggruppa in una stessa cartella un insieme di file, non necessariamente collegati, che si utilizzano per lo sviluppo e la compilazione di un'applicazione, e permette di definire proprietà del progetto, tra cui librerie aggiuntive, nel nostro caso *icommand.jar*. Assieme ad un *Project*, JCreator utilizza un *Workspace*, che è un file in cui vengono salvate tutte le informazioni relative al progetto.

Si può scegliere tra due tipologie di progetto: *Basic Java Application* oppure *Empty Project* (**Figura 5**). La differenza sta nel fatto che la prima include due file di base per le applicazioni grafiche, e raggruppa in due diverse cartelle, *src* e *classes*, i file sorgenti ed i file compilati; la seconda include tutti i file in una stessa cartella.

Alla comparsa della finestra *Project ClassPath*, bisogna selezionare la linguetta *Required Libraries*, poi *New*, quindi *Add* e *Add Archive* (**Figura 6**).

Si cerca quindi il percorso della directory home di Java (C:\programmi\java\jdk1.5.0\lib) e si seleziona tra le librerie il file *icommand.jar* (**Figura 7**) quindi si preme *Apri*. Nella finestra precedente *Set Library*, si assegna il nome *ICOMMAND* (o un altro nome a piacere) alla libreria, quindi si preme *OK*. Ora non resta che selezionare con il segno di spunta, nella finestra *Required Libraries* la libreria *ICOMMAND*, che in questo modo sarà riconosciuta dal compilatore, e premere il pulsante *Finish*.

FIGURA 5 Tipologie di progetto

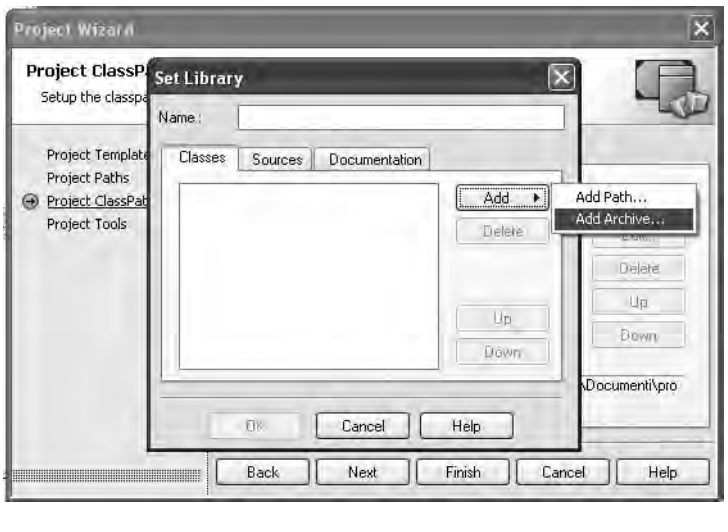


Il primo programma

Dopo tutte queste premesse, il software è installato, la chiavetta è predisposta per rilevare NXT.

Siamo pronti a provare il primo programma, che sarà semplicemente costituito dalla rilevazione e comunicazione del nome del robot, avvio dei motori A e B per 2 secondi, successivo arresto e chiusura della connessione. Chiamiamo la classe, che conterrà solo un metodo *main*, *provaMovimento.java*. All'inizio inseriamo le due istruzioni necessarie ad importare i package di *iCommand*; il dettaglio di tutti i comandi si trova nella documentazione di *iCommand* (docs\api).

FIGURA 6 Aggiunta di un archivio al progetto

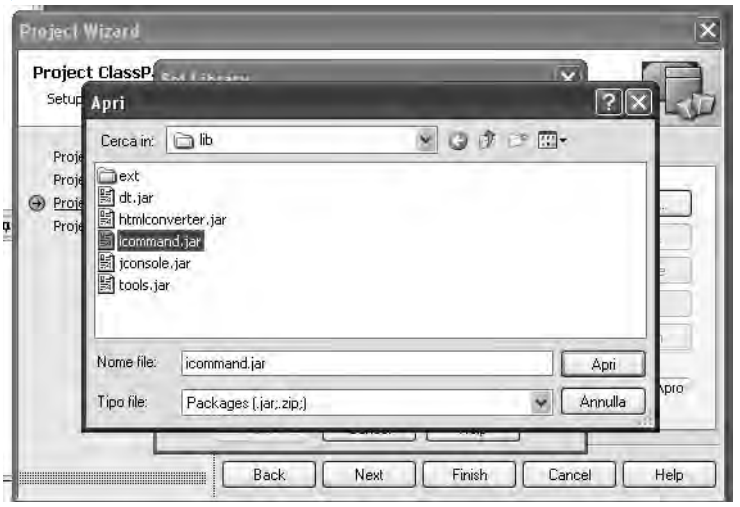


Ai due motori A e B si assegna una velocità bassa, pari a 20 (la massima è 100), con *setSpeed(20)* e si fanno partire entrambi in avanti, utilizzando il comando *forward()*. La posizione dei motori rispetto all'asse è ininfluente in questo caso, trattandosi di movimento in avanti.

La velocità di NXT non può essere modificata in modo istantaneo, come se premessimo o rilasciassimo il pedale dell'acceleratore della nostra automobile: ogni modifica del valore della velocità di un motore è attivata dalla successiva esecuzione di un comando sul motore stesso.

Dopo il ritardo di 2000 millisecondi, si arrestano i due motori e si chiude la connessione bluetooth con *NXTCommand.close()*. Secondo la documentazione, utilizzando questa istruzione non dovrebbe essere necessario a questo punto resettare NXT, invece spesso non è possibile far ripartire il robot senza averlo prima spento e poi nuovamente acceso. Il mio consiglio è di spegnerlo e riaccenderlo dopo ogni esecuzione.

FIGURA 7 Scelta dell'archivio da aggiungere al progetto




```
//provaMovimento.java
import icommand.platform.nxt.*;
import icommand.nxtcomm.*;
class provaMovimento {
  public static void main(String arg[]) {
    String nome=NXT.getBrickName();
    System.out.println("Il robot si chiama
                                  "+nome);
    Motor.A.setSpeed(20);
    Motor.B.setSpeed(20);
    Motor.A.forward();
    Motor.B.forward();
    try  {
      Thread.sleep(2000);
    }
    catch(InterruptedException e) {};
    Motor.A.stop();
    Motor.B.stop();
    NXTCommand.close();
  }
}
```

FIGURA 8 La finestra DOS durante l'esecuzione del programma

Dopo aver controllato e corretto eventuali errori di battitura segnalati dal compilatore, si ottiene da JCreator la segnalazione *Process completed*; si può quindi accendere NXT ed eseguire il programma. La connessione bluetooth viene attivata

automaticamente e l'icona sulla barra delle applicazioni diventa verde. La correttezza delle varie installazioni è provata dalla finestra DOS che deve apparire simile a quella mostrata nella **Figura 8**.

L'attivazione del robot non è istantanea; bisogna attendere qualche secondo, poi si vedrà comparire sulla finestra DOS il messaggio "Il robot si chiama Cindy".

Programmi con interfaccia grafica

Quali azioni si possono far compiere al robot? Possiamo per esempio farlo muovere in avanti, indietro, a destra, a sinistra e fermarlo utilizzando pulsanti su una finestra grafica.

All'avvio del programma, è utile leggere il nome di NXT e il voltaggio della batteria, per tenere sotto controllo la carica, e mostrare questi dati sulla finestra, in un'etichetta.

Quando il nome di NXT compare nell'etichetta, si sa che il collegamento è effettuato e che i comandi iniziano ad arrivare a NXT (**Figura 10**).

Il mattoncino Lego nasce con il nome NXT, ma sembra ovvio che ciascuno di noi desidera "chiamarlo" in qualche modo, personalizzandolo. Il comando per assegnare il nome al robot, però, in questa prima versione di iCommand, non funziona; per ora bisogna utilizzare il programma MINDSTORMS Edu NXT fornito da LEGO, e utilizzare il centro di controllo NXT Window.

Vediamo il metodo che farà girare a destra il robot, supponendo che, rispetto alla direzione di marcia in avanti, il motore A si trovi a destra e il motore B a sinistra. Questa è la configurazione più naturale, perché osservando i nomi delle porte di NXT nella **Figura 9** si vede che la A si trova a sinistra. Normalmente il mattoncino NXT viene montato in modo da avere lo schermo e i comandi nella posizione anteriore.

Per girare, si può eseguire una rotazione sul centro dell'asse delle ruote, facendo girare un motore a destra e uno a sinistra, oppure muovere un solo motore. In questo secondo caso, per far girare a destra NXT si arresta il motore A di destra, e l'altro motore deve seguire la direzione di marcia.

Si ipotizza che nella classe siano state definite due costanti intere AVANTI, INDIETRO che servono per identificare le due direzioni di marcia:

```
public void aDestra(int direzione) {
  if (direzione == AVANTI)
    Motor.B.forward();
  else
    Motor.B.backward();
  Motor.A.stop();
}
```

Un passo successivo può essere quello di far avanzare in modo autonomo il robot, configurato con il sensore di contatto nella parte anteriore: appena incontra un ostacolo, emette un suono, retrocede, gira a sinistra o a destra e ricomincia ad avanzare.

Lo stop può essere dato in svariati modi: con un pulsante, con la gestione dell'evento di chiusura della finestra, oppure si può assegnare un tempo prefissato per l'esecuzione.

Il suono viene riprodotto a partire da un file *rso* (Robot Sound file) che deve essere stato caricato in precedenza nella memoria di NXT.

Si può realizzare una finestra grafica molto semplice, con la classe *Esplora.java*, che estende la classe *JFrame* e implementa l'interfaccia *Runnable*, per poter definire un thread. La classe avrà come attributi privati un'etichetta *JLabel*, nella

FIGURA 9 Vista frontale di NXT

FIGURA 10 Finestra grafica per la segnalazione del collegamento bluetooth

LISTATO 1 Una semplice applicazione: il robot esploratore

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import java.util.Random;

import icommand.platform.nxt.*;
import icommand.nxtcomm.*;

public class Esplora extends JFrame implements
                                        Runnable {
   public static final String SOUND_FILE=
                                      "Woops.rso";
   public static final int FRAME_HEIGHT = 150;
   public static final int FRAME_WIDTH = 370;
   public static final int AVANTI = 1;
   public static final int INDIETRO = 2;
   private String name;
   private JPanel p=new JPanel();
   private JLabel n=new JLabel();
   private Touch sensTocco=new Touch(Sensor.S1);
   private int direction;
   private Random r;

   public Esplora() {
      this.setBounds(0, 0, FRAME_WIDTH,
                                    FRAME_HEIGHT);
      p.add(n);
      p.setBackground(Color.cyan);
      p.setLayout(null);
      n.setOpaque(true);
      n.setBounds(0,80,FRAME_WIDTH-8,30);
      n.setBackground(Color.white);
      n.setHorizontalAlignment(JLabel.CENTER);
      n.setBorder(BorderFactory.createLineBorder
                                   (Color.red,2));
      this.setContentPane(p);
      this.setVisible(true);
      this.addWindowListener(new ascoltatore());
   }

   public void run()  {
      int batteria,k;
      r=new Random();
      name=NXT.getBrickName();
      name.trim();
      this.setTitle(" Il robot si chiama "+name);
      batteria = Battery.getVoltageMilliVolt();
      n.setText("La mia batteria e' a "+batteria);
      Motor.A.setSpeed(20);
      Motor.B.setSpeed(20);
      while(true)  {
         direction=AVANTI;
         Motor.A.forward();
         Motor.B.forward();
         if(sensTocco.isPressed()) {
           Speaker.playSoundFile(SOUND_FILE,true);
           direction=INDIETRO;
           Motor.A.backward();
           Motor.B.backward();
           attesa(400);
           Speaker.stopSoundPlayback();
           attesa(600);
           //poi gira per 1 secondo
           k=r.nextInt(2);
           if(k == 0)
              aSinistra();
           else
              aDestra();
           attesa(1000);
         }
      }
   }

   public void aDestra() {
      if (direction == AVANTI)
         Motor.B.forward();
      else
         Motor.B.backward();
      Motor.A.stop();
   }

   public void aSinistra() {
      if (direction == AVANTI)
         Motor.A.forward();
      else
         Motor.A.backward();
      Motor.B.stop();
   }

   public void attesa(int millis) {
      try  {
         Thread.sleep(millis);
      }
      catch(InterruptedException e) {
         System.exit(-1);
      }
   }

   //gestione evento chisura finestra
   public class ascoltatore extends WindowAdapter {
      public void windowClosing(WindowEvent e) {
         Motor.A.stop();
         Motor.B.stop();
         NXTCommand.close();
         System.exit(0);
      }
   }

   public static void main(String[] args) {
      Esplora  istanza = new Esplora();
      Thread t = new Thread(istanza);
      t.start();
   }
}
```

quale verrà inserito il valore del voltaggio della batteria all'accensione del robot, e un pannello *JPanel*, che farà da contenitore (**Figura 10**).

Tra gli attributi della classe saranno definite la costante che indica il nome del file da riprodurre e la variabile di tipo *Touch* che rappresenta il sensore di tocco, collegato alla porta 1:

```
public static final String SOUND_FILE=
                                    "Woops.rso";
private Touch sensTocco=new Touch(Sensor.S1);
```

Per rilevare il valore del sensore, si utilizza il metodo *isPressed()* applicato all'oggetto *sensTocco*; il metodo restituisce un valore di tipo *boolean*. Analizzando questo valore, si controlla il comportamento del robot. Nel metodo *run()* della classe *Esplora* un ciclo infinito simulerà l'esplorazione compiuta dal robot: i motori vanno avanti finché il sensore non incontra un ostacolo. Alla pressione del sensore, il robot emette un suono per 400 millisecondi. La riproduzione del suono termina con la chiamata del metodo *Speaker.stopSoundPlayback()*; il robot torna indietro, per un tempo complessivo di 1 secondo, quindi gira a destra per 1 secondo.

```
while(true)  {
  direction=AVANTI;
  Motor.A.forward();
  Motor.B.forward();
  if(sensTocco.isPressed()) {
    Speaker.playSoundFile(SOUND_FILE,true);
    direction = INDIETRO;
    Motor.A.backward();
    Motor.B.backward();
    attesa(400);
    Speaker.stopSoundPlayback();
    attesa(600);
    //poi gira per 1 sec
    aDestra();
    attesa(1000);
    }
  }
```

Si può vedere il listato completo in **Listato 1**, con una piccolo aggiunta: il robot emette il suono e retrocede, quindi viene estratto un numero a caso per decidere se girare a destra o a sinistra.

Conclusioni

Con un computer, una chiavetta bluetooth, svariati software open source e una scatola di Lego Mindstorms possiamo iniziare a creare nuovi robot secondo la nostra fantasia e secondo le necessità. È possibile, avendo a disposizione più di un mattoncino NXT, realizzare una rete con un NXT master e tre slave.

Ho potuto vedere in rete parecchie applicazioni in Java, più che altro a livello universitario, spesso realizzate con RCX; mi pare che ancora molto ci sia da fare con il nuovo NXT. Per quanto concerne l'uso dei robot nella didattica, argomento che mi riguarda da vicino, c'è un intero mondo da esplorare, soprattutto nella scuola superiore, lavorando in modo interdisciplinare con la meccanica e l'elettronica.

Il mio progetto personale vorrebbe portare tutti gli studenti a lavorare con i robot e Java, modificando la programmazione didattica a vantaggio di un'esperienza di laboratorio vissuta in modo molto attivo.

Come scrisse Seymour Papert, inventore del LOGO, si impara molto meglio con l'esperienza pratica che con la teoria.

BIBLIOGRAFIA & RIFERIMENTI

[1] Camagni- Nikolassy – "Java – Interfacce grafiche e programmazione concorrente", Hoepli, 2005
[2] LEGO – mindstorms.lego.com/Overview/The_NXT.aspx
[3] Eurobot – www.eurobot.org/fr/index.php
[4] Bricx Command Center – bricxcc.sourceforge.net/nbc/
[5] ANSAS Piemonte – robotica.irrepiemonte.it
[6] nxtasy.org, a LEGO Mindstorms NXT community – nxtasy.org/2007/04/08/using-visual-basic-to-control-nxt/
[7] leJOS, Java for LEGO Mindstorms – lejos.sourceforge.net/
[8] Sun Microsystems – java.sun.com/javase/downloads/index_jdk5.jsp
[9] RXTX – www.rxtx.org/
[10] LEGO – mindstorms.lego.com/overview/Bluetooth.aspx
[11] JCreator – www.jcreator.com

Linux NAPI-compliant network device driver

L'articolo parla delle NAPI: l'architettura dei network device driver di Linux adatta a supportare gli adattatori di rete di nuova generazione.

di Antonino Calderone

Torniamo a discutere di network device driver (NDD) di Linux ricollegandoci a un precedente articolo ([1]) centrato sulle VPN. Alcuni degli argomenti esposti in quell'articolo, in particolare la struttura di un generico NDD, sono propedeutici alla lettura di questo, ma fruibili anche da molte altre fonti, vi segnaliamo in particolare [2].

Questa volta ci concentreremo sulle New API (*NAPI*), ovvero sul meccanismo di packet processing pensato per supportare al meglio i driver per gli adattatori di rete "veloci" - come i sempre più diffusi adattatori Gigabit-Ethernet - introdotto nella versione 2.6 del Kernel e successivamente portato "indietro" nella versione 2.4.20 (e successive).

L'architettura dei network device driver non-NAPI non è adatta a supportare dispositivi capaci di generare migliaia di interrupt al secondo, e questi d'altra parte possono rappresentare una potenziale causa di "starvation" per l'intero sistema. Alcuni dispositivi hanno funzionalità avanzate di *interrupt coalescing* (si veda a titolo di esempio [3]), ovvero la capacità di raggruppare più pacchetti per interruzione attraverso una logica per l'*interrupt mitigation*.

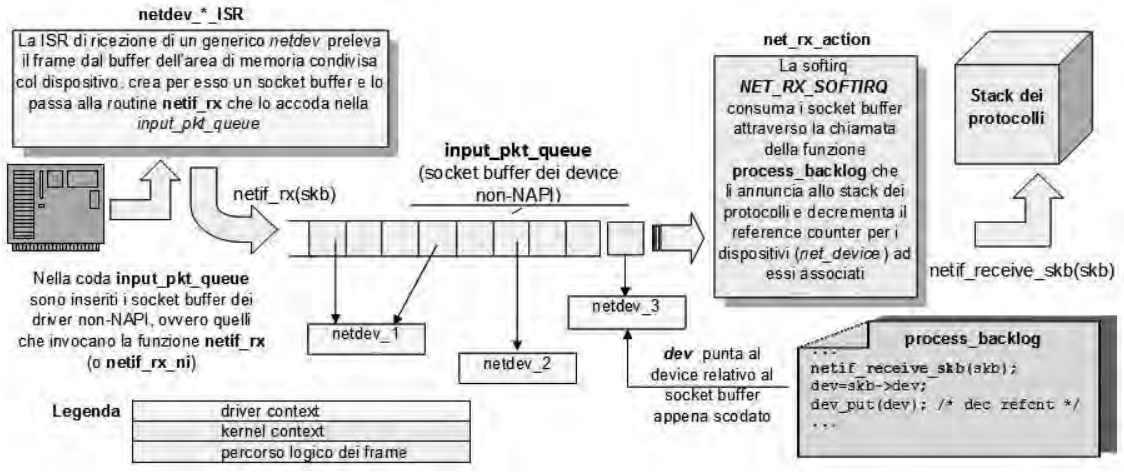
Senza l'ausilio delle NAPI, quindi senza un sostanziale supporto del Kernel, queste funzionalità dovrebbero essere interamente implementate nel driver, unite a meccanismi più o meno efficaci di prelazione (basati per esempio su timer-interrupt), e ancora su metodi di polling gestiti in contesti differenti da quello della ISR di ricezione (vale a dire kernel thread, tasklet, ecc.).

Come vedremo, il nuovo modello di NDD aggiunge al sottosistema di networking di Linux caratteristiche per supportare in modo efficace *interrupt mitigation* e *packet throttling*, e cosa importante, consente al Kernel di distribuire il "carico" mediando l'occupazione di risorse attraverso una politica round-robin tra i device.

Antonino Calderone acalderone@infomedia.it

Lavora nella R&D per Ericsson, con la qualifica di Technical Development Engineer. Progetta e sviluppa software, device driver e firmware per i sistemi per le telecomunicazioni, sistemi di controllo e automazione. Lo interessano in modo particolare gli internals dei sistemi operativi, la programmazione nello spazio di nucleo, e gli aspetti implementativi del software per il networking, in relazione soprattutto alle problematiche di interprocess communication per sistemi eterogenei, multiprogrammati e real-time.

FIGURA 1 Percorso logico dei frame ricevuti dai dispositivi pilotati da driver non-NAPI



Ricezione dei frame nel modello non-NAPI

Parleremo dei meccanismi di packet processing per la ricezione di frame interni al Kernel, senza la pretesa di essere esaustivi. Riteniamo sia necessario conoscere tali meccanismi per caratterizzare gli elementi che rendono comprensibili le differenze tra i due diversi modelli: NAPI e non-NAPI (consigliamo di consultare [4] per ulteriori approfondimenti).

Nel modello non-NAPI (schematizzato in **Figura 1**) la consegna dei frame allo stack dei protocolli avviene attraverso la funzione di Linux *netif_rx* che di norma è invocata nell'handler dell'IRQ per la ricezione di frame. Una variante di questa funzione, che può essere usata fuori dal contesto di interrupt, è la routine *netif_rx_ni*.

La funzione *netif_rx* mette nella coda di ricezione del sistema (*input_pkt_queue*) i pacchetti ricevuti dal dispositivo (imbustati in *socket buffer*), a patto che la lunghezza della coda non sia più grande di *netdev_max_backlog*: questo e altri parametri collegati sono esportati nel */proc file system* a partire dal percorso */proc/sys/net/core* e utilizzabili per il tuning.

La *input_pkt_queue* è contenuta in una struttura denominata *softnet_data* (**Listato 1**) definita nel file *netdevice.h* ([5]); della struttura *softnet_data* ne esiste un'istanza per CPU.

Quando il frame ricevuto non è scartato per il congestionamento della *input_pkt_queue*, il compito di processarlo è affidato alla softirq denominata *NET_RX_SOFTIRQ*, schedulata tramite la funzione *netif_rx_schedule*, a sua volta invocata internamente dalla routine *netif_rx*.

La softirq *NET_RX_SOFTIRQ* è implementata nella routine *net_rx_action*.

Al momento ci accontenteremo di dire che questa funzione ha il compito di passare i frame prelevati dalla *input_pkt_queue* e consegnarli alle routine dei protocolli affinché possano essere processati.

LISTATO 1 Definizione della struttura softnet_data

```
/*
 * Incoming packets are placed on per-cpu
                                queues so that
 * no locking is needed.
 */

struct softnet_data
{
   struct net_device       *output_queue;
   struct sk_buff_head     input_pkt_queue;
   struct list_head        poll_list;
   struct sk_buff          *completion_queue;

   struct net_device       backlog_dev;
#ifdef CONFIG_NET_DMA
   struct dma_chan         *net_dma;
#endif
}
```

Il nuovo modello di ricezione dei frame

Nel nuovo modello (schematizzato in **Figura 2**), nel caso di interrupt per ricezione di nuovi pacchetti, il driver informa il sottosistema di networking della disponibilità dei nuovi frame

(anziché processarli immediatamente) in modo che questi possa consumarli, attraverso un opportuno "metodo di polling", al di fuori del contesto di esecuzione dell'ISR.

I dispositivi che possono supportare le NAPI devono soddisfare dunque dei prerequisiti ([6]): il driver non può più utilizzare la coda di input dei pacchetti, pertanto il dispositivo pilotato deve essere in grado di accumulare i frame in ricezione mantenendo un buffer per quelli ricevuti anche con interrupt di ricezione disabilitati.

Questa logica riduce l'insorgenza di interruzioni a fronte di eventi generati dal dispositivo e delega al driver il compito di scartare i frame in caso di burst, evitando di saturare le code del sottosistema di networking.

Dal punto di vista dell'implementazione, le parti che differiscono sostanzialmente con il vecchio modello sono la routine di interrupt (di ricezione) e del nuovo metodo poll (attributo di *net_device*), il cui prototipo è definito nel seguente modo:

```
int (*poll)(struct net_device *dev,
                                int *budget);
```

Oltre a questo, fanno parte della struttura *net_device* due nuovi attributi (di tipo *int*), denominati *quota* e *weight*, usati per attuare il meccanismo di prelazione nel ciclo di polling (come vedremo tra poco).

La routine di interrupt nel modello NAPI delega al metodo *poll* la consegna dei frame allo stack dei protocolli. In pratica il suo lavoro si "riduce" a disabilitare l'interrupt di ricezione del dispositivo (che continuerà ad accumulare i frame entranti), effettuare le operazioni specifiche di acknowledgment dell'IRQ e infine schedulare (usando la routine *netif_rx_schedule*) la softirq *NET_RX_SOFTIRQ* associata alla funzione *net_rx_action*.

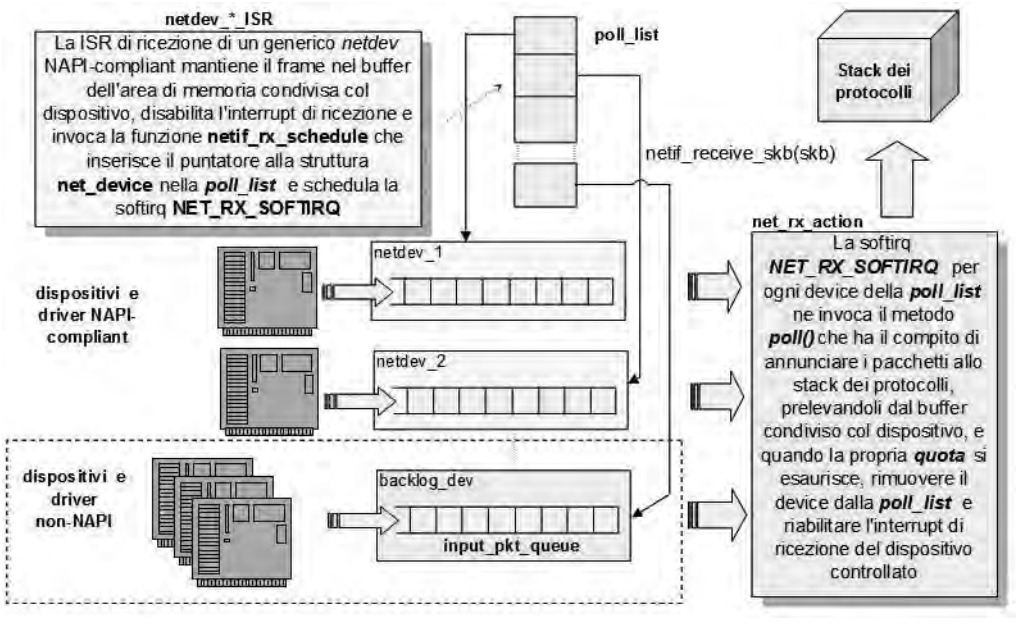
I driver in attesa di essere sottoposti a *polling* sono inseriti in una lista (*poll_list*) dalla routine *netif_rx_schedule* che prende come argomento il puntatore all'istanza di *net_device*.

La *poll_list* viene scandita durante l'esecuzione della softirq *NET_RX_SOFTIRQ* nella routine *net_rx_action*, che per ogni driver invoca il relativo metodo *poll* e questo imbusta i frame in *socket buffer* e li notifica allo stack dei protocolli.

Andando più in dettaglio la routine *net_rx_action* effettua le seguenti operazioni:

- Recupera il riferimento alla *poll_list* per il processore corrente;
- Salva nella variabile *start_time* il valore di *jiffies*;
- Imposta il parametro *budget* (passato come riferimento a *poll*) al valore iniziale di *netdev_budget* (configurabile da */proc/sys/net/core/netdev_budget*);
- Per ogni device nella *poll_list* oppure finché non si esaurisce il *budget*, oppure ancora se non è

FIGURA 2 Modello di packet processing NAPI - compliant



trascorso più di un "*jiffies*" dallo *start_time*:

- se la *quota* è positiva invoca il metodo *poll* del "device" (ovvero il suo riferimento instanza della struttura *net_device*), altrimenti somma alla quota il valore *weight* e riaccoda il device nella *poll_list*;
- se il metodo *poll* restituisce un valore non nullo, ripristina la *quota* in base all'attributo *weight* e riaccoda il device nella *poll_list*;
- se il metodo *poll* restituisce zero, assume che il device sia stato rimosso dalla lista e quindi che non sia più in *polling-state*.

Il riferimento alla variabile *budget* è passato alla routine *poll* assieme al puntatore alla struttura *net_device* del driver. La funzione *poll* deve decrementare questo valore del numero dei frame processati. I frame scodati dal buffer popolato dal dispositivo sono imbustati in *socket buffer* e consegnati allo stack dei protocolli attraverso la funzione *netif_receive_skb*.

Al criterio di prelazione basato sulla variabile *budget* è affiancato quello per device su base *quota*: il metodo *poll* deve farsi carico di verificare quanti frame possa consegnare al kernel in base alla massima *quota* assegnata al device. Quando questa si esaurisce nessun altro pacchetto dovrebbe essere inoltrato al kernel, consentendo a quest'ultimo di effettuare il polling di un altro device in attesa nella *poll_list*. Per questo la funzione *poll* deve decrementare il valore dell'attributo *quota* del numero di pacchetti processati dal driver, in modo del tutto analogo a quanto fatto per il *budget*.

Se il driver ha esaurito la propria *quota* prima di aver completato la consegna di tutti i frame accodati, allora il metodo *poll* deve cessare l'esecuzione e restituire un valore non nullo al Kernel.

Nel caso in cui tutti i pacchetti ricevuti siano stati consegnati allo stack dei protocolli, il driver deve riabilitare le interruzioni del dispositivo e arrestare il polling invocando la funzione di sistema *netif_rx_complete* (che estrae il device dalla *poll_list*), quindi deve cessare l'esecuzione e restituire zero alla funzione chiamante (che sappiamo essere *net_rx_action*).

Un altro importante attributo della struttura *net_device*, abbiamo visto, è il campo *weight*, usato per ripristinare la *quota* a ogni invocazione di *poll*.

Va da sé che il campo *weight* deve essere sempre inizializzato con un valore strettamente positivo.

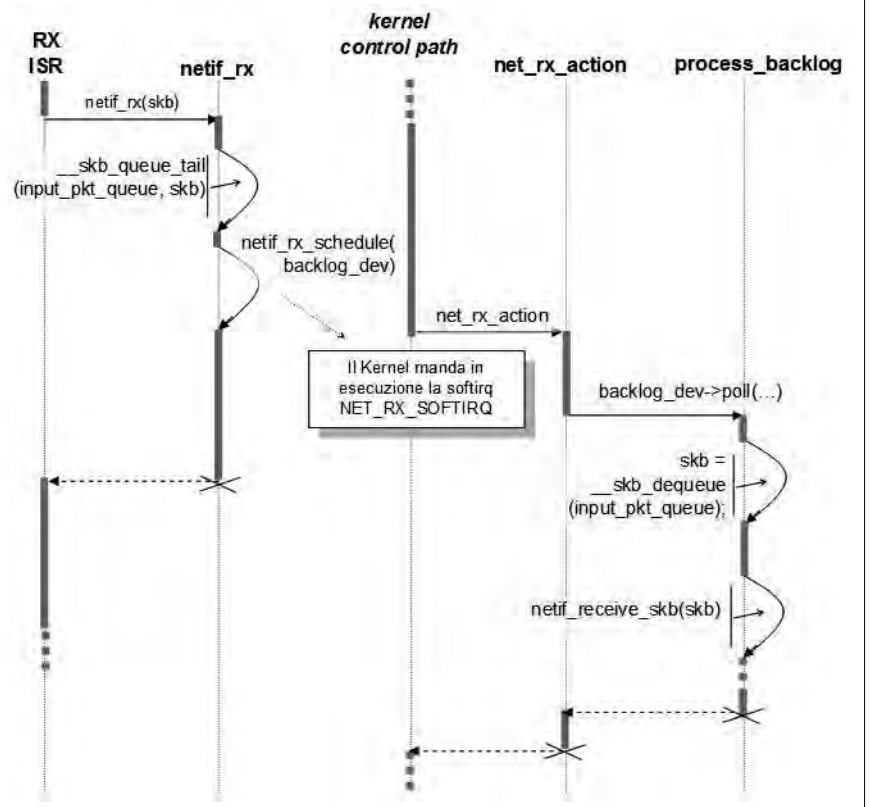
LISTATO 2 Esempio di implementazione di ISR di ricezione e del metodo poll, attributo della struttura net_device

```
static irqreturn_t sample_netdev_intr(int irq, void *dev)
{
   struct net_device *netdev = dev;
   struct nic *nic = netdev_priv(netdev);

   if (! nic->irq_pending())
      return IRQ_NONE;

   /* Ack interrupt(s) */
   nic->ack_irq();

   nic->disable_irq();

   netif_rx_schedule(netdev);

   return IRQ_HANDLED;
}

static int sample_netdev_poll(struct net_device *netdev,
                                                int *budget)
{
   struct nic *nic = netdev_priv(netdev);

   unsigned int work_to_do = min(netdev->quota, *budget);
   unsigned int work_done = 0;

   nic->announce(&work_done, work_to_do);

   /* If no Rx announce was done, exit polling state. */

   if(work_done == 0) || !netif_running (netdev)) {

      netif_rx_complete(netdev);
      nic->enable_irq();

      return 0;
   }

   *budget -= work_done;
   netdev->quota -= work_done;

   return 1;
}
```

In genere per le schede Ethernet o Fast questo valore è compreso tra 16 e 32, mentre per le Gigabit assume valori più elevati (tipicamente 64).

Guardando all'implementazione della funzione *net_rx_action* ([7]) possiamo osservare che se un driver utilizza un "extra-budget" della propria quota (fatto semanticamente ammissibile), nelle successive iterazioni deve attendere di tornare oltre una soglia positiva prima di essere nuovamente sottoposto a polling, e questo avviene tanto prima quanto più è elevato il valore attribuito al campo *weight*.

FIGURA 3 Sequence diagram relativo alle principali funzioni invocate durante il meccanismo di packet processing per driver non-NAPI



Nel **Listato 2** abbiamo riportato lo pseudo-codice relativo alla routine di interrupt di ricezione e l'implementazione del metodo *poll* di un immaginario *sample* device.

Nelle **Figure 3** e **4** sono riportati i *sequence diagram* relativi alla principali chiamate effettuate durante la fase di *packet processing* in ricezione, relativi rispettivamente al modello non-NAPI e quello NAPI-compliant.

Integrazione dei driver non-NAPI con la nuova architettura

Illustrato il meccanismo di packet processing della nuova architettura, completiamo la nostra descrizione parlando di come questo sia stato integrato con l'architettura non-NAPI.

Esiste un attributo della già discussa struttura *softnet_data* denominato *backlog_dev*. Questo attributo è istanza della struttura *net_device*.

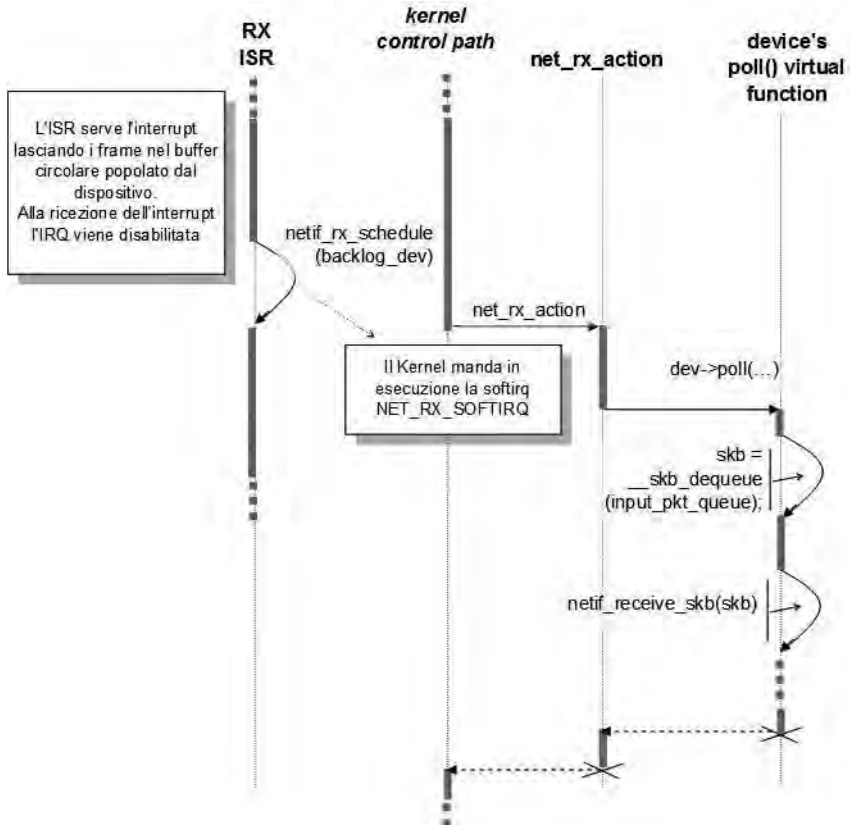
Al momento dell'inizializzazione del sottosistema di networking (in particolare nella routine *net_dev_init*) il puntatore *poll* di *backlog_dev* è inizializzato con l'indirizzo della funzione interna al kernel *process_backlog*.

La funzione *process_backlog* viene invocata nella softirq *NET_RX_SOFTIRQ* come qualunque altro metodo *poll* dei device NAPI-compliant della *poll_list*. Si ricorderà che *netif_rx* chiama la funzione *netif_rx_schedule*: il parametro passato a questa è proprio un puntatore a *backlog_dev* che è un attributo dell'istanza della struttura *softnet_data* associata alla CPU che esegue la *softirq*.

È la funzione *process_backlog* a scodare i frame della *input_pkt_queue* inseriti dalla funzione *netif_rx* e a consegnarli allo stack dei protocolli invocando la funzione *netif_receive_skb* (la stessa che è chiamata nel metodo *poll* di un generico driver NAPI-compliant).

Il numero massimo di frame che possono essere consegnati dipende anche in questo caso dal valore dell'attributo *quota* inizializzato con il parametro globale *weight_p* (modificabile da */proc file system*), oltreché dalla durata della funzione stessa che è limitata a non più di un tick del contatore *jiffies*. Esaurito questo "quanto", o esauriti i socket buffer da processare, la funzione *process_backlog* cede nuovamente il controllo alla funzione chiamante che, ormai dovrebbe essere

FIGURA 4 Sequence diagram relativo alle principali funzioni invocate durante il meccanismo di packet processing per driver NAPI-compliant



chiaro, è la funzione *net_rx_action*.

Avendo citato più volte il */proc file system* ci sembra opportuno segnalare che le impostazioni relative ai device driver (in particolare per i parametri della nuova infrastruttura NAPI) sono esportate nel *sysfs* a partire dalla directory */sys/class/net*.

Conclusioni

L'adozione delle NAPI comporta un certo numero di vantaggi.

Col nuovo modello si lascia al dispositivo il compito di gestire i buffer dei pacchetti in ingresso, delegando a questo o al software del driver il compito di scartare i frame in caso di congestione, per circoscriverne l'effetto ed evitare di interessare l'intero sistema.

Si sfruttano al meglio i meccanismi di interrupt mitigation di cui il dispositivo è capace.

Il vecchio criterio di packet processing FIFO è rimpiazzato da una politica di gestione round-robin che utilizza un meccanismo di prelazione che, affidandosi alla "cooperazione" dei driver, consente una migliore distribuzione delle risorse del sistema.

I sistemi SMP sono supportati in modo da garantire il miglior livello di concorrenza possibile.

In conclusione crediamo che l'architettura di networking di Linux basata sulle NAPI, purché supportata dal dispositivo da pilotare, sia da preferire rispetto il modello non-NAPI. Questo in effetti è l'orientamento nella comunità degli sviluppatori del "Pinguino".

BIBLIOGRAFIA & RIFERIMENTI

[1] A. Calderone - "Network Device Driver e VPN in Linux", Computer Programming N.159 - Luglio/Agosto 2006
[2] J. Corbet, G. Kroah-Hartman, A. Rubini - "Linux Device Drivers, 3rd Edition", O'Reilly, 2005
[3] http://lxr.linux.no/source/Documentation/networking/cxgb.txt
[4] C. Benvenuti - "Understanding Linux Network Internals", O'Reilly, 2006
[5] http://lxr.linux.no/source/include/linux/netdevice.h#L616
[6] http://lxr.linux.no/source/Documentation/networking/NAPI_HOWTO.txt
[7] http://lxr.linux.no/source/net/core/dev.c#L1904
[8] http://www.digifacta.com/~calderon/html/art5HTML/tvpn.gz

Programmazione concorrente in Erlang

Nell'ultimo di una serie di tre articoli su Erlang, il linguaggio sviluppato da Ericsson, affronteremo le sue potenti funzionalità di programmazione concorrente.

Terza puntata

di Federico Feroldi

Con questo terzo articolo affrontiamo quella che sicuramente può essere considerata la caratteristica principe di Erlang, ovvero la programmazione concorrente. Infatti l'unicità di questo linguaggio sta nel fatto che le primitive di gestione dei processi fanno parte integrante del linguaggio stesso.

Inizieremo con una panoramica sulla gestione dei processi e il passaggio di informazioni tra gli stessi, passeremo poi a qualche esempio includendo la gestione degli errori e termineremo con qualche accenno sulle applicazioni distribuite.

Federico Feroldi fferoldi@infomedia.it

Ha iniziato a lavorare nel 1996 in un piccolo ISP di Brescia, la sua città natale. Ha avuto modo di assistere alla nascita delle prime tecnologie web dinamiche e ha cavalcato la bolla speculativa nel 2000. Dal 2001 lavora per Yahoo!, storico attore nel mondo della Rete, dove da qualche hanno ha assunto il ruolo di Senior Architect Engineer responsabile dello sviluppo del portale My Yahoo! europeo. Appassionato da sempre di tecnologia, matematica e programmazione, recentemente è affascinato dallo sviluppo di sistemi distribuiti e dalla programmazione in linguaggi funzionali come Erlang e Haskell.

Processi in Erlang

In Erlang il termine *processo* assume un significato diverso da quello a cui siamo abituati. Normalmente quando si parla di processi ci si riferisce ai processi del sistema operativo su cui gira il nostro programma. In Erlang invece i processi vengono gestiti dalla macchina virtuale e quindi sono indipendenti dal sistema operativo sottostante. Questa astrazione porta notevoli vantaggi, per esempio:

- la creazione e la distruzione di processi è estremamente veloce;
- l'invio di messaggi tra processi è anch'esso estremamente veloce;
- essendo la gestione dei processi gestita dalla macchina virtuale, questi hanno le stesse caratteristiche su ogni sistema operativo;
- è possibile creare senza problemi un gran numero di processi, dell'ordine di decine o centinaia di migliaia;
- i processi non condividono memoria e sono completamente indipendenti tra loro, si può dire che ogni processo è una macchina virtuale a se stante;
- l'unico modo per far interagire due processi è il passaggio di messaggi.

Come già anticipato, le primitive per la gestione dei processi sono integrate nel linguaggio, per esempio per creare un nuovo processo è sufficiente usare la funzione *spawn/1* che andrà a creare una copia del processo corrente restituendo un valore di tipo *PID* che identifica il nuovo processo.

Ogni processo è identificato da un identificatore univoco detto *process identifier* (*PID*), che come vedremo viene usato per comunicare con il processo stesso.

Proviamo subito la funzione *spawn/1* tramite la console di Erlang (che può essere lanciata tramite il comand *erl*):

```
1> spawn(fun() -> io:format("hello from
                    another process~n") end).
hello from another process
<0.41.0>
```

L'unico parametro ricevuto da *spawn/1* è una funzione che verrà eseguita dal nuovo processo, mentre il valore restituito al processo corrente è il PID del nuovo processo appena creato, che ha un valore formato da tre numeri interi separati da un punto e racchiusi tra un simbolo di minore e un simbolo di maggiore.

Dal risultato che vediamo in console potrebbe sembrare che la stampa del messaggio e quella del PID siano avvenute in sequenza, in realtà esse sono avvenute in parallelo e sono state concatenate a causa dell'unica console disponibile.

Un semplice server

Come già detto, oltre a creare nuovi processi è anche possibile inviare e ricevere messaggi tra i vari processi. Per inviare un messaggio ad un processo è sufficiente utilizzare l'operatore "!" mentre per ricevere un messaggio viene utilizzata la primitiva *receive*.

Vediamo ora un esempio un po' più complesso del precedente per capire come integrare le funzionalità appena viste.

Aprite il vostro editor preferito e create il file *conc1.erl*, il nostro primo programma concorrente sarà un semplice server per fare somme e sottrazioni:

```
-module(conc1).
-export([run/0]).

loop() ->
  receive
    {'+', A, B} ->
      io:format("~p + ~p = ~p~n",
                                [A, B, A + B]),
      loop();
    {'-', A, B} ->
      io:format("~p - ~p = ~p~n",
                                [A, B, A - B]),
      loop();
    _Other ->
      io:format("Operazione sconosciuta~n"),
      loop()
  end.

run() ->
  spawn(fun loop/0).
```

Il cuore del nostro server sarà la funzione *loop/0* che rimane in attesa di una richiesta di operazione sotto forma di tupla. Il primo elemento della tupla identificherà l'operazione da compiere sugli addendi e potrà essere l'atomo '+' o l'atomo '-', mentre i due elementi successivi identificheranno gli addendi su cui effettuare l'operazione.

La sintassi della primitiva *receive* utilizza il pattern matching per elaborare i messaggi in arrivo. Nel momento in cui receive viene chiamata, vengono esaminati tutti i messaggi arrivati e non processati. Ogni messaggio viene comparato in sequenza con ogni clausola presente. Non appena viene trovata una clausola corrispondente al messaggio esaminato, questa viene eseguita. Se non vengono trovate clausole valide, il messaggio viene lasciato nella coda di ricezione e il messaggio successivo viene esaminato. Quando tutti i messaggi sono stati esaminati, la funzione si blocca e attende che un nuovo messaggio venga ricevuto.

Quindi, nel nostro server, nel caso la tupla che descrive l'operazione non venga riconosciuta come valida, l'ultimo match avrà successo (ricordiamoci che l'assegnamento di variabile è un match sempre vero se la variabile è priva di valore) e verrà quindi stampato il messaggio di errore.

Infine, la funzione *run/0* ci permetterà di lanciare un nuovo processo server e, restituendone il PID, ci permetterà di inviare le operazioni al server stesso.

Vediamo ora come utilizzare il server dalla console:

```
2> c(conc1).
3> Pid = conc1:run().
<0.93.0>
4> Pid ! {'+', 1, 2}.
1 + 2 = 3
{'+',1,2}
5> Pid ! {'-', 35, 56}.
35 - 56 = -21
{'-',35,56}
```

Come si può notare, per ogni operazione inviata viene stampato il risultato della stessa dal server ma viene anche stampata la tupla che abbiamo inviato al server stesso. Questo accade perché l'operazione "*Pid ! M*" ha per definizione *M* come valore, il quale viene quindi stampato dalla console come valore restituito dall'operatore di invio messaggio.

Architettura Client/Server

Come abbiamo visto, con poche linee di codice è possibile creare un server che esegue semplici operazioni e comunicare con esso. Come però alcuni di voi avranno notato, il server stampa le risposte alle nostre domande sulla console quando in realtà, seguendo l'architettura client-server, dovrebbe restituirle al client che le ha richieste.

La risposta è estremamente semplice; sarà infatti sufficiente inviare al server, oltre ai dati sull'operazione da eseguire, anche il PID del processo client, in modo che il server possa rispondere direttamente allo stesso.

Per implementare questa funzionalità andremo a modificare il codice del nostro server come segue:

```
-module(conc2).
-export([run/0, calc/2]).

calc(Pid, Op) ->
  Pid ! {self(), Op},
  receive
    Answer ->
      Answer
  end.

loop() ->
  receive
    {From, {'+', A, B}} ->
      From ! A + B,
      loop();
    {From, {'-', A, B}} ->
      From ! A - B,
      loop();
    {From, Other} ->
      From ! {errore, Other},
      loop()
  end.

run() ->
  spawn(fun loop/0).
```

Per prima cosa abbiamo introdotto la funzione *calc/2* che riceve come parametri il PID del processo server e la tupla che descrive l'operazione. Questa funzione andrà ad inviare un messaggio al server composto dal proprio PID (ottenuto grazie alla funzione *self/0*) e rimarrà in attesa della risposta dal server stesso. Una volta ricevuta la risposta, questa verrà restituita dalla funzione.

Andiamo a verificare il suo funzionamento:

```
6> c(conc2).
{ok,conc2}
7> Pid2 = conc2:run().
<0.110.0>
8> conc2:calc(Pid2, {'+', 10, 25}).
35
9> conc2:calc(Pid2, {'-', 73, 12}).
61
10> conc2:calc(Pid2, {'*', 3, 2}).
{errore,{'*',3,2}}
```

Resta infine un'ultima modifica da fare al nostro server. Il problema è insito nella funzione *calc/2* appena creata, infatti dopo aver inviato il messaggio di richiesta al server, resta in attesa di un messaggio di risposta. Cosa accade se mentre la funzione è in attesa, inviamo un messaggio qualsiasi al PID del suo processo? Succede che la *receive* accetta il messaggio e lo interpreterà come risposta alla richiesta inviata al server. Per ovviare a questo inconveniente è necessario inserire il PID del server nei messaggi di risposta, in modo che la funzione chiamante possa filtrare soltanto le risposte del server.

Andremo quindi a modificare la *receive* della funzione *calc/2* in modo da ricevere solo messaggi contenenti il PID del server:

```
calc(Pid, Op) ->
  Pid ! {self(), Op},
  receive
    {Pid, Answer} ->
```

```
    Answer
  end.
```

Ricordiamo inoltre che contrariamente alla variabile *Other* presente nella clausola receive della funzione *loop/0*, la variabile *Pid* in questo caso ha già assunto un valore e non può essere riassegnata, quindi la clausola causerà un match soltanto se il messaggio contiene il valore assunto precedentemente da *Pid*.

Per quanto riguarda la funzione *loop/0*, questa sarà modificata per inviare il proprio PID come parte del messaggio di risposta:

```
loop() ->
  receive
    {From, {'+', A, B}} ->
      From ! {self(), A + B},
      loop();
    {From, {'-', A, B}} ->
      From ! {self(), A - B},
      loop();
    {From, Other} ->
      From ! {self(), {errore, Other}},
      loop()
  end.
```

Infine copiamo il nuovo codice nel file *conc3.erl* e verifichiamo il suo funzionamento in console:

```
11> c(conc3).
{ok,conc3}
12> Pid = conc3:run().
<0.46.0>
13> conc3:calc(Pid, {'+', 5, 6}).
11
```

Funziona come previsto.

La gestione dei timeout

Come detto precedentemente, la primitiva *receive* vista finora attende un messaggio in arrivo bloccando l'esecuzione del programma. Effettivamente la funzione può attendere un tempo infinito, poiché può accadere per molti motivi che il messaggio non arrivi mai.

Per ovviare a questo problema esiste una variante della primitiva *receive* che permette di specificare un tempo di timeout dopo il quale la funzione ripristina il flusso di esecuzione del programma eseguendo se presente del codice per gestire l'evento.

La sintassi è la seguente:

```
receive
  Clausola1 ->
    Espressioni1;
  Clausola2 ->
    Espressioni2;
  ...
after TempoTimeout ->
  EspressioniTimeout
end
```

Se non viene ricevuto un messaggio corrispondente alle clausole specificate entro *TempoTimeout* millisecondi dall'entrata nella *receive*, vengono eseguite le espressioni *EspressioniTimeout* dopodiché il flusso di esecuzione si blocca in attesa di nuovi messaggi.

Per un valore di *TempoTimeout* intero e positivo la primitiva *receive* attende quindi il tempo di timeout per poi proseguire l'esecuzione. In questo modo è possibile implementare una funzione di *sleep/1*:

```
sleep(T) ->
  receive
  after T ->
    true
  end.
```

Quando invece *TempoTimeout* assume un valore nullo, le *EspressioniTimeout* vengono eseguite immediatamente dopo aver tentato il match dei messaggi in coda con le clausole presenti nella *receive*. In questo modo è possibile implementare una funzione che svuoti il buffer in ricezione:

```
flush_buffer() ->
  receive
    _Any ->
      flush_buffer()
  after 0 ->
    true
  end.
```

Inoltre un timeout nullo può essere utilizzato per implementare una ricezione prioritaria dei messaggi:

```
priority_receive() ->
  receive
```

```
    {alarm, X} ->
      {alarm, X}
  after 0 ->
    receive
      Any ->
        Any
    end
  end.
```

In questo modo la prima *receive* ha la possibilità di controllare tutti i messaggi presenti nella coda di ricezione. Se uno dei messaggi presente in coda corrisponde alla clausola "*{alarm, X}*" questo verrà restituito immediatamente. Se nessuno di questi messaggi corrisponde alla clausola, allora viene eseguita la seconda *receive* che potrà quindi processare i messaggi in coda, attendendo un nuovo messaggio se necessario.

Infine, se il *TempoTimeout* assume un valore corrispondente all'atomo *infinity* si otterrà lo stesso comportamento dovuto all'assenza della clausola di timeout. Ovvero la funzione *receive* attenderà un tempo infinito fino all'arrivo di un nuovo messaggio. Questa possibilità viene utilizzata in quei casi in cui il valore di timeout viene passato alla clausola after attraverso una variabile, in modo da poter variare il comportamento della primitiva *receive* a tempo di esecuzione.

Processi registrati

Come abbiamo visto, per comunicare con un processo è necessario conoscere il suo *process identifier* (PID), senza il quale sembrerebbe impossibile identificare il processo destinatario dei messaggi. In realtà Erlang ci mette a disposizione un secondo modo per comunicare con un processo, permette infatti di dare un nome al processo e di utilizzare successivamente questo nome in luogo del PID.

Questa possibilità risulta essere molto conveniente ed utile in quei casi in cui non è sempre possibile conoscere il PID del processo con cui vogliamo comunicare, per esempio se a creare il processo è stato un terzo processo che non ha provveduto ad inviarci il valore del PID corrispondente. Non ha provveduto o non ha voluto intenzionalmente, infatti dando un nome al processo si perde quel grado di sicurezza per cui soltanto i processi terzi che conoscono il PID di un processo possono comunicare con esso.

Per registrare un processo utilizziamo la funzione register/2 che ci permette di associare un atom ad un PID:

```
14> register(calc_server, conc3:run()).
true
```

Per ottenere nuovamente il PID del processo conoscendone il nome useremo la funzione *whereis/1*:

```
15> whereis(calc_server).
<0.49.0>
```

Se non esiste un processo associato al nome richiesto, la funzione *whereis/1* restituirà l'atomo *undefined*:

```
16> whereis(another_server).
undefined
```

Per eliminare l'associazione fra il nome ed il PID del processo viene usata la funzione *unregister/1*:

```
17> unregister(calc_server).
true
18> whereis(calc_server).
undefined
```

Infine, per avere una lista dei processi attualmente registrati viene utilizzata la funzione *registered/0*:

```
19>  registered().
[rex, kernel_sup, global_name_server, ... ]
```

Conclusioni

In questa breve serie di articoli abbiamo avuto modo di conoscere Erlang, un linguaggio ancora sconosciuto alla maggior parte dei programmatori ma che in un futuro di processori multi-core e di applicazioni distribuite saprà sicuramente trovare il suo spazio e sono convinto che molti inizieranno ad utilizzarlo. Spero di essere riuscito a far scoccare nei lettori quella scintilla che ha fatto di Erlang uno dei miei linguaggi preferiti (al pari di Ruby), e spero inoltre che chiunque abbia trovato interesse nelle caratteristiche certo non semplici di questo linguaggio, continui ad approfondirlo attraverso le molte informazioni presenti in rete.

Per concludere vi voglio lasciare con una piccola perla apparsa in rete [1] nel Giugno 2006 ad opera di Joe Armstrong, il progettista di Erlang.

Si tratta di una versione concorrente della funzione di *map/2* che applica una funzione *F* ad ogni elemento di una lista *L*, restituendo una lista composta dai risultati:

```
pmap(F, L) ->
  S = self(),
  Pids = map(fun(I) ->
    spawn(fun() -> do_f(S, F, I) end)
  end, L),
  gather(Pids).
gather([H|T]) ->
  receive
    {H, Ret} ->
      [Ret | gather(T)]
  end;
gather([]) ->
  [].
do_f(Parent, F, I) ->
  Parent ! {self(), (catch F(I))}.
```

BIBLIOGRAFIA & RIFERIMENTI

[1] http://www.erlang.org/ml-archive/erlang-questions/200606/msg00187.html
[2] Joe Armstrong – "Concurrent Programming in Erlang", Prentice Hall
[3] Joe Armstrong – "Programming Erlang", The Pragmatic Programmers, 2007

Java Skill

Descrizione delle competenze e dei principali profili professionali in funzione dei layer architetturali costituenti un sistema basato su tecnologia Java.

Terza puntata

di Giampaolo Marucci

In questa ultima puntata, affrontiamo finalmente il discorso delle figure professionali coinvolte nello sviluppo Java.

Figure Professionali

Le skill che descrivono le competenze o conoscenze delle varie figure professionali, sono descritte principalmente tramite acronimi, i cui significati basilari sono riportati nel paragrafo "Acronimi e Glossario dei termini" (pubblicato insieme alla prima puntata della serie) e le cui applicabilità sono state descritte nel paragrafi precedenti.

Nella descrizione delle figure professionali, il diverso livello di competenze richiesto alla figura, è individuato dalle seguenti frasi (che iniziano la descrizione di uno skill):

- *Capacità d'uso*: La figura professionale sa utilizzare (scrivere, leggere) la tecnologia o linguaggio o paradigma che è richiesto.
- *Capacità di lettura*: La figura professionale sa leggere un documento o un disegno o un artefatto scritto tramite una determinata tecnologia, linguaggio, ecc.
- *Conoscenza*: La figura professionale conosce anche solo superficialmente la tecnologia, il linguaggio la specifica o l'artefatto che gli è chiesto di utilizzare.

Giampaolo Marucci gmarucci@infomedia.it

Ha un'esperienza ventennale nello sviluppo, disegno e gestione di sistemi informatici. Laureato in Scienze dell'Informazione è attualmente Project Manager (certificato dal "Project Management Institute") e Software Architect per Eustema. È specializzato in sistemi Content-Management, Business-Process-Management, Skill-Management, Risk-Management, Portals su Web, JEE e SOA. Ha disegnato e coordinato la realizzazione di diversi prodotti sw di mercato. Gestisce progetti di medie dimensioni per la pubblica amministrazione e le enterprise.

Sviluppatore Enterprise (Enterprise Developer)

Si occupa di sviluppare con le opportune tecnologie Java e le necessarie competenze non Java, ciò che comunemente è noto come "Business Layer" o "Middleware" o "Service Layer" di una applicazione Java. Prende in input il disegno della soluzione realizzato dalla figura "Designer" ed i requisiti funzionali e non funzionali realizzati dalla figura "Analyst". Produce in output il codice sorgente Java conforme ai requisiti dati. Si possono identificare le seguenti competenze tecniche di base:

1. Capacità d'uso di linguaggi di programmazione o framework di sviluppo lato server: Java, EJB, JNDI, JMS, JCA, RMI, IIOP, Servlet, Hibernate, Spring, XML, XSD, WSDL, SOAP
2. Capacità d'uso dei seguenti paradigmi di sviluppo: OOP, Distribuited Programming, Idempotenza (ambienti cluster), Event driven (Exception Handling), Multi-Tasking
3. Capacità d'uso di linguaggi di sviluppo basi dati relazionali: SQL (platform independant), Oracle PL-SQL

4. Conoscenza delle principali specifiche SUN J2EE, in particolare: EJB, JNDI, JDBC, JAR, EAR
5. Capacità d'uso di ambienti di sviluppo: Eclipse + JST, Borland jbuilder, IBM RAD (Rational Application Developer), IBM WSAD (Websphere Application Developer), Axis
6. Conoscenza di Design Pattern: Conoscenza approfondita di tutti i design pattern riportati nel "GOF Book" e conoscenza del modello MVC
7. Capacità d'uso di ambiente di deployement per il test delle applicazioni: Tomcat/Jboss
8. Conoscenza di ambienti di deployement di produzione: Tomcat/Jboss, IBM Websphere, Bea Weblogic, OC4J
9. Capacità di lettura dei seguenti schemi UML: tutti i diagrammi tipo di UML 1.x o UML 2.0
10. Conoscenza anche marginale di uno o più delle seguenti tecnologie o linguaggi innovativi (propri delle architetture SOA): BPEL, WS-*, SOAP
11. Capacità d'uso di software di strumenti di configurazione del software: MS-VSS, IBM Rational Clearcase.

Sviluppatore Web (Web Developer)

Si occupa di sviluppare con le opportune tecnologie Java e le necessarie tecnologie non Java, ciò che comunemente è noto come "Presentation Layer" o "Web User Interface" di una applicazione web basata su Java. Prende in input il disegno della soluzione realizzato dalla figura "Designer" ed i requisiti funzionali e non funzionali realizzati dalla figura "Analyst". Produce in output il codice sorgente conferme ai requisiti dati. Si possono identificare le seguenti competenze tecniche di base:

1. Capacità d'uso dei linguaggi di programmazione o framework di sviluppo Web: JSP, Servlet, JSF, JSTL, Javabean, Struts (alternativo a JSF), XML, XSTL, XSD, WSDL, SOAP, XHTML, HTML, CSS, SQL
2. Capacità d'uso di linguaggi o tecnologie di programmazione client per applicazioni client/server (non tipicamente di tipo Web): Applet, Swing, AWT
3. Conoscenza delle principali specifiche SUN J2EE, in particolare: JSP, JNDI, JDBC, WAR
4. Capacità d'uso di ambienti di sviluppo: Eclipse + WTP, Borland JBuilder, MS FrontPage, CSS editor, Macromedia Dreamweaver, IBM RAD (Rational Application Developer), IBM WSAD (Websphere Application Developer)
5. Conoscenza del Design Patterns MVC
6. Capacità d'uso di ambienti di deployement per il test delle applicazioni: Apache/Tomcat
7. Conoscenza ambienti di deployement di produzione: IBM Websphere, Bea Weblogic, Tomcat/Jboss
8. Capacità di lettura dei seguenti diagrammi UML: Use Case Diagram, Sequence Diagram, Activity Diagram, Deployement Diagram, Package Diagram
9. Conoscenza di normative e standard del mondo W3C, in particolare si evidenzia: WCAG per l'accessibilità delle diverse abilità umane all'uso della applicazione web
10. Conoscenza di tecnologie estremamente innovative ed ancora "instabili" (o comunque poco utilizzate), quali: AJAX
11. Capacità d'uso di software di strumenti di configurazione del software: MS-VSS, IBM Rational Clearcase

Analista-Sviluppatore (Designer)

Si occupa di disegnare e documentare con le opportune metodologie e strumenti, ciò che deve essere sviluppato da uno o più sviluppatori (Web Developer o Enterprise Developer). Deve disegnare e documentare in modo comprensibile al gruppo di sviluppo (NON è necessario che documenti il disegno in modo comprensibile al cliente). Garantisce la conformità del disegno ai requisiti dei vari Stakeholder del progetto (vedi figura professionale "Analyst"). Prende in input i requisiti funzionali e non funzionali prodotti dalla figura "Analyst" e produce in output il disegno della soluzione richiesta. È in grado di sviluppare da sé uno o più parti del disegno prodotto. Si possono identificare le seguenti competenze tecniche di base:

1. Capacità d'uso di una o più skill derivanti dalle conoscenze descritte per le figure: Web Developer , Enterprise Developer
2. Capacità d'uso dei seguenti schemi UML: Class Diagram, Sequence Diagram, Activity Diagram, Object Diagram, State Chart Diagram, Package Diagram, Component Diagram
3. Capacità di lettura dei seguenti schemi UML: tutti i diagrammi tipo di UML 1.x o UML 2.0
4. Capacità d'uso dei seguenti Design Pattern:

tutti i Design Pattern riportati nel "GOF Book" e conoscenza del modello MVC
5. Capacità d'uso dei seguenti paradigmi di disegno e analisi: OOD, OOA
6. Conoscenza dei seguenti paradigmi di sviluppo: OOP, Distribuited Programming, Idempotenza (ambienti cluster), Event driven (Exception Handling), Multi-Tasking
7. Conoscenza delle principali specifiche SUN J2EE, in particolare: EJB, JSP, JNDI, JDBC, JAR, EAR, WAR
8. Capacità d'uso dei seguenti ambienti di disegno: IBM Rational Rose, Borland Togheter, Argo UML, MS-Visio (per la sola parte UML Design)

Per lo sviluppo di progetti Java sono applicabili diversi Management. Da parte del gruppo di lavoro, è necessario avere la "conoscenza" di almeno uno di tali processi

9. Conoscenza dei principali framework di sviluppo e librerie di facilitazione (API) Java.
10. Conoscenza di ambienti di deployement per sviluppo e produzione: Tomcat/Jboss, Bea Weblogic, IBM Websphere.
11. Capacità d'uso di strumenti di configurazione del software: MS-VSS, IBM Rational Clearcase

Analista (Analyst)

Si occupa di raccogliere, strutturare, documentare e controllare, con le opportune metodologie e strumenti, i requisiti funzionali e non funzionali provenienti dai vari Stakeholder di un progetto. Deve documentare i requisiti in modo comprensibile al gruppo di progetto (in particolare ai Designer) ed al cliente. Garantisce la completa aderenza dei requisiti strutturati (analizzati) alle richieste dei vari Stakeholder del progetto. Prende in input le necessità del richiedente (che può essere caratterizzato da uno o più Stakeholder) e produce in output i "Requisiti della soluzione". È in grado di disegnare da sé una o più parti del prodotto software da realizzare in conformità ai requisiti. Si possono identificare le seguenti competenze tecniche di base:

1. Capacità di analisi delle necessità
2. Conoscenza del modello (ciclo di vita del software) RUP
3. Capacità d'uso delle seguenti metodologie (strumenti) di gestione dei requisiti: Strutturazione, Dettaglio, Identificazione, Tracciamento, Versionamento
4. Capacità d'uso degli strumenti logici che consentono di stabilire la fattibilità di un dato requisito o insieme di requisiti in riferimento alle tecnologie Java che lo/li implementeranno
5. Capacità d'uso dei quattro principali tipi di requisito riconosciuti dal RUP, ovvero: STRQ, SR, TPRT, TERT
6. Capacità d'uso dei seguenti diagrammi UML: Use Case Diagram, Sequence Diagram, Deployement Diagram
7. Capacità di mediazione ed influenza tra le varie richieste provenienti dai diversi Stakeholder di progetto
8. Capacità d'uso degli strumenti di Office Automation più diffusi
9. Capacità d'uso dei seguenti strumenti di gestione dei requisiti. IBM Rational Requisite-Pro, IBM Rational SODA, IBM Rational Clearquest.
10. Capacità d'uso di una o più skill derivanti dalle conoscenze descritte per la figura: Designer

Sistemista J2EE (J2EE System Architect)

Si occupa della realizzazione della fase di "Deployement" di una applicazione J2EE. Installa, configura ed esegue le procedure di "running" dell'applicazione Java nel contesto del J2EE Application Server "target". È responsabile della configurazione dei package costituenti l'architettura software del sistema. Documenta le procedure necessarie per la corretta installazione e configurazione, start-up e shut-down degli ambienti di sviluppo, collaudo ed esercizio. Prende in input il disegno prodotto dalla figura "Designer", i requisiti NON funzionali prodotti dalla figura "Analyst" ed i componenti "eseguibili" prodotti dal gruppo di sviluppo (figure "Web Developer" ed "Enterprise Developer") e produce in output la documentazione di deploye-

ment, l'esecuzione delle fasi di deployement, e la configurazione degli ambienti di sviluppo, esecuzione e collaudo. È in grado di eseguire il controllo ("Monitoring") di una applicazione in fase di esecuzione, e l'eventuale e successivo "Tuning". Si possono identificare le seguenti competenze tecniche di base:

1. Capacità d'uso in termini di installazione, configurazione e running dei seguenti J2EE Application Server: Tomcat/Jboss, Bea Weblogic, IBM WebSphere, OC4J
2. Capacità d'uso, in termini di interfacciamento ai vari Application Server, dei seguenti sistemi operativi: MS-Win 2000 Server, MS-Win 2003 Server, Unix, Linux (diverse distribuzioni)
3. Capacità d'uso dei seguenti meccanismi propri di un J2EE Application Server: Configurazione dei componenti di gestione delle transazioni, Configurazione dei componenti di gestione della Sicurezza applicativa, Configurazione dei componenti di interfacciamento alla base dati esterna, Configurazione delle funzionalità di "clustering" e load balancing
4. Capacità d'uso dei tool (strumenti) software in dotazione ai diversi J2EE Application Server e che hanno lo scopo di facilitare le operazioni di deployement di cui al punto precedente e di eseguire le operazioni di "Monitoring" e "Tuning"
5. Capacità di lettura dei seguenti diagrammi UML: Deployement Diagram, Component Diagram, Package Diagram
6. Conoscenza delle principali specifiche SUN J2EE, in particolare: EJB, JSP, JNDI, JDBC, WAR, JAR, EAR
7. Capacità di ottimizzazione delle risorse HW e dei sistemi software di base in funzione dell'uso delle stesse da parte del J2EE Application Server su cui è "deployata" l'applicazione Java

Project Manager

Si occupa delle fasi di inizializzazione, pianificazione, esecuzione, monitoraggio e controllo del progetto atto a produrre il prodotto software basato sull'architettura J2EE. È garante verso il cliente ed i diversi Stakeholder di progetto, della buona riuscita del progetto stesso, in termini di: Tempi e schedulazione, Costi e budget, Requisiti, Qualità, Comunicazione tra gli Stakeholder, Integrazione del gruppo di progetto nell'ambito del contesto aziendale, Approvvigionamenti, ecc… Ha competenze delle tecnologie e dell' architettura J2EE tali da poter stabilire: fattibilità dei requisiti provenienti dai vari Stakeholder, eseguibilità e controllo dell'andamento del progetto, rilascio, controllo qualità e chiusura di tutti gli artefatti prodotti durante l'esecuzione del progetto. Si possono identificare le seguenti competenze tecniche di base:

1. Capacità d'uso degli strumenti di pianificazione e controllo standard di mercato: Gantt, CPM, Pert, Earned Value
2. Capacità d'uso degli strumenti di comunicazione e relazionamento con gli Stakeholder, in particolare con il committente: Conduzione riunioni, verbalizzazioni, comunicazione, mediazione, ecc.
3. Capacità di redigere documentazione sullo stato di avanzamento del lavoro al fine di fornire dati utili, ai diversi Stakeholder del progetto, sull'andamento dello stesso, in termini di: Performance di produttività, Andamento Costi, Scostamento Tempi, Previsioni a finire, Lesson Learned
4. Capacità di lettura, al fine di eseguire le necessarie fasi di validazione e controllo del progetto, dei seguenti artefatti: STRQ, SR, TPRT, TERT
5. Capacità di lettura, al fine di eseguire le necessarie fasi di validazione e controllo del progetto, dei seguenti diagrammi UML: Tutti i tipi di UML Diagram
6. Capacità di lettura, al fine di eseguire le necessarie fasi di validazione e controllo del progetto, dei seguenti linguaggi: Java, XML, UML
7. Conoscenza della principali specifiche SUN J2EE, al fine di guidare la fase iniziale di disegno architetturale e controllare l'esecuzione del progetto: JSP, EJB, JNDI, JDBC, WAR, JAR, EAR
8. Conoscenza dei principali J2EE Application Server di mercato in termini di: differenze funzionali, differenze architetturali, differenze prestazionali al fine di guidare la fase iniziale di disegno architetturale della soluzione in funzione degli Application Server e dei sistemi di base scelti
9. Conoscenza delle principali tecniche di misurazione della qualità del software prodotto nel contesto del OOP, OOA, OOD.

Java e gli algoritmi di ordinamento

In questa ultima puntata presentiamo quello che può essere considerato il miglior algoritmo di ordinamento, il Quicksort.

Quarta puntata

di Fabrizio Ciacchi

Nelle scorse puntate abbiamo parlato dei vari algoritmi di ordinamento. Abbiamo presentato un algoritmo "base" – l'Insertion Sort – un algoritmo con metodologia ricorsiva – il Merge Sort – e quello che utilizza una struttura dati ausiliaria – l'Heap Sort.

Abbiamo visto che è importante sia progettare un algoritmo più raffinato (e più intelligente), nonché utilizzare un'adeguata struttura dati per poter risparmiare la memoria utilizzata dall'algoritmo.

Il Quicksort è un algoritmo di tipo ricorsivo che non utilizza una struttura dati particolare. Il suo vantaggio è di essere uno degli algoritmi più efficienti e meno dispendiosi di memoria. Il Quicksort, per quanto efficiente, è tuttavia un algoritmo estremamente "fragile", in quanto errori nella sua implementazione possono portare ad un drastico calo delle prestazioni.

Fabrizio Ciacchi fabrizio@ciacchi.it

È uno sviluppatore e consulente informatico. Tra le sue maggiori aree di interesse ci sono GNU/Linux, il linguaggio di programmazione Java e lo sviluppo di siti web in PHP. Nel tempo libero scrive articoli su GNU/Linux e legge libri di Asimov.
Il suo sito è http://fabrizio.ciacchi.it.

La teoria del Quicksort

Il Quicksort è un algoritmo di ordinamento ideato da Charles Antony Richard Hoare nel 1960 ed è uno di quelli che, in genere, ha le prestazioni migliori. Utilizza la stessa tecnica del "Divide et Impera" del Merge Sort, richiede pochissime risorse e opera direttamente sui dati da ordinare (algoritmo *in place*). Purtroppo non è un algoritmo stabile, cioè non preserva eventuali dati già ordinati (contrariamente all'Insertion Sort e al Merge Sort) e in alcuni casi può avere un'elevata complessità.

L'idea alla base del Quicksort è abbastanza semplice, analogamente al Merge Sort: ad ogni stadio si effettua un ordinamento parziale e poi si procede all'ordinamento sui vari sottoinsiemi. Nel caso particolare si cerca un elemento come perno (o pivot), che è generalmente un elemento di valore intermedio tra il minimo ed il massimo da ordinare, e poi si spostano tutti gli elementi più piccoli di quello alla sua sinistra e più grandi alla sua destra. In questo modo si avranno due insiemi non ordinati, ma con elementi minori e maggiori del perno (che quindi si trova nella sua corretta posizione).

Si applica, quindi, il Quicksort ai due sottoinsiemi (stabilendo nuovi perni) fino a quando l'insieme degli elementi non è correttamente ordinato.

Uno dei principali vantaggi del Quicksort è che risulta essere estremamente semplice sia da capire che da implementare. Non a caso nel tempo sono state cercate diverse ottimizzazioni all'algoritmo che, tuttavia, è praticamente rimasto immutato proprio per la sua efficienza originaria.

Volendo il Quicksort può essere "convertito" in un algoritmo completamente iterativo o misto ricorsivo-iterativo, tuttavia ciò rappresenta soprattutto un buon esercizio di stile, anche se apporta dei vantaggi in termini di risorse occupate.

Implementazione in Java

Scrivere l'algoritmo Quicksort in Java non presenta grosse difficoltà. Si crea, infatti, la classe principale che estende la classe SortAlgorithm (l'algoritmo di ordinamento generico):

```
public class
QuickSortAlgorithm
extends SortAlgorithm
{
```

Il Quicksort prende in input l'array di interi da ordinare, il limite inferiore ed il limite superiore. La prima chiamata viene fatta su tutti gli elementi (lo0 = 0 e hi0 = a.length – 1) dell'array (**Figura 1**).

```
void QuickSort(int
a[], int lo0, int hi0)
throws Exception {
```

Si assegna, quindi, il valore dei parametri alle variabili locali temporanee e si dichiara la variabile *mid*, ovvero il perno.

```
int lo = lo0;
int hi = hi0;
int mid;
```

Viene poi richiamato il metodo di pausa per ridisegnare l'applet

```
pause(lo, hi);
```

per poi procedere con l'esecuzione dell'algoritmo, la cui condizione base è che il margine superiore sia maggiore di quello inferiore.

```
if ( hi0 > lo0) {
```

Si assegna, quindi, un valore "intermedio" al perno *mid* (**Figura 2**)

```
mid = a[ ( lo0 + hi0 ) / 2 ];
```

e si effettua un ciclo fino a quando gli indici superiore ed inferiore non si sovrappongono, ovvero non si è fatto il ciclo su tutti gli elementi a sinistra e a destra del perno.

```
while( lo <= hi ) {
```

FIGURA 1 L'array non ordinato su cui viene applicato il Quicksort.

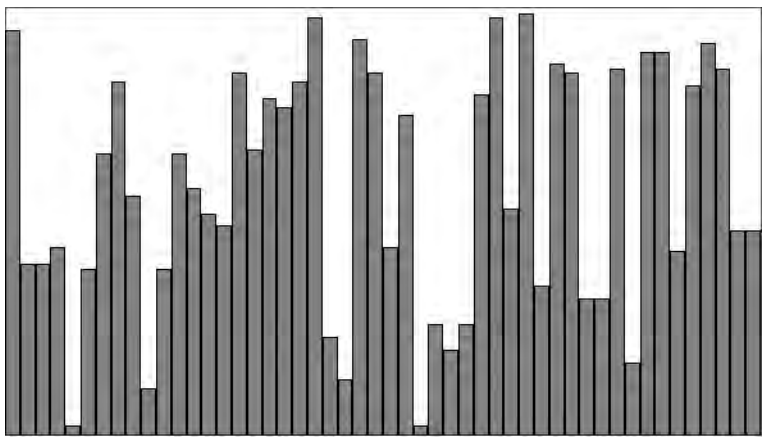
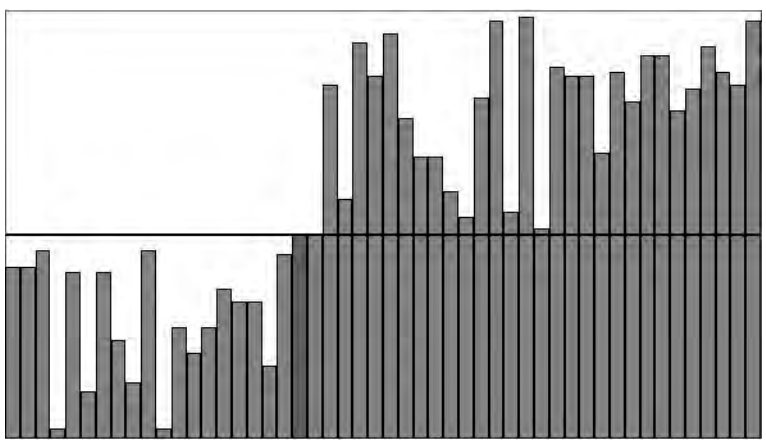


FIGURA 2 Scelto il pivot, si procede con l'ordinamento degli elementi.



Si cerca, poi, il primo elemento che è maggiore od uguale al perno mid partendo dall'indice sinistro,

```
while( ( lo < hi0 ) && ( a[lo] < mid ) )
++lo;
```

e si ripete l'analoga operazione, cercando un elemento minore od uguale al perno, partendo dall'indice destro

```
while( ( hi > lo0 ) && ( a[hi] > mid ) ) --
hi;
```

Se gli indici non si sono sovrapposti, si scambiano i valori degli indici

```
if( lo <= hi ) {
 swap(a, lo, hi);
 pause();
```

Si spostano ulteriormente gli indici e poi si chiude sia l'if che il ciclo while, che si ripete fino a quando la condizione precedente non è verificata.

```
++lo;
--hi;
} }
```

Se l'indice destro originario non ha raggiunto il lato sinistro dell'array, si esegue il Quicksort sulla partizione sinistra

```
if( lo0 < hi ) QuickSort(a, lo0, hi );
```

Analogamente se l'indice sinistro originario non ha raggiunto il lato destroy dell'array, si esegue il Quicksort sulla partizione destra

```
if( lo < hi0 ) QuickSort(a, lo, hi0 );
```

Infine si chiude l'if ed il metodo (**Figura 3**).

```
} }
```

Il metodo di swap è abbastanza semplice e scambia i valori dell'array dei due indici passati.

```
private void swap(int a[], int i, int j) {
 int T;
 T = a[i];
 a[i] = a[j];
 a[j] = T; }
```

Il metodo sort, il costruttore della classe, chiama Quicksort, come già detto, sull'array di interi a, con indici da 0 alla lunghezza (meno uno) dell'array (**Figura 4**).

```
public void sort(int a[]) throws Exception {
 QuickSort(a, 0, a.length - 1); } }
```

L'applet Java

L'applet si compone di tre elementi, di cui due fissi, il file **SortItem.java**, adibito alla creazione dell'applet ed il file **SortAlgorithm.java** che opera sugli oggetti di tipo SortItem per ordinarli; il terzo file può invece cambiare, poiché definisce l'algoritmo specifico da usare per ordinare i numeri (la classe java dell'algoritmo di ordinamento, infatti, generalmente estende la classe SortAlgorithm), ed il suo nome viene passato come parametro quando si richiama l'applet.

In questo modo è abbastanza facile implementare diversi algoritmi di ordinamento con nomi diversi, e poi darli in pasto al codice dell'applet passandogli il nome dell'algoritmo.

FIGURA 3 La chiamata ricorsiva del Quicksort sui sottoinsiemi.

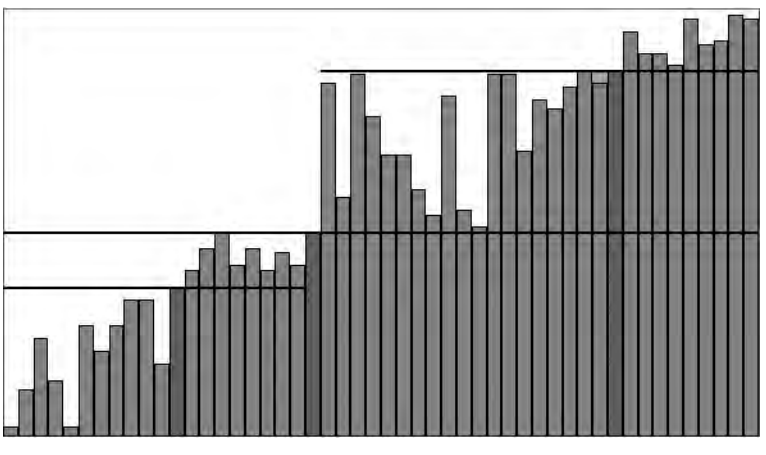
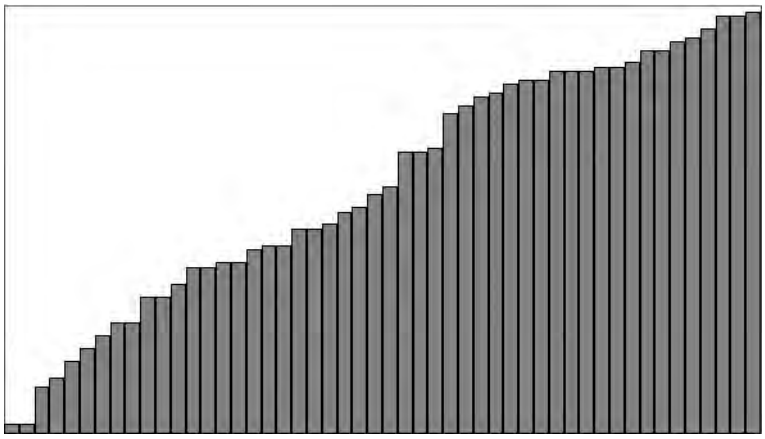


FIGURA 4 Infine l'array è ordinato.



Nell'esempio è stato creato un file di nome **QuickSortAlgorithm.java**, la cui classe estende la classe SoltAlgorithm. Quando poi dovremo richiamare l'applet dalla pagina web, richiameremo il file **SortItem.class** (la versione compilata in bytecode di SortItem.java), dicendogli che l'algoritmo da usare sarà il QuickSort.

Sul sito ftp.infomedia.it è possibile trovare i sorgenti dell'algoritmo di ordinamento. Per far funzionare correttamente l'applet di ordinamento è necessario compilare i file sorgenti e poi creare una pagina html che permetta il caricamento dell'applet. Una volta scaricato ed installato il JDK è possibile compilare i file Java con un unico comando, dalla cartella che contiene i sorgenti delle tre classi proposte:

```
javac
QuickortAlgorith.java
```

A questo punto è necessario creare una pagina html con il codice per richiamare l'applet in questione; basta aprire un comunissimo editor di testi e creare un file con estensione ".htm" (ad esempio QuickSort.htm), inserendo questo tag all'interno del body:

```
<applet codebase="./" code=SortItem.class
width=200 height=200>
<param name="alg" value="QuickSort">
</applet>
```

FIGURA 5 L'applet di ordinamento in funzione sotto Linux.

dove **codebase** identifica la cartella che contiene le classi Java (se ad esempio si carica l'applet da un percorso differente), **code** deve contenere il nome della classe SortItem, e **width** ed **height** rappresentano la dimensione a video.

Per richiamare l'algoritmo specifico bisogna inserire il tag <param> passandogli il parametro **alg** come nome del parametro e QuickSort (cioè la prima parte del nome della classe) come suo valore.

Per visualizzare l'applet, infine, aprire il file html con il proprio browser internet (**Figura 5**).

Calcolo della complessità

La complessità del Quicksort è nell'ordine di O(n*log*n), così come il suo utilizzo di memoria. Come abbiamo già spiegato, la complessità di un algoritmo è in funzione della dimensione di input dell'algoritmo. Un algoritmo come il Quicksort ha quindi una complessità "poco influenzata" dalla dimensione n dei dati che gli si possono passare.

Ma vediamo come sia possibile risalire alla complessità dell'algoritmo analizzando le varie casistiche:

- CASO PEGGIORE – Nel caso peggiore l'algoritmo ha una complessità di $O(n^2)$; il caso peggiore si ha quando un sottoinsieme ha n-1 elementi e l'altro zero elementi.
  Tale situazione si presenta quando l'array è ordinato in maniera crescente o in maniera decrescente;

- CASO MIGLIORE – Nel caso migliore la chiamata ricorsiva opera su sottoinsiemi, uno di grandezza n/2 e l'altro di grandezza n/2-1, in questo caso $T(n) = 2T(n/2) + O(n)$, la cui complessità risulta essere O(n*log*n);

- CASO MEDIO – La particolarità del Quicksort è che anche con una partizione sfavorevole, esempio 9 a 1, mantiene proprietà interessanti, infatti il tempo impiegato sarebbe dettato dalla funzione $T(n) = T(9n/10) + T(n/10) + cn$.
  In definitiva la ricorsione termina alla profondità di $log_{9/10}n \sim O(log n)$, per cui il Quicksort ha complessità O(n*log*n) e la avrebbe anche se la partizione fosse 99 a 1.

L'applet SortItem, la classe SortAlgorithm e la classe QuickSortAlgorithm sono state create da James Gosling e sono di proprietà della Sun Microsystems. Il codice è disponibile e modificabile gratuitamente per scopo non commerciale o commerciale senza fini di lucro, senza nessun supporto e a patto di non modificare il copyright originale. Le prime quattro immagini dell'ordinamento sono tratte da Wikipedia inglese e sono sotto pubblico dominio.

RIQUADRO 1 Copyright

Per poter compilare ed eseguire il programma sono necessari tre strumenti, il compilatore Java (per compilare i file sorgenti), le librerie java e il plugin java per il proprio browser (che cambia anche in base al sistema operativo). Fortunatamente per ottenere tutti gli strumenti necessari al corretto funzionamento, sia per Windows che per Linux, è possibile scaricare il JDK (Java Development Kit), il quale contiene al suo interno il JRE (Java Runtime Environment) con i relativi plugin per abilitare la JVM per il proprio browser. Il JDK è scaricabile dal sito http://java.sun.com/javase/downloads/index.jsp.

RIQUADRO 2 Installare Java

L'algoritmo di ordinamento Shell Sort può essere un ottimo sostituto del Quicksort quando si ha bisogno di un algoritmo di più facile implementazione e molto efficiente.
L'idea di base è quella di estendere l'Insertion Sort (trattato nella prima lezione), il quale si comporta generalmente bene su valori già abbastanza ordinati. Per questo si può suddividere un array in porzioni (di volta in volta più piccole) in cui si ordinano i valori di ciascuna porzione. In pratica una volta diviso l'array iniziale in, ad esempio, tre array, si spostano i valori più piccoli nel primo array, e poi quelli più grandi nel secondo e nel terzo.
Aumentando il numero delle partizioni e ricomponendo l'array iniziale i dati sono praticamente già quasi ordinati. Basta applicare, a questo punto, l'Insertion Sort vero e proprio. Per approfondimenti visitare http://it.wikipedia.org/wiki/Shell_sort.

RIQUADRO 3 Lo Shell Sort

Conclusioni

Con questa ultima puntata abbiamo concluso la trattazione degli algoritmi di ordinamento in Java.

Ripercorrendo le scorse puntate si può capire come il Quicksort sia in parte l'algoritmo "conclusivo", ovvero il miglior compromesso per scopi general purpose.

Si spera per il lettore che lo studio della complessità degli algoritmi sia ora una materia meno oscura e che ciò lo porti ad applicare le conoscenze nel proprio studio e nel proprio lavoro.

Se si vuole approfondire l'argomento si può fare ricorso a Wikipedia che, sia in italiano che in inglese, tratta molto bene gli algoritmi di ordinamento, il calcolo della complessità e le varie implementazioni degli algoritmi.

L'autore rimane a disposizione per ulteriori chiarimenti, consigli e precisazioni al proprio indirizzo email.

CODICE ALLEGATO

ftp.infomedia.it  **Ordinamento4**

Implementazione dell'algoritmo Quicksort

OpInioNi

Phishing, è così facile abboccare?

di Renzo Boni

La parola phishing fa venire in mente la pesca – con la e chiusa – (phishing si pronuncia fiscin, e fish in inglese è il ben noto pesce…), quindi phishing equivale a pescare: gettare l'amo e vedere chi abbocca. E pare che abbocchino in tanti, specialmente se il presunto pescatore si presenta come l'ente "Poste Italiane". Ma stavolta non possiamo biasimare il nostro caro istituto, che anzi nella propria home page ha messo da tempo un avviso contro questa forma di possibile frode telematica.

Gli utenti però ci cascano lo stesso; sono di giorni recenti notizie secondo le quali solo a Roma hanno abboccato in 2.000. Anch'io ricevo giornalmente decine di mail provenienti apparentemente da Poste Italiane, subitamente cancellate senza indugi; ma mai avrei creduto che in così tanti avrebbero dato credito a mail del genere. Non so, forse è l'italico timore verso qualsiasi tipo di "autorità", forse l'inesperienza verso lo strumento dell'ebusiness, o un'eccessiva fiducia verso il mezzo che ci semplifica di molto la vita (evitandoci in questo caso spossanti file agli sportelli postali).

Fatto sta che anche mail scritte con grossolani errori grammaticali, che odorano già a prima vista di falsità, riescono a centrare il bersaglio. È anche vero che senza adeguati strumenti è più difficile annusare l'imbroglio, e una volta accettata come vera l'email iniziale e quindi aperta la pagina web truccata, capire da essa che si stanno inviando informazioni a qualcuno che non è Poste è più difficile; la barra di stato, se attivata, diligentemente ci avverte che il link verso cui ci stiamo

FIGURA 1 La pagina fraudolenta viene bloccata da Firefox

indirizzando non è quello che dice di essere, ma mi rendo conto che utenti inesperti, alle prime armi, o disattenti, possano cadere nel tranello.

A ben guardare le false email si riconoscono in pochi secondi. A cominciare dagli errori grammaticali, intollerabili e impensabili in messaggi autentici; è vero che qualcosa ci può scappare sempre, ma più di un errore nella stessa frase dovrebbe farci drizzare le antenne. Anche il contenuto è indicativo: sentirsi dire che il conto sta per essere chiuso perché "non abbiamo effettuato operazioni nel mese precedente" (tanto più che il costo del servizio prevede una quota mensile, quindi che interesse avrebbe Poste a chiuderlo?) oppure "Messagio di errore. La vostra sessione ha espirato, prego inizio attività ancora. Tutti i diritti hanno riservato" può essere realistico ma solo per servizi amatoriali, gratuiti, un tanto al chilo…

[Tuttavia, quando accedo al mio conto, qualche brivido mi parte in automatico…]

Quindi: qualche giustificazione per l'inesperienza, poche o nessuna per l'ignoranza. Siccome stiamo gestendo i nostri presumibilmente sudati risparmi (pochi o tanti che siano) non è ammissibile dare credito alla prima email che ci arriva e ci chiede tutte le nostre credenziali di accesso. Chi ha attivato un conto online sa benissimo che le stesse credenziali non vengono neanche inviate – per posta ordinaria – tutte insieme, per evitare per l'appunto una sottrazione completa e definitiva. Quindi perché sul web dovrebbe essere diverso?

Forse c'è bisogno di un salto culturale, di una crescita "sociale" che possa ridurre il gap che si è creato tra la tecnologia e la nostra umana capacità – media! – di gestione e comprensione. Così come abbiamo assistito ai falsi funzionari delle Poste che gentilmente si offrono di cambiare le banconote "errate" consegnate al pensionato dall'impiegato postale – ed abbiamo imparato a difenderci – così dovremo fare con gli strumenti telematici.

È necessario prendere la patente per usare questo strumento, come la si prende per guidare un'automobile: entrambi oggetti che possono far danni, per sé o per gli altri. Quindi informarsi, leggere, chiedere, acquisire la consapevolezza che non è un gioco. Per i lettori di Computer

Programming non credo che servano consigli su come comportarsi in tali situazioni; per parenti e amici e – perché no? – clienti, raccomandare senz'altro di usare versioni aggiornate di browser, antispam, antivirus… ma soprattutto usare il buonsenso.

E se ci turbiamo per questo tipo di truffe, ben altra attenzione dovremmo riservare ad episodi sicuramente più nascosti – ma ben più gravi – che da anni stanno accadendo nella quasi generale indifferenza dei non addetti ai lavori. Episodi che hanno a che fare con la violazione della privacy e dei dati personali, ad opera di società che stanno marciando sull'onda dei DRM e dei diritti d'autore, che nell'incertezza della legge, nella sua difformità tra Paese e Paese, nelle varie Authority che forse ben poca autorità hanno, stanno portando un attacco forse mortale allo spirito originario della comunità Internet.

Ricordo ancora – e questo testimonia che ho diversi anni sulle spalle – quando si parlava di Internet come di uno "spazio libero", libero da regole che non fossero l'autodisciplina e l'autocensura degli utenti, una sorta di mondo nuovo e ideale, una "nuova frontiera" tutta da costruire, magari con regole diverse da quanto fatto fino allora. Quando un'azienda che inviava una pubblicità non richiesta veniva collassata di mail di protesta dopo un veloce appello in rete… Adesso queste cose fanno certamente sorridere o sembrano inverosimili ai più giovani.

Ci sono voluti anni di battaglie per avere strumenti che ci tutelino almeno teoricamente dalle intrusioni alle nostre informazioni più personali e riservate, ed ecco che tutto è di nuovo in gioco, col cavallo di Troia del P2P.

Scrivete la vostra opinione a:
red_cp@gruppoinfomedia.it.

Primi passi con Eclipse: scriviamo un Web Service

Scrivere applicazioni con Eclipse può agevolare sensibilmente il compito del programmatore.

di Ludwig Bargagli e Gabriele Fatigati

Con questo articolo mostreremo come creare Web Service, con l'aiuto del tool di sviluppo open source WTP (Web Tools Platform) di Eclipse.

Prerequisiti

Innanzitutto è necessario aver installato nel proprio sistema il JDK, versione 1.5 o superiore. La versione di Eclipse utilizzata è la WTP all-in-one (http://www.eclipse.org/webtools/) che integra la Web Tools Platform 1.5 ed Eclipse 3.2. Questa versione comprende tutte le librerie e gli applicativi necessari per la creazione e la gestione di WS. È necessario installare anche Apache Tomcat (http://tomcat.apache.org/, quello usato è la versione 5.5.17) ed Apache Axis 1.4 (http://ws.apache.org/axis/) per fare il test dell'applicazione.

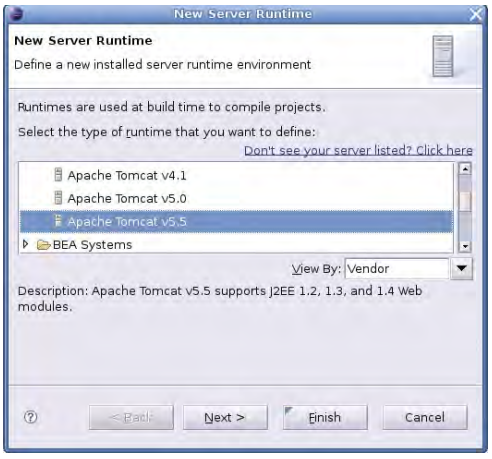
Ludwig Bargagli lbargagli@infomedia.it

È sviluppatore in ambiente enterprise e sistemista presso i "Servizi Informatici" del Comune di Grosseto Si occupa di sicurezza informatica, di sistemi di autenticazione e di sso, di sistemi di archiviazione sostitutiva. Segue studenti universitari per stage e tesi di laurea. È uno dei fondatori del JUG Toscana e di GuruAtWork.

Gabriele Fatigati gfatigati@infomedia.it

Programmatore JSE e JEE docente Java, segue importanti progetti del JUG Toscana e di GuruAtWork.

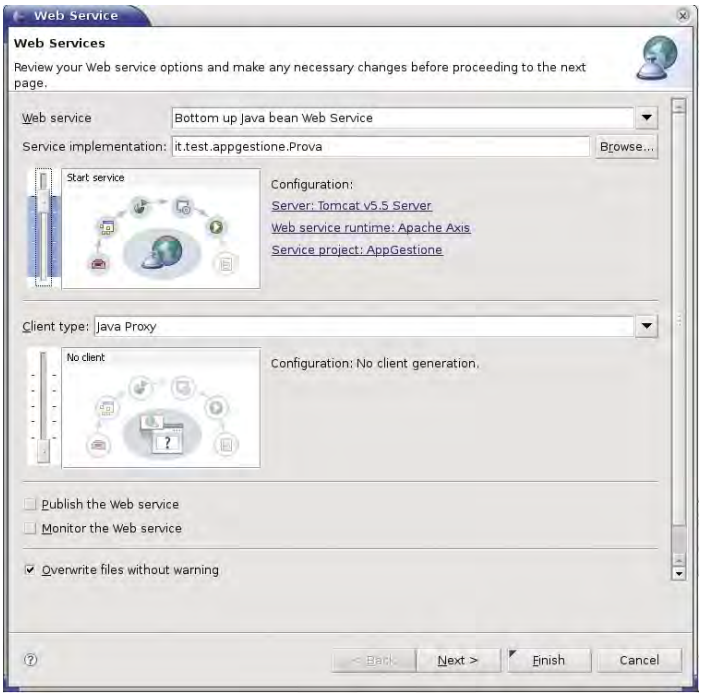
FIGURA 1 Configurazione iniziale



Passi iniziali

Il plug-in WTP permette di generare il Web Service (WS) in due differenti modalità: *Bottom Up* e *Top Down*. La modalità Bottom Up permette di creare il WS partendo da una classe Java; è utile quando vogliamo esporre un WS. La modalità Top Down invece, permette di generare il WS partendo da un file WSDL (Web Service Description Language). Con questo tipo di file è possibile descrivere l'interfaccia pubblica del Web Service (contenente i metodi messi a disposizione), e l'Endpoint, (l'indirizzo a cui è disponibile il Web Service). Questa modalità serve per utilizzare un WS già esposto su Internet.

FIGURA 2 Generiamo il Web Service – Metodo Bottom up



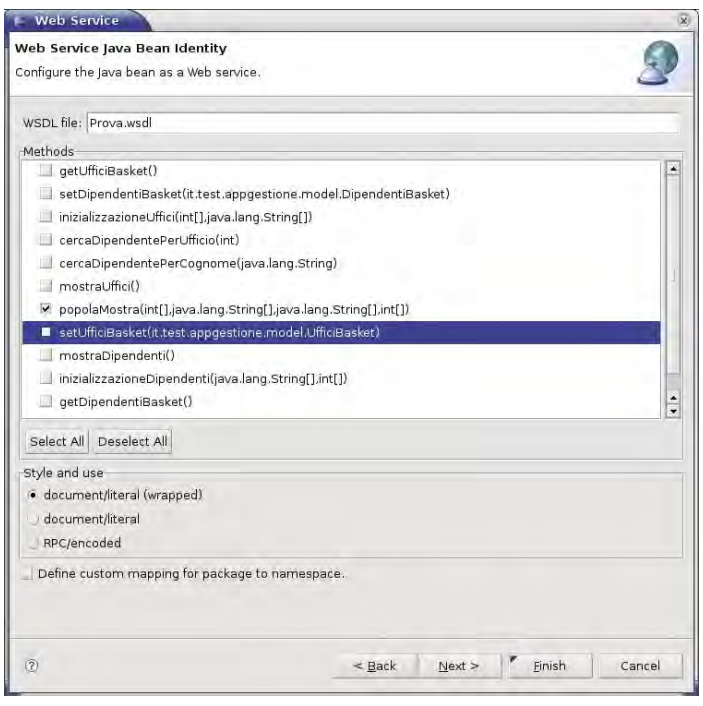
La prima cosa da fare è configurare Eclipse in modo che possa utilizzare l'installazione di Tomcat presente nel sistema: selezionare "Window -> Preferences -> Server -> Installed Runtimes" **(Figura 1)**, cliccare su "Next" ed inserire il percorso di Tomcat e della JRE usata.

Possiamo creare due tipi di progetti: "*Static Project*" e "*Dynamic Web Project*". Il secondo, a differenza del primo, può utilizzare contenuti dinamici, quali ad esempio *servlet* e *jsp*.

Creiamo un Dynamic Web Project: "File-> New-> Other->Web ->Dynamic Web Project". La proprietà *Target Runtime* deve essere impostata alla versione di Tomcat installata.

Le classi che utilizzeremo come esempio sono i due Java Bean descritti negli articoli precedenti []: una classe *Ufficio* ed una classe *Dipendente*. Scriviamo poi una classe (*Prova*.java, package "it.test.appgestione") con un metodo per popolare i bean (*popolaMostra*, **Listato 1**) e funzionalità per fare ricerche nel loro contenuto: cercare un dipendente per matricola, per cognome, per ufficio (**Listato 2**).

FIGURA 3 Selezioniamo i metodi da esporre



LISTATO 1 Metodo popolaMostra

```
public void popolaMostra(int[] id,String[] descIndirizzo,String[] nomecognome,int[] matricolaufficio) {
  setUfficiBasket(inizializzazioneUffici(id,descIndirizzo));
  setDipendentiBasket(inizializzazioneDipendenti(nomecognome,matricolaufficio));
  mostraDipendenti();
  mostraUffici();
  ...
  }
```

Metodologia Bottom Up

Dopo aver scritto la classe *Prova*, possiamo creare automaticamente il Web Service. Selezioniamo la classe appena creata e: "File -> New -> Other -> Web Services -> Web Service" (**Figura 2**).

La funzionalità "Start Service" genera il Web Service. Le funzionalità "Test client", "Test Service" o "Web Service Explorer" generano un'interfaccia per invocarlo. Tramite l'utilizzo delle due barre verticali, selezioniamo le funzionalità "Start Service" e "No Client".

Le impostazioni della configurazione di Tomcat ed Axis sono quelle relative al nostro sistema: "Server: Tomcat v5.5 Server", "Web service runtime: Apache Axis".

Clicchiamo su "Next" e, nel wizard successivo (**Figura 3**), selezioniamo i metodi che vogliamo esporre tramite il WS (in questo caso "popolaMostra") e scegliamo il nome del file WSDL che sta per essere generato. Naturalmente non è necessario che un WS esponga tutti i metodi che abbiamo a disposizione (**Figura 4**).

Eclipse genera il file *web.xml* (contenente le proprietà del WS), il file *WSDL* (**Figura 5**), il file

LISTATO 2 Metodi per le ricerche

```
public Dipendente cercaDipendentePerMatricola(int matricola) {
  Iterator<Dipendente> iter = dipendentiBasket.getDipendenti().iterator();
  while (iter.hasNext()) {
    Dipendente dipendente = iter.next();
    if (dipendente.getMatricola()==matricola) return dipendente;
  }
  return null;
}

public List<Dipendente> cercaDipendentePerCognome(String cognome) {
  List<Dipendente> dipendenti = new ArrayList<Dipendente> ();
  Iterator<Dipendente> iter = dipendentiBasket.getDipendenti().iterator();
  while (iter.hasNext()) {
    Dipendente dipendente = iter.next();
    if (dipendente.getCognome().equals(cognome)) dipendenti.add(dipendente);
  }
  return dipendenti;
}

public List<Dipendente> cercaDipendentePerUfficio(int idUfficio) {
  List<Dipendente> dipendenti = new ArrayList<Dipendente> ();
  Iterator<Dipendente> iter = dipendentiBasket.getDipendenti().iterator();
  while (iter.hasNext()) {
    Dipendente dipendente = iter.next();
    if (dipendente.getUfficio() == idUfficio) dipendenti.add(dipendente);
  }
  return dipendenti;
}
```

WSDD (Web Service Deployment Descriptor), il file *server-config.wsdd* (descrive la configurazione del motore AXIS SOAP).

Il WS è pronto, possiamo verificarne il corretto funzionamento usando la funzionalità "Web Service Explorer". Non possiamo usare la funzionalità "Test Client" perché (al momento) non supporta i metodi che ricevono in input gli array, quindi non è adatta al nostro scopo.

Selezioniamo il file WSDL appena creato e con il tasto destro: "Web Service -> Test with Web Service Explorer" (**Figura 6**).

Notiamo l'*endpoint* del WSDL, ed i campi di input che ci permettono di popolare gli array e passarli al metodo *popolaMostra*. Dopo aver inserito i dati (**Figura 7**) possiamo invocare il Web Service (**Figura 8**).

FIGURA 4 File generati automaticamente

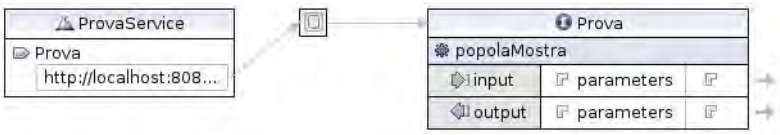
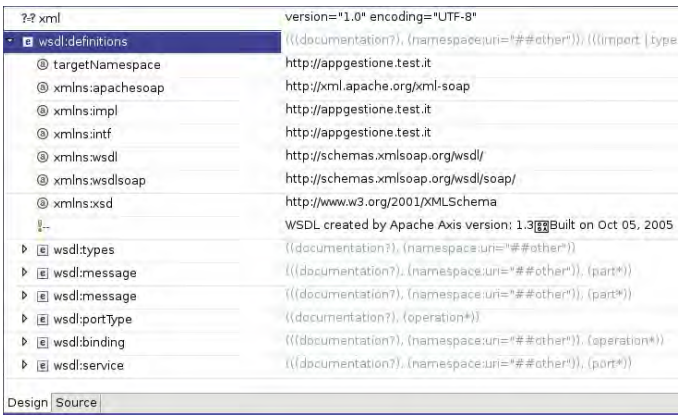


FIGURA 5 File generati automaticamente



Metodologia Top Down

Questa metodologia crea un WS partendo da un file WSDL; è sufficiente indicare il percorso del WSDL ed Eclipse è in grado di generare lo *skeleton* per il server e lo *stub* per il client. Se non abbiamo a disposizione un WS, per mostrare il funzionamento di questa metodologia, possiamo creare una classe wrapper di nome *Prova2*, con il metodo *popolamostra* senza corpo, che servirà per invocare il metodo *popolaMostra* presente nella classe Prova:

```
package servizi;
import java.lang.String.*;
```

FIGURA 6 Facciamo il test del Web Service

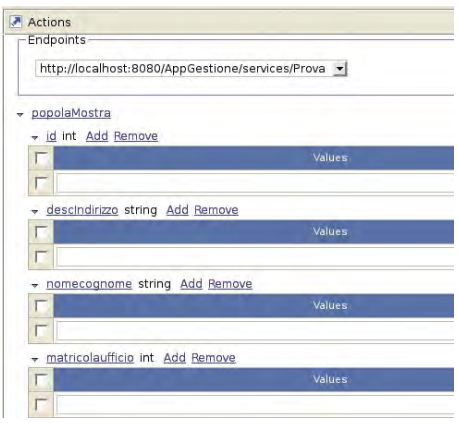


FIGURA 7 Facciamo il test del Web Service

```
public class Prova2 {
    public void popolamostra(int[] id,
            String[] descIndirizzo,String[]
        nomecognome, int[] matricolaufficio){};
}
```

A questo punto possiamo creare il WSDL con la metodologia vista al punto precedente.

Nell'utilizzo reale di questo tipo di metodologia, possiamo semplicemente scaricare il WSDL di un WS esposto in rete. Adesso possiamo generare del Web Service. Selezioniamo la classe Prova2 e, con il tasto destro, clicchiamo su "Generate Web Service", questa volta selezionando la metodologia Top Down. Dobbiamo passargli il WSDL appena generato (**Figura 9**).

Il sistema produce tutte le interfacce necessarie al collegamento con il WS (**Figura 10**):

Prova2.java: contiene la nuova interfaccia del servizio (è stata sovrascritta la classe Prova2 creata in precedenza).

Prova2Service.java: contiene l'interfaccia del servizio per il client.

Prova2ServiceLocator: implementazione dell'interfaccia per il client.

Prova2SoapBindingImpl: implementazione del Web Service per il server.

Prova2SoapBindingSkeleton: skeleton per il server.

Prova2SoapBindingStub: stub per il client.

Questo passo possiamo anche farlo direttamente dal WSDL, cliccandoci sopra con il tasto destro e scegliendo "Generate Java Bean Skeleton". Vengono riprodotti esattamente gli stessi file.

In precedenza abbiamo definito un'interfaccia del servizio; ma dov'è il corpo del metodo esposto? L'implementazione del servizio va inserita nella classe "*Prova2SoapBindingImpl*" come segue:

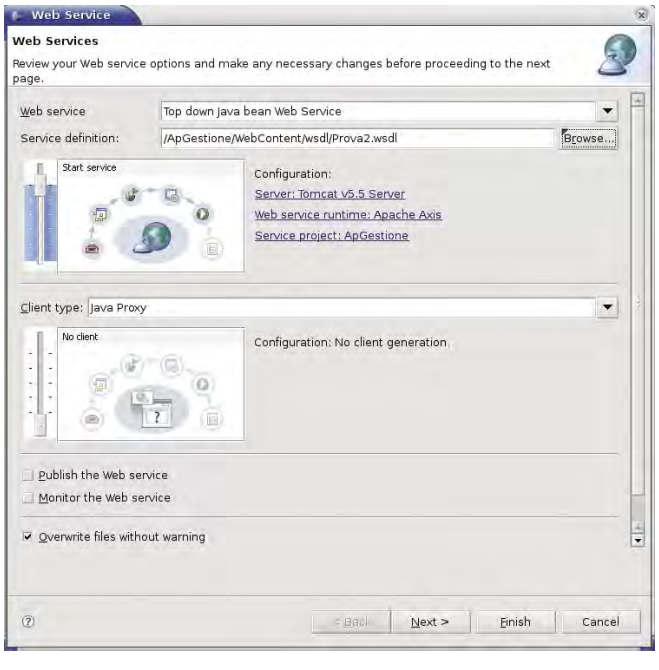
```
package servizi;
import it.test.appgestione.model.*;

public class Prova2SoapBindingImpl implements
                                    servizi.Prova2{
    public void popolamostra(int[] id, java.
                lang.String[] descIndirizzo,
            java.lang.String[] nomecognome,
             int[] matricolaufficio) throws
                    java.rmi.RemoteException {
   Prova prova = new Prova();
   prova.popolaMostra(id,descIndirizzo,
                nomecognome,matricolaufficio)
   }
}
```

Il metodo esposto dal WebService è *popolamostra*, che non fa altro che richiamare il vero metodo *popolaMostra* della classe *Prova*, quella

FIGURA 8 Invochiamo il Web Service

```
Apache Tomcat v5.5 [Apache Tomcat] /home/

Nome: Francesco
Cognome: Verdi
Ufficio: Ambiente

--------
Ufficio: Personale
Id: 1
Indirizzo: Via Roma, 1 - Grosseto

Ufficio: Suap
Id: 2
Indirizzo: Via Roma, 2 - Grosseto

Ufficio: Ambiente
Id: 3
Indirizzo: Via Roma, 3 - Grosseto

--------
Matricola 2: Bianchi
--------
Dipendente: Verdi Simone
Dipendente: Verdi Francesco
--------
Dipendente ufficio 2: Neri Carlo
Dipendente ufficio 2: Verdi Simone
--------
```

FIGURA 9 Generiamo il Web Service – Metodo Top down



in cui vi è tutto il codice che crea e gestisce gli Uffici e Dipendenti.

È sempre buona norma separare le due cose, ovvero l'implementazione vera del servizio e la sua esposizione, ad esempio creando una classe *wrapper* come mostrato.

FIGURA 10 File generati automaticamente

A questo punto abbiamo generato tutti i file necessari ad esporre il WS sul server. Non ci resta che creare il client. Per farlo, scriviamo la classe *Chiama*:

```
package pr;
import servizi.*;
import java.rmi.RemoteException;
import javax.xml.rpc.*;

public class Chiama {
  public static void main(String[]
                args) throws RemoteException,
                             ServiceException {
    Prova2Service  service =
                  new Prova2ServiceLocator();
    Prova2 prova= service.getProva2();
    int[] id= new int[2];
    id[0]=2; id[1]=3;
    String[] descIndirizzo= new String[6];
    descIndirizzo[0]=" Personale";
    descIndirizzo[1]=" Via Bulgaria 7";
    descIndirizzo[2]="SUAP";
    descIndirizzo[3]=" Personale";
    descIndirizzo[4]=" Via Garibaldi 12";
    descIndirizzo[5]="SUAP";
    String[] nomeCognome=new String[4];
    nomeCognome[0]= "Michele";
    nomeCognome[1]= "Massucci";
    nomeCognome[2]= "Antonia";
    nomeCognome[3]= "Sensi";
    int[] matricolaUfficio= new int[4];
    matricolaUfficio[0]=2;
    matricolaUfficio[1]=4;
    matricolaUfficio[2]=3;
    matricolaUfficio[3]=5;
    prova.popolamostra(id,descIndirizzo,
                nomeCognome,matricolaUfficio);
    }
  }
```

L'istruzione

```
Prova2Service  service = new
        Prova2ServiceLocator();
```

richiama il *Locator*, ovvero la classe che fornisce l'*endpoint* del Web Service.

Se diamo uno sguardo all'interno di ProvaServiceLocator, troveremo il codice:

```
private java.lang.String
     Prova2_address = "http://localhost:8080/
                  ApGestione/services/Prova2";
```

che è proprio dove Eclipse ha eseguito il deployment del Web Service.

L'istruzione:

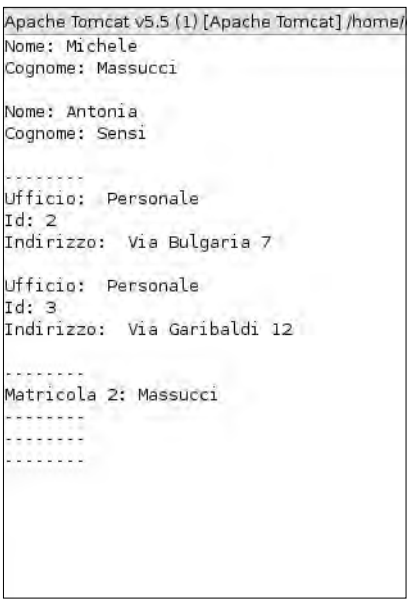
```
Prova2 prova= service.getProva2();
```

crea un riferimento alla classe remota *Prova2*. Da questo momento in poi, i metodi remoti appariranno come locali. Infatti li richiamiamo allo stesso modo:

```
prova.popolamostra(id,descIndirizzo,
                nomeCognome,matricolaUfficio);
```

Il risultato è mostrato in **Figura 11**.

FIGURA 11 Invochiamo il Web Service



Conclusioni

I tool di sviluppo di Eclipse permettono di creare Web Service in modo semplice e veloce, anche se soffrono ancora di qualche problema di gioventù. Forniscono un aiuto concreto sia a chi conosce bene i WS, sia a chi li utilizza da poco tempo, offrendo una buona automazione e rendendo i processi di creazione abbastanza intuitivi. Scrivere un WS senza l'aiuto di questo tool, ad esempio con l'utility (a linea di comando) *WSDLJava*, può risultare molto più complicato e meno intuitivo.

Ambienti di sviluppo come Eclipse WTP semplificano notevolmente la vita dello sviluppatore.

CODICE ALLEGATO

ftp.infomedia.it  **EclipseWS**

Il progetto creato con Eclipse (compreso il codice sorgente). Per utilizzarlo occorre scompattarlo ed importarlo con Eclipse nel workspace corrente.

BIBLIOGRAFIA & RIFERIMENTI

[1] L. Bargagli, G. Fatigati – "Primi passi con Eclipse: scriviamo insieme un'applicazione", Computer Programming n.163
[2] L. Bargagli, A. Fruchi – "Primi passi con Eclipse: scriviamo un'interfaccia Swing", Computer Programming n.168

di Andrea Baruzzo

Progetto e collaudo di gerarchie d'ereditarietà

L'articolo conclude la miniserie sui pattern di test per il collaudo di gerarchie discutendo alcune alternative al Percolation pattern.

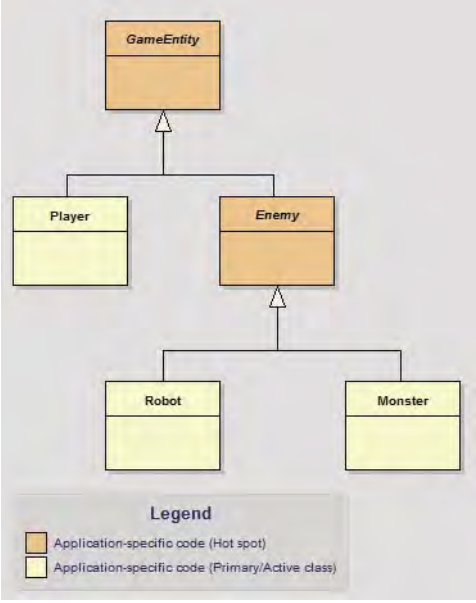
Seconda parte

Si conclude con questa puntata la miniserie sulla progettazione di test driver. Nello scorso articolo abbiamo visto come utilizzare il pattern Percolation [1] per collaudare una gerarchia di classi ed assicurare la conformità rispetto al Principio di Sostituibilità di Liskov [3]. Il Percolation risulta conveniente quando la gerarchia è già dotata di un contratto esplicito, ossia quando le classi sono fornite di invarianti ed almeno tutti i metodi pubblici sono forniti di una specifica che prevede sia le precondizioni, sia le postcondizioni. Ma cosa fare se tali condizioni non sono verificate? Aggiungere manualmente un contratto potrebbe non essere economicamente conveniente, specialmente se è passato del tempo da quando il codice è stato progettato e scritto. Se è già complicato ricordare tutte le assunzioni che abbiamo fatto mentre eravamo noi stessi che scrivevamo il codice, potrebbe inoltre essere addirittura impossibile risalire a tutti i vincoli e le proprietà implicite di un codice scritto da altri, soprattutto in mancanza di una specifica completa. È evidente, quindi, che risulta necessario avere delle tecniche di collaudo alternative al Percolation. In sintesi, questa è la motivazione ai pattern che di seguito verranno presentati. Per esemplificare la struttura statica di ognuno di essi, laddove possibile, riprenderò l'esempio della gerarchia GameEntity, che riportiamo per comodità anche in **Figura 1**.

Andrea Baruzzo abaruzzo@infomedia.it

È laureato in Scienze dell'Informazione presso l'Università degli Studi di Udine. È membro del gruppo infoFACTORY presso il Laboratorio di Intelligenza Artificiale ed Applicazioni Avanzate per la Rete Internet. Si occupa di ricerca, formazione e consulenza sia in ambito accademico, sia in ambito aziendale. Le sue principali aree di interesse sono l'analisi, la progettazione e lo sviluppo di sistemi ad oggetti (OOA/OOD/OOP), la qualità del software e le tecniche machine learning. È inoltre membro di IEEE Computer Society.

FIGURA 1 La gerarchia d'esempio

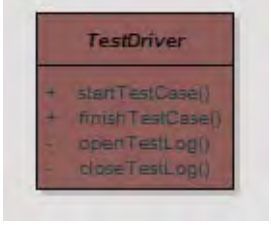


Test Driver Superclass

Il primo pattern che vediamo si chiama Test Driver Superclass [1] e, come il nome suggerisce, funge un po' da infrastruttura per i successivi, in quanto il suo scopo è quello di definire un'interfaccia comune per tutti i test driver più specifici che dovremo implementare nel nostro framework di test. Da un punto di vista implementativo, esso consiste nella scrittura di una classe base astratta contenente l'insieme dei servizi di test che risulta conveniente ereditare (o ridefinire) all'interno dei test driver specifici. La struttura statica di Test Driver Superclass è quindi molto banale, come illustrato in **Figura 2**. Fornire un'interfaccia comune per metodi di servizio (si pensi alla gestione dei log o all'esecuzione dei test case) permette sia di evitare eventuali duplicazioni di codice, sia di ridurre ridondanze logiche indesiderabili nelle interfacce dei test driver. Inoltre la classe base fornisce un singolo e conveniente punto di accesso per definire un'API di test uniforme.

I metodi definiti nella classe *TestDriver* sono ovviamente indicativi. È possibile incapsulare qualsiasi servizio che un cliente del framework può richiedere al test driver.

FIGURA 2 Struttura statica di Test Driver Superclass



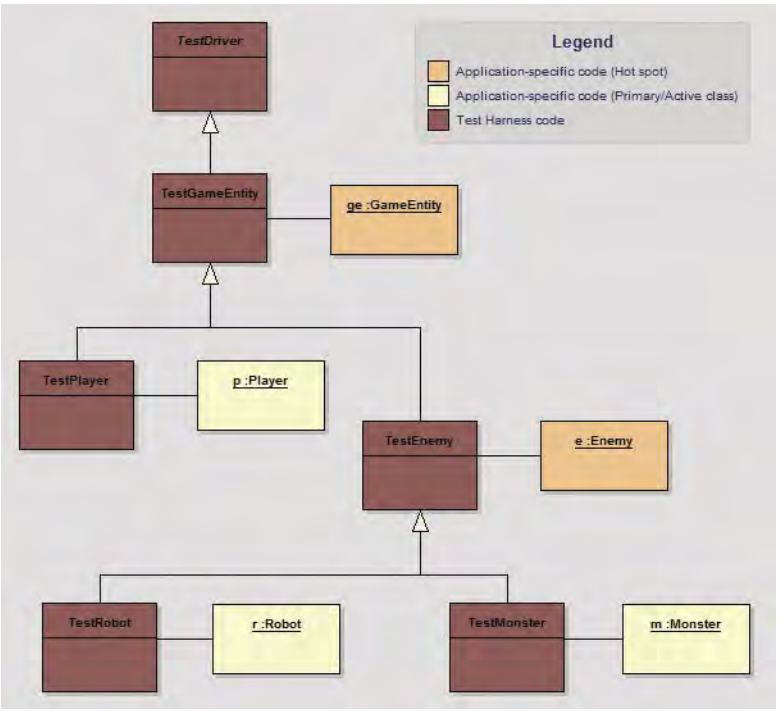
In generale, l'interfaccia della classe *TestDriver* dovrebbe essere suddivisa in due porzioni: la parte pubblica e la parte protetta/privata. Nella parte pubblica, vengono definiti quei metodi che devono rimanere visibili dall'esterno. La loro implementazione, tuttavia, potrebbe essere un dettaglio interno dell'infrastruttura di test, per cui risulta conveniente definire alcuni metodi di supporto usati in tale implementazione, nascondendoli all'esterno. La sezione privata della classe serve esattamente a questo scopo. Al loro interno, infatti, i metodi pubblici sono liberi di effettuare chiamate ai metodi di servizio protetti/privati. Nel nostro esempio, *startTestCase* può essere pensato come un servizio pubblico che un cliente può invocare. La gestione dei log durante l'esecuzione di un test case, invece, può essere considerata una responsabilità del test driver, il quale potrebbe lasciare al cliente solo la possibilità di abilitare/disabilitare il meccanismo di trace e la sorgente su cui scrivere (terminale, file su disco, database, eccetera). In questo caso, perciò, i metodi *openTestLog* e *closeTestLog* sono privati (o protetti). Quando il cliente richiederà l'esecuzione di un test case, il metodo *startTestCase* genererà automaticamente un file di log (o aggiungerà una nuova entry sul file di log corrente), memorizzando le informazioni relative alle variabili di ambiente, alla traccia dei dati di input e ad ogni altro dato utile per diagnosticare un eventuale guasto e ricostruire lo scenario di test in un secondo momento. Le classi specifiche dei test case vengono derivate da *TestDriver*. Si può inoltre usare il pattern Composite [2] per raggruppare più test case in una test suite, così da trattare uniformemente entrambi.

Symmetric Driver

A partire da Test Driver Superclass è possibile definire un'intera famiglia di pattern per il collaudo di gerarchie[1]. Un membro di questa famiglia è Symmetric Driver [4] (**Figura 3**), il cui intento consiste nell'implementare un driver mediante una gerarchia simmetrica rispetto alla gerarchia da collaudare.

1. Più in generale, infatti, una variante di Test Driver Superclass è incapsulata all'interno di praticamente tutti i pattern di test analizzati in questa miniserie.

FIGURA 3 Architettura del pattern Symmetric Driver



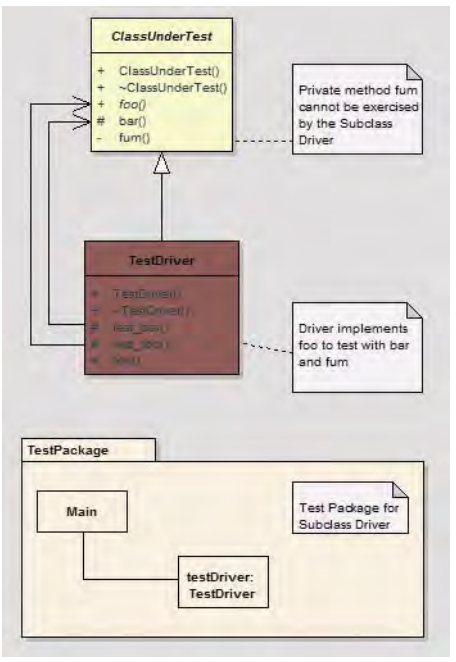
Si tratta di una pratica piuttosto diffusa perché è abbastanza meccanica e, a differenza del Percolation, non richiede la scrittura di asserzioni specifiche tipo gli invarianti e le precondizioni/postcondizioni. Pur essendo ritenuta una soluzione meno costosa per il tester sia in termini di capacità di design, sia in termini di facilità d'utilizzo, essa non è in grado di garantire la conformità di Liskov, né di diagnosticare eventuali fragilità architetturali intrinseche alla gerarchia.

Test Driver Superclass definisce un'interfaccia comune per tutti i test driver

Ogni classe di test in un Symmetric Driver deve essere accoppiata alla sua corrispondente classe applicativa. Ad esempio, se vogliamo collaudare la classe *Player*, dovremmo creare ogni volta un oggetto di quest'ultima all'interno della classe *TestPlayer*, migrando sulla gerarchia di test le stesse relazioni di ereditarietà presenti nella gerarchia applicativa. In altre parole, se nella nostra applicazione *Player* eredita da *GameEntity*, nel framework di test la classe *TestPlayer* dovrà ereditare dalla classe *TestGameEntity*. Per abbinare correttamente le diverse coppie di oggetti applicativi e di test si può utilizzare il pattern Abstract Factory [2] all'interno del main che manda in esecuzione i test.

Come molti altri pattern, anche Symmetric Driver è utile per effettuare sia i test di unità, sia quelli di integrazione, anche in presenza di gerarchie d'ereditarietà multipla. L'ostacolo principale per quanto concerne la fase di test consiste nel fatto che mediante questo test driver è possibile esercitare solamente i metodi pubblici delle classi applicative della gerarchia. Per collaudare i metodi non pubblici, è neces-

FIGURA 4 Architettura del pattern Subclass Driver



sario integrare questo pattern con Private Access Driver, Drone o Built-in Test Driver, a seconda delle specifiche necessità.

Vediamo ora un semplice codice d'esempio (**Listato 1**).

Come esemplificato dal frammento di codice, i metodi di test nei test driver ricevono come argomento l'istanza della classe applicativa che devono collaudare. In questo caso, sarà il main principale ad effettuare il setup di ogni test case. Un'alternativa consiste nel far sì che sia il test driver ad impostare i test case all'interno del proprio costruttore, aggregando la classe applicativa nei suoi attributi.

LISTATO 1

```
class GameEntity {
public:
  void runAI() = 0;
  bool canShootAt(const GameEntity& entity) = 0;
  bool isDestroyable() =0;
  ...
private:
  string m_name;
  int m_nHitPoint;
};

class Player : public GameEntity {
public:
  void runAI() {...}
  bool canShootAt(const GameEntity& entity){...};
  bool isAlive() {return m_health > 0.0};
  ...
private:
  double m_health;
};

class TestGameEntity : public TestDriver {
public:
  // tests of the services in the GameEntity's
  // public interface
  void testRunAI01(GameEntity entity) {
    ...
    bool isThisDestroyable= entity.isDestroyable();
    Assert(isThisDestroyable == true);
    return;
  }
};

class TestPlayer : public TestGameEntity {
public:
  // tests of the services in the Player's
  // public interface
  void testRunAI01(Player player) {...}
    ...
    bool isThisDestroyable= entity.isDestroyable();
    Assert(isThisDestroyable == true);
    bool isThisAlive= player.isAlive();
    Assert(isThisAlive == true);
    return;
  }
};
```

Subclass Driver

La tecnica forse più utilizzata per collaudare una gerarchia è quella definita dal pattern Subclass Driver [5] che consiste nell'utilizzare l'ereditarietà nei linguaggi ad oggetti per implementare un driver di test come sottoclasse della classe da verificare. In questo modo si ereditano automaticamente sia le signature, sia le implementazioni dei metodi da testare, senza bisogno di utilizzare l'aggregazione o il passaggio per argomento per collaudare l'oggetto applicativo. Il Subclass Driver consente il collaudo sia delle classi base, sia delle classi derivate. Ovviamente, per collaudare un metodo astratto contenuto in una classe base, il driver di test deve fornire un'implementazione concreta, in modo da poter risolvere la chiamata polimorfa. Questo, come vedremo, ha almeno un'importante conseguenza a livello di design.

Consideriamo il diagramma di **Figura 4** che illustra l'architettura di Subclass Driver [1]. La classe applicativa *ClassUnderTest* è una classe astratta in quanto contiene la dichiarazione del metodo astratto *foo*. Senza perdita di generalità, immaginiamo che questa classe dichiari tre metodi: uno pubblico (*foo*), uno protetto (*bar*) e uno privato (*fum*). Supponiamo infine che i metodi concreti *bar* e *fum* effettuino al loro interno una chiamata a *foo* (che ovviamente sarà risolta a run-time in base all'oggetto di classe derivata che viene invocato in modo polimorfo). Subclass Driver impone di derivare *TestDriver* da *ClassUnderTest*, implementando i metodi *test_bar* e *test_foo*. Il metodo *bar* fa parte dell'interfaccia (protetta) di *TestDriver* solo per incidente (come classe derivata, *TestDriver* eredita l'implementazione di *bar* fornita dalla classe base).

Consideriamo ora il caso più generale in cui la classe base (astratta) *ClassUnderTest* viene specializzata mediante la classe base *Derived*, come descritto dal seguente frammento di codice.

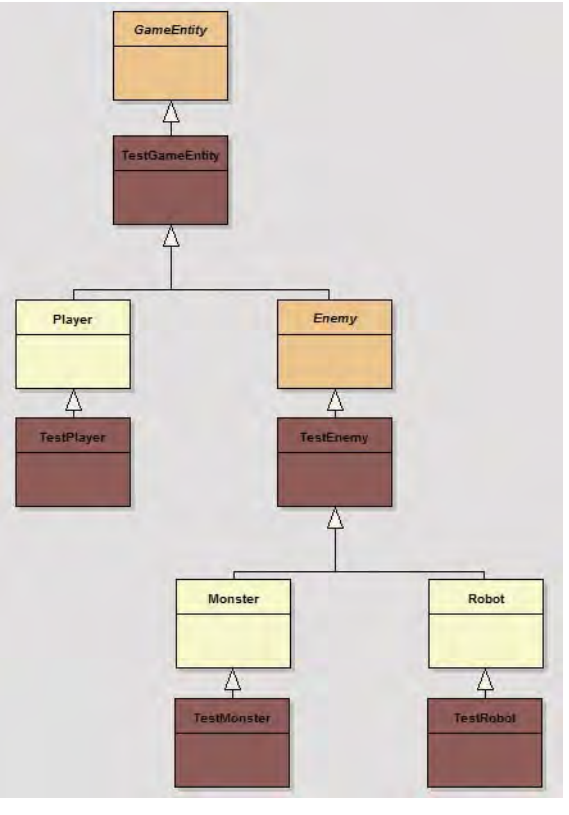
```
class ClassUnderTest {
public:
  void foo()=0; // abstract method
protected:
  void bar(){ // concrete method
    ...
    foo();    // polymorphic call
    ...
  }
private:
  void fum(){ // private method!
    ...
    foo();
    ...
  }
};

class Derived : ClassUnderTest {
public:
  void foo() {...} // foo implementation
protected:
  void bar() {
    ...
    foo();
    ...
  } // bar redefinition
};
```

In tale situazione, è possibile collaudare direttamente solo il metodo *bar* della classe *Derived* poiché quest'ultima fornisce un'implementazione concreta dei metodi *bar* e *foo*. Per collaudare la classe base, invece, il test driver deve prima implementare *foo*. Ciò comporta una potenziale duplicazione di codice. Per scrivere il driver, infatti, dovremmo copiare la definizione del metodo astratto fornita da una qualsiasi classe derivata, oppure potremmo fornire un surrogato di implementazione direttamente a livello del test driver, simulando la funzionalità del vero metodo, ma implementando allo stesso tempo il minimo codice necessario per modificare lo stato interno della classe, come richiesto dallo specifico test case. In entrambi i casi, si tratta di codice aggiuntivo che va scritto manualmente, e che va successivamente manutenuto nella batteria di test, aumentando il costo di realizzazione dell'impalcatura di test. Un'altra limitazione di questo pattern consiste nell'impossibilità di collaudare lo stato privato di una classe, almeno in tutti quei linguaggi che supportano lo scoping privato.

> In Subclass Driver il codice di test diventa a tutti gli effetti un componente dell'intera applicazione

Subclass Driver è molto noto grazie alla sua semplicità, ma riflette anche una cattiva pratica di progettazione, in quanto fa uso dell'ereditarietà per riusare l'implementazione dei metodi di una classe, anche se esclusivamente per ragioni di collaudo. La facilità con cui si abbinano i test case al codice applicativo va quindi bilanciata a livello di design con un forte accoppiamento, in quanto il codice di test diventa a tutti gli effetti un componente dell'intera applicazione, come illustra la **Figura 5** relativa al collaudo

FIGURA 5 La gerarchia d'esempio collaudata con Subclass Driver



della gerarchia d'esempio *GameEntity*. Questo approccio è tollerabile per gerarchie a profondità limitata, tipicamente ad un solo livello, ma risulta poco raccomandabile nel caso generale di gerarchie più profonde. Per la stessa ragione, non risulta un approccio molto scalabile. Per contro, è assai comune vederlo impiegato per collaudare classi concrete che non fanno parte di una gerarchia d'ereditarietà.

Conclusioni

In questa puntata abbiamo esaminato alcuni approcci alternativi al Percolation per collaudare le gerarchie di classi.

Nel corso degli ultimi articoli, abbiamo mostrato come realizzare una semplice impalcatura di test basata sul pattern Incremental Testing Framework per effettuare i test di unità e di integrazione di un'applicazione scritta in un linguaggio orientato agli oggetti. In generale, non esiste una ricetta magica per effettuare il collaudo di un sistema software, per cui diventa molto importante disporre di un insieme variegato di tecniche e strumenti.

Se poi questi strumenti si inseriscono all'interno di un'architettura estendibile e riusabile, sfruttiamo i benefici a medio-lungo termine dell'approccio object-oriented anche per organizzare e scrivere il codice di test.

BIBLIOGRAFIA & RIFERIMENTI

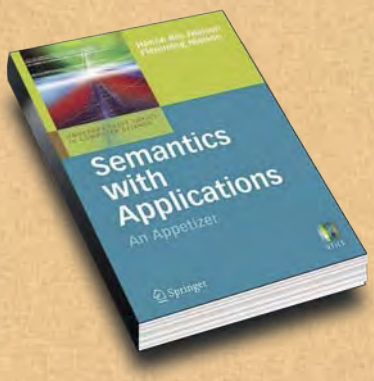
[1] Binder, Robert V. – "Testing Object-Oriented Systems: Models, Patterns, and Tools", Addison Wesley, 2000

[2] Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John M. – "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, 1995

[3] Liskov, Barbara – "Data Abstraction and Hierarchy", SIGPLAN Notices, Vol. 23, N. 5, May 1988

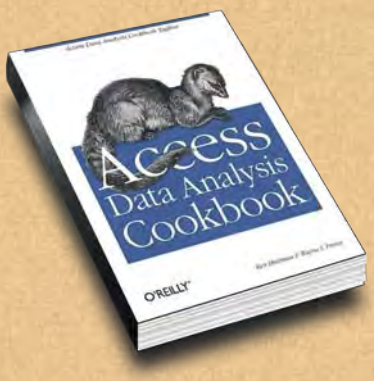
[4] McGregor, John D; Kare, A. – "Parallel Architecture for Component Testing of Object-Oriented Software", In Proceedings, 9th Annual Software Quality Week, S. Francisco, Software Research Institute, May 1996

[5] Firesmith, Donald G. – "Pattern Language for Testing Object-Oriented Software", Object Magazine, Vol. 5, N. 9, pp 32-38, January 1996

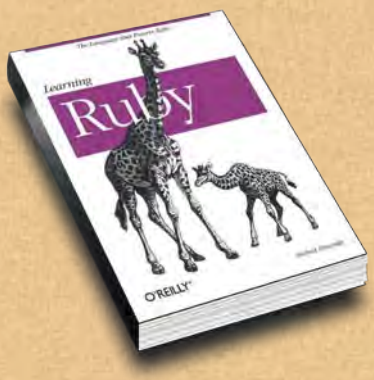


Semantics with Applications
An Appetizer
di H. Riis Nielson e F. Nielson
Springer
278 pp - 31,40 Euro
ISBN 9781846286919

Object-Oriented Programming Languages
Interpretation
di Craig, I. D.
Springer
256 pp - 36,70 Euro
ISBN 9781846287732



Access Data Analysis Cookbook
di K.Bluttman, W.S. Freeze
O' Reilly
366 pp - 47,20 Euro
ISBN 9780596101220

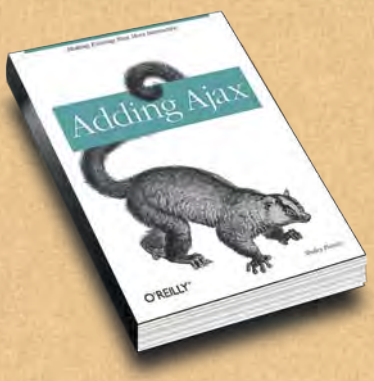


Learning Ruby
di M. Fitzgerald
O' Reilly
255 ppp - 34,50 Euro
ISBN 9780596529864

MAKE: Technology on Your Time
Volume 10
di M. Frauenfelder
O' Reilly
190 pp - 20,20 Euro
ISBN 13: 9780596513863

3ds Max 9.0
Accelerated
di Young Jin
O' Reilly
320 pp - 31,00 Euro
ISBN 9788931433715

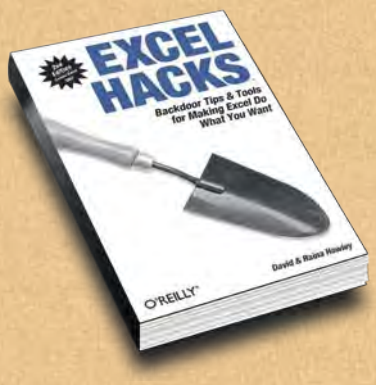
Ableton Live 6
Tips and Tricks
di M. Delaney
PC Publishing-O' Reilly
160 pp - 25,00 Euro
ISBN 9781906005023



Adding Ajax
di S. Powers
O' Reilly
399 pp - 36,50 Euro
ISBN 9780596529369

Beautiful Code
Leading Programmers Explain How They Think
di A. Oram e G Wilson
O' Reilly
618 pp - 45,00 Euro
ISBN 9780596510046

Mastering Perl
di Brian Foy
O' Reilly
368 pp - 40,00 Euro
ISBN 9780596527242



Excel Hacks
Second Edition
di D. Hawley e R.Hawley
O' Reilly
410 pp - 32,00 Euro
ISBN 9780596528348

Fonts & Encodings
di Y. Haralambous
O' Reilly
992 pp - 60,00 Euro
ISBN 9780596102425

Scrivi a **book@gruppoinfomedia.it** specificando nell'oggetto della e-mail:
IN VETRINA Computer Programming n. 171
oppure inviaci il coupon 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**

GRUPPO EDITORIALE INFOMEDIA

CP 171

THE SOFTWARE SIDE OF YOUR MIND

DATI ANAGRAFICI

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

(*) campo obbligatorio

ORDINE

TITOLO	CODICE ISBN	PREZZO	QUANTITÀ

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 000000014804 CAB 25200 ABI 01030 CIN "A" MONTE DEI PASCHI DI SIENA - AGENZIA DI PERIGNANO

☐ CARTA DI CREDITO VISA/MASTERCARD/CARTASÌ

N. SCADENZA

TITOLARE CODICE DI SICUREZZA "CV2"

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.

SI, mi voglio iscrivere gratuitamente alla newsletter Infomedia che mi tiene aggiornato sui prossimi argomenti delle mie riviste di programmazione e mi informa delle offerte, le nuove uscite e le prossime pubblicazioni.
Questo è il mio indirizzo di posta elettronica: _____@_____ Posso annullare la mia iscrizione in qualsiasi momento alla pagina: www.infomedia.it/newsletter.htm

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

OFFERTE ABBONAMENTI
Estate 2007

GRUPPO EDITORIALE INFOMEDIA

RIVISTE CARTACEE

	1 anno	2 anni
COMPUTER PROGRAMMING	€ 55.00 ☐	€ 100.00 ☐
DEV	€ 55.00 ☐	€ 100.00 ☐
VISUAL BASIC JOURNAL	€ 45.00 ☐	€ 80.00 ☐
LOGIN	€ 45.00 ☐	€ 80.00 ☐
JAVA JOURNAL	€ 43.00 ☐	€ 75.00 ☐
CP+DEV+VBJ+LOGIN+JJ	€ 196.00 ☐	€ 380.00 ☐

RIVISTE WEB

	1 anno (IVA INCLUSA)
CP web compreso archivio	€ 45.00 ☐
DEV web compreso archivio	€ 45.00 ☐
VBJ web compreso archivio	€ 35.00 ☐
LOGIN web compreso archivio	€ 35.00 ☐
JJ web compreso archivio	€ 35.00 ☐
2 web a scelta compreso archivio	€ 60.00 ☐
3 web a scelta compreso archivio	€ 85.00 ☐
4 web a scelta compreso archivio	€ 115.00 ☐
5 web compreso archivio	€ 140.00 ☐

- **OFFERTA SPECIALE**: Con soli 20€ di differenza avrai la possibilità di accedere per un intero anno anche alla versione online (compreso archivio) della rivista a cui ti sei abbonato

RIVISTE CARTACEE + WEB

	1 anno (IVA INCLUSA)
COMPUTER PROGRAMMING	€ 75.00 ☐
DEV	€ 75.00 ☐
VISUAL BASIC JOURNAL	€ 65.00 ☐
LOGIN	€ 65.00 ☐
JAVA JOURNAL	€ 63.00 ☐
CP+DEV+VBJ+LOGIN+JJ	€ 296.00 ☐

- I prezzi degli abbonamenti Web sono conprensivi dell' IVA al 20%
- I prezzi degli abbonamenti per l'estero sono maggiorati di spese di spedizione.
- Gli abbonamenti decorrono dal primo numero raggiungibile.
  Per attivazioni retroattive contattaci al numero 0587/736460 o invia una e-mail all'indirizzo: abbonamenti@gruppoinfomedia.it

PRIVACY

Con la presente si autorizza al trattamento dei dati personali ai sensi delle vigenti norme sulla privacy

SI, mi voglio iscrivere gratuitamente alla newsletter Infomedia che mi tiene aggiornato sui prossimi argomenti delle mie riviste di programmazione e mi informa delle offerte, le nuove uscite e le prossime pubblicazioni. Questo è il mio indirizzo di posta elettronica:

_____@_____

Posso annullare la mia iscrizione in qualsiasi momento alla pagina: www.infomedia.it/newsletter.php

Per l'abbonamento in spedizione con **posta prioritaria** aggiungere i seguenti importi:

durata 1 anno:	durata 2 anni:
☐ + € 18.00 per CP	☐ + € 36.00 per CP
☐ + € 18.00 per DEV	☐ + € 36.00 per DEV
☐ + € 10.00 per VBJ	☐ + € 20.00 per VBJ
☐ + € 10.00 per LOGIN	☐ + € 20.00 per LOGIN
☐ + € 10.00 per JJ	☐ + € 20.00 per JJ

Totale ordine € _____

INDIRIZZO DI SPEDIZIONE

Nome/Società _____

Codice Fiscale *(obbligatorio)* _____

Indirizzo _____

CAP _____ Città _____ Prov. _____

Telefono _____ Fax _____

E-Mail _____@_____

Codice Cliente _____

INDIRIZZO DI FATTURAZIONE
(solo per riviste web)

Nome e Cognome _____

Ditta _____

Indirizzo _____

CAP _____ Città _____ Prov. _____

P.IVA _____

E-Mail per invio fattura _____@_____

MODALITA' DI PAGAMENTO

☐ Allego fotocopia del Bonifico Bancario effettuato sul **C/C 000000014804**
CAB 25200 ABI 01030 Cin "A" - Monte dei Paschi di Siena - Agenzia di Perignano

☐ 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 (+ € 11.00 contributo spese postali)

☐ Autorizzo l'addebito dell'importo di € _____ sulla mia CARTASI'/VISA

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

Scad. __ __ / __ __ (mm/aa) Codice di sicurezza "CV2" __ __ __

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)

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 196/03 sulla tutela dati personali.
L'IVA sul prezzo dell'abbonamento cartaceo 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.
Lei può dedurre il costo dell'abbonamento dal reddito d'impresa e dai redditi derivanti dall'esercizio di arti e professioni (artt. 54 e 56 del TUIR)

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

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