

UNIVERSITÀ DEGLI STUDI DI PISA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Laurea Triennale in Informatica

Anno accademico 2001/2002



Progetto di tirocinio:

**Realizzazione di "netT", portale dedicato al
mondo dei trasporti**

Candidato:

Andrea Lorenzani

Tutore accademico:

Prof. Alessandra Di Pierro

Tutore aziendale:

Ing. Fabio Bertella

INDICE

| | |
|--|-----------|
| INDICE..... | 2 |
| PREFAZIONE | 4 |
| CONTENUTO DELLA RELAZIONE | 4 |
| RINGRAZIAMENTI | 4 |
| INTRODUZIONE..... | 5 |
| IL PROGETTO DI TIROCINIO | 5 |
| LA SOCIETÀ OPTISOFT S.R.L. | 5 |
| CAPITOLO 1..... | 7 |
| DESCRIZIONE DEL CONTESTO | 7 |
| 1.1 SCOPO DI TRAVEL..... | 7 |
| 1.2 IL CONTESTO APPLICATIVO | 7 |
| 1.3 L'INSTRADAMENTO E LA SCHEDULAZIONE NELLA DISTRIBUZIONE DELLE MERCI..... | 8 |
| 1.3.1 DEFINIZIONE DEL PROBLEMA..... | 8 |
| 1.3.2 LA SCHEDULAZIONE DEI VIAGGI IN TRAVEL | 9 |
| 1.3.3 L'ALGORITMO DI SCHEDULAZIONE..... | 11 |
| 1.3.3.1 <i>La risoluzione del loading problem</i> | 12 |
| 1.4 L'ARCHITETTURA DI TRAVEL..... | 12 |
| 1.4.1 I DATI GEOGRAFICI | 13 |
| 1.4.2 DATI D'INPUT PER LA SCHEDULAZIONE | 14 |
| 1.4.2.1 <i>I dati relativi a clienti, ordini e articoli</i> | 14 |
| CAPITOLO 2..... | 17 |
| LA TECNOLOGIA IMPIEGATA | 17 |
| 2.1 ASP.NET | 17 |
| 2.1.1 INTRODUZIONE | 17 |
| 2.1.2 MOTIVAZIONI SULLA NUOVA VERSIONE | 18 |
| 2.1.3 LINGUAGGI MULTIPLI | 19 |
| 2.1.4 ELABORAZIONE LATO SERVER | 19 |
| 2.1.5 I CONTROLLI DELLE WEB FORM | 20 |
| 2.1.6 LA GERARCHIA DEI SERVER CONTROL | 20 |
| 2.1.7 I WEB CONTROL | 21 |
| 2.1.8 LA SEPARAZIONE TRA CODICE E CONTENUTO | 21 |
| 2.3 IL LINGUAGGIO C# | 22 |
| 2.3.1 INTRODUZIONE E VISIONE DI INSIEME | 22 |
| 2.3.2 LA MICROSOFT INTRODUCE IL C# | 22 |
| 2.3.3 PRODUTTIVITÀ E SICUREZZA | 23 |
| 2.3.3.1 <i>Abbracciare gli standard emergenti della programmazione Web</i> | 23 |
| 2.3.3.2 <i>Eliminare costosi errori di programmazione</i> | 23 |
| 2.3.3.3 <i>Ridurre i costi di sviluppo in corso col supporto per la versione</i> | 24 |
| 2.3.4 POTENZA, ESPRESSIVITÀ E FLESSIBILITÀ..... | 24 |
| 2.3.4.1 <i>Interazione estesa</i> | 24 |
| 2.2 IL FRAMEWORK .NET | 25 |
| 2.2.1 COSA È .NET? | 25 |
| 2.2.2 LE PARTI DI .NET | 25 |
| 2.2.3 VANTAGGI E SVANTAGGI DELLA PIATTAFORMA .NET..... | 25 |

| | | |
|---------------------|--|-----------|
| 2.2.4 | SULLE FUNZIONALITÀ DEI LINGUAGGI | 26 |
| 2.2.5 | IL “CUORE” DI .NET: IL COMMON LANGUAGE RUNTIME (CLR) | 26 |
| 2.2.6 | I SERVIZI DEL CLR | 28 |
| | IL SISTEMA DI TIPI COMUNE (COMMON TYPE SYSTEM – CTS) | 28 |
| | FIGURA 2.4: STRUTTURA DELLA REFLECTION | 29 |
| 2.2.8 | IL COMMON LANGUAGE SPECIFICATION (CLS)..... | 29 |
| | GLI ASSEMBLY | 30 |
| CAPITOLO 3 | | 32 |
| | IL LAVORO SVOLTO..... | 32 |
| 3.1 | SCOPO DEL PROGETTO..... | 32 |
| 3.2 | LE PAGINE ASP.NET..... | 33 |
| 3.2.1 | LA PAGINA DI REGISTRAZIONE DI UNA AZIENDA | 33 |
| 3.2.1.1 | Descrizione della pagina | 33 |
| 3.2.1.2 | Descrizione del codice | 34 |
| 3.2.2 | LA PAGINA DI AUTENTICAZIONE | 35 |
| 3.2.2.1 | Descrizione della pagina | 35 |
| 3.2.2.2 | Descrizione del codice | 35 |
| 3.2.3 | LA PAGINA PRINCIPALE | 35 |
| 3.2.3.1 | Descrizione della pagina | 35 |
| 3.2.3.2 | Descrizione del codice | 36 |
| 3.2.4 | LA PAGINA DI INSERIMENTO DEI MEZZI..... | 36 |
| 3.2.4.1 | Descrizione della pagina | 36 |
| 3.2.4.2 | Descrizione del codice | 37 |
| 3.2.5 | LA PAGINA DI INSERIMENTO DEI VANI DI UN MEZZO | 38 |
| 3.2.5.1 | Descrizione della pagina | 38 |
| 3.2.5.2 | Descrizione del codice | 38 |
| 3.2.6 | LA PAGINA DI REGISTRAZIONE DEI CLIENTI | 39 |
| 3.2.6.1 | Descrizione della pagina | 39 |
| 3.2.6.2 | Descrizione del codice | 40 |
| 3.2.7 | LA PAGINA DI RICERCA DI UN CLIENTE | 40 |
| 3.2.7.1 | Descrizione della pagina | 40 |
| 3.2.7.2 | Descrizione del codice | 41 |
| 3.3 | I WEB-CONTROL CREATI..... | 42 |
| 3.3.1 | IL CODICE FISCALE..... | 42 |
| 3.3.1.1 | Descrizione del Web Control..... | 42 |
| 3.3.1.2 | Descrizione del codice | 42 |
| 3.3.2 | IL CALENDARIO | 43 |
| 3.3.2.1 | Descrizione del Web Control..... | 44 |
| 3.3.2.2 | Descrizione del codice | 44 |
| 3.3.3 | LA OPTIDATAGRID | 46 |
| 3.3.3.1 | Descrizione del Web Control..... | 46 |
| 3.3.3.2 | Descrizione del codice | 46 |
| | LA OPTIFULLGRID | 48 |
| 3.3.4.1 | Descrizione del Web Control..... | 48 |
| 3.3.4.2 | Descrizione del codice | 49 |
| BIBLIOGRAFIA | | 51 |

Prefazione

Il presente testo costituisce la relazione dello stage svolto presso l'azienda Optisoft S.r.l. di Sarzana (SP), per il conseguimento del primo livello di laurea in Informatica all'Università degli Studi di Pisa. Il progetto di tirocinio assegnatomi aveva per titolo: "Realizzazione di 'netT', portale dedicato al mondo dei trasporti".

Contenuto della relazione

La presente relazione è stata divisa in tre capitoli: nel primo ho descritto il funzionamento di "TRAVEL", applicazione di proprietà dell'azienda in cui lavoravo. Principalmente il mio compito consisteva nel riportare le funzionalità di questa applicazione su Internet, adattandola all'utilizzo via rete. Nel secondo capitolo invece ho trattato la tecnologia che sta alla base del mio lavoro, quindi ho descritto in sintesi linguaggio e tipologia applicativa (ASP.NET con funzioni in C#) e il Framework (.NET) alla base.

Di questi ho trattato anche vantaggi e svantaggi rispetto alle tecnologie precedenti (ASP nel caso di ASP.NET e C/C++ nel caso di C#).

Nell'ultimo capitolo infine ho preso in esame il mio lavoro presso l'azienda. Ho descritto con precisione tutta l'attività svolta, trascurando la parte riguardante l'interazione col database, poiché questo lavoro è risultato particolarmente ripetitivo (si è trattato di creare semplicemente qualche tabella e di scrivere qualche richiesta).

Ringraziamenti

Ringrazio tutto lo staff dell'azienda Optisoft per aver messo a disposizione gli strumenti (risorse fisiche, documentazione e testi) necessari alla stesura di questa relazione e per la disponibilità dimostrata. In particolare ringrazio l'ingegnere Fabio Bertella, che ha supervisionato tutto il progetto, oltre ad aver consigliato testi e articoli sia per documentarmi sia per scrivere questa relazione, Valentino Orlandi, che ha dato vita al progetto in cui ho lavorato e mi ha aiutato a comprendere il funzionamento di tutti gli strumenti di cui ho avuto bisogno e Ettore Nardone, il cui aiuto si è mostrato rilevante durante il lavoro di stesura di questa relazione. Ringrazio infine la tutrice accademica, professoressa Alessandra Di Pierro.

Introduzione

Il progetto di tirocinio

Il progetto di tirocinio da me svolto “Realizzazione di ‘netT’, portale dedicato al mondo dei trasporti” ha riguardato principalmente il software applicativo TRAVEL.

L’attività svolta nell’ambito di questo progetto consisteva nel riportare alcune funzionalità di questa applicazione (quali ad esempio la registrazione delle tipologie di mezzi, dei vani per ogni tipo di mezzo e dei clienti) sulla rete, in modo da renderla parte del portale che l’azienda Optisoft sta creando.

Ciò ha comportato la creazione di pagine Web per l’autenticazione dell’utente che sta richiedendo il servizio. Infatti se TRAVEL è un’applicazione che viene usata solo all’interno di un’azienda (e quindi i vari utenti appartengono anche loro sicuramente alla stessa azienda), nel momento che si rende disponibile il servizio su rete, si ha bisogno di distinguere i vari utenti e di rendere quindi inaccessibili i dati personali di un usufruttore del servizio da un altro.

Infine il mio lavoro ha compreso la creazione di Web Control (si rimanda al capitolo 2 per una spiegazione dettagliata di cosa sia un Web Control), utili per velocizzare la creazione delle future pagine che faranno parte del portale.

La società Optisoft S.r.l.

Optisoft S.r.l. nasce nel 1992 in seguito all’iniziativa di alcuni liberi professionisti specializzati nello sviluppo di applicazioni in ambiente Windows e nella realizzazione di Algoritmi di Ottimizzazione. I componenti di *Optisoft S.r.l* vantano un’esperienza nella realizzazione di applicazioni in ambiente Windows che risale ai primi mesi del 1990 (versione Beta di Windows 3.0). In quel periodo tale sistema operativo era conosciuto da pochi esperti del settore e non era assolutamente profetizzabile che sarebbe divenuto lo standard per la quasi totalità dei Personal Computers. La conoscenza approfondita degli strumenti di sviluppo in ambiente Windows, a partire dal SDK 3.0 (Software Development Kit) del 1990 fino ad arrivare all’odierno Visual C++ .NET ha fatto sì che *Optisoft S.r.l* sia diventata il punto di riferimento per quelle aziende che intendono sviluppare applicazioni professionali negli ambienti Windows (Windows 98/ME, Windows 2000/XP). Oggi l’impegno è nell’attività dell’Internet Programming, impiegando XML/ASP perché *Optisoft S.r.l* è convinta che tale strumento diverrà il riferimento per le applicazioni di rete e non solo. *Optisoft S.r.l* vanta, tra i propri clienti, società di profilo nazionale ed internazionale: IBM Semea, Alenia, Eltag, Marconi Communications, Eli Lilly, Gruppo Maggioli, JMAC (Gruppo SIEL), SET, Pioneer Hi Bred. Una

serie di grandi gruppi che vedono in *Optisoft S.r.l* una risorsa per consulenze o implementazioni e approfondite conoscenze sistemistiche in ambiente Internet.

L'attività di *Optisoft S.r.l.* è rivolta principalmente alla progettazione ed alla vendita di applicazioni software sia per conto proprio sia per conto terzi. L'azienda svolge inoltre, sempre in ambito informatico, attività di consulenza e corsi di formazione professionale.

Optisoft ha realizzato numerose applicazioni software tra cui:

- Programmi gestionali di contabilità
- Sistemi Client/Server
- Sistemi di monitoraggio
- Interfacce per software OCR
- Software per la conversione di documenti in formato EDI (Electronic Data Interchange)

L'azienda svolge attività di consulenza presso importanti aziende tra cui Elsag S.p.A., Gruppo Maggioli, IBM Semea, Marconi Communications, Pioner Hi Bred, Kataweb, e svolge attività di formazione e riqualificazione del personale nei seguenti campi:

- Internet Programming
- Metodologie di programmazione (OOP, Event Driven Programming)
- Tools di sviluppo software (Visual C++, Java, Visual .NET, Visual Basic .NET)
- Architettura sistemi operativi Microsoft
- Office Automation.

Capitolo 1

Descrizione del contesto

Dato che il progetto di tirocinio ha riguardato il software applicativo TRAVEL, in questo capitolo viene presentato lo scopo dell'applicazione e il modo in cui essa affronta il problema della schedulazione dei viaggi su strada.

1.1 Scopo di TRAVEL

TRAVEL è un software applicativo sviluppato allo scopo di pianificare i viaggi su strada di una flotta di automezzi impiegati per il ritiro e/o la consegna di merci.

La pianificazione (schedulazione) dei viaggi determinata dall'applicazione risulta nell'ottimizzazione del trasporto delle merci; pertanto, si desidera razionalizzare l'impiego delle risorse e ridurre i costi, rispettando un determinato insieme di vincoli, propri del particolare contesto operativo.

In sintesi, gli obiettivi più comuni di TRAVEL sono tre:

- massimizzare il numero delle consegne effettuate dagli automezzi;
- minimizzare il costo di consegna;
- coprire in modo equilibrato i mezzi di trasporto disponibili.

1.2 Il contesto applicativo

Il tipico contesto d'utilizzo per TRAVEL è quello di un'azienda o società che abbia la necessità di spedire merci a partire dagli ordini ricevuti.

In generale, ogni giorno, l'azienda deve evadere un certo numero di ordini a più clienti, utilizzando i mezzi di trasporto disponibili. In vista di ciò, è necessario organizzare o pianificare i viaggi degli automezzi. Questo comporta valutare quanti automezzi impiegare, assegnare ad ogni mezzo un certo numero di ordini, soddisfacendo determinati vincoli, come la capacità di carico massima dell'automezzo, il rispetto delle fasce orarie di consegna indicate dal cliente, ecc.

L'obiettivo della pianificazione, dipendente dalla politica aziendale, può essere quello di consegnare il maggior numero di ordini possibile, minimizzando i costi; tra l'altro, questo può significare la necessità di ottimizzare il caricamento di ogni automezzo impiegato e stabilire una rotta che minimizzi il percorso effettuato (in termini di tempo di percorrenza o di distanza).

1.3 L'instradamento e la schedulazione nella distribuzione delle merci

1.3.1 Definizione del problema

Il problema affrontato da TRAVEL rientra nella categoria dei cosiddetti *Vehicle Routing Problem* o *VRP* (problema d'instradamento di veicoli).

In termini generali, il problema consiste nella distribuzione di n ordini (relativi a uno o più depositi) a più clienti sparsi geograficamente, utilizzando i mezzi di trasporto disponibili e sottostando a un insieme di vincoli imposti.

La soluzione del problema comporta la determinazione di un insieme di itinerari, ognuno dei quali è assegnato a un solo veicolo, che parte da un certo deposito e ritorna allo stesso, e tali da soddisfare gli ordini dei clienti, rispettare i vincoli imposti e raggiungere uno o più obiettivi di ottimizzazione che dipendono dalla politica aziendale applicata.

Nella definizione di un problema di *vehicle routing* devono essere presi in considerazione numerosi fattori inerenti alle componenti fondamentali del problema stesso (tra cui i veicoli, i clienti e i depositi) cui si aggiungono gli obiettivi da raggiungere e i vincoli da rispettare. Alcuni di questi fattori sono:

- In relazione ai veicoli: la grandezza della flotta, il tipo di automezzi disponibili (ad esempio, nel caso di veicoli adibiti al trasporto alla rinfusa rivestono particolare importanza il numero e le caratteristiche degli scompartimenti di cui il mezzo è dotato, oppure, nel caso di trasporto di merce deperibile, il veicolo dovrebbe disporre di una cella frigorifera), la disponibilità temporale, il costo di gestione, il tipo di operazione da effettuare che si distingue in consegna (*delivery*) e ritiro (*pickup*).
- In relazione ai clienti: gli intervalli di tempo o *finestre temporali* in cui il cliente può essere servito (orari di apertura od orari in cui è permesso il transito in zone a traffico limitato), la loro locazione, la quantità di merce che deve essere consegnata o prelevata.
- In relazione ai depositi: il loro numero, la loro collocazione, il tipo e il numero di veicoli che vi fanno riferimento.

Per quanto concerne l'obiettivo di ottimizzazione, quest'ultimo potrebbe consistere in uno dei seguenti (o in una loro combinazione):

- minimizzazione dei costi di gestione dovuti all'impiego dei mezzi e del personale;
- minimizzazione dei ritardi di consegna;
- minimizzazione della distanza percorsa o del tempo di viaggio;
- minimizzazione del numero di veicoli e/o autisti necessari;
- bilanciamento dei diversi viaggi in relazione alla distanza percorsa e alla quantità di lavoro associata.

Se l'obiettivo di ottimizzazione che ci si prefigge è composito, allora nella funzione obiettivo si tiene conto di ogni componente attribuendole un particolare peso.

Inoltre, è necessario tenere presente l'insieme dei vincoli. Questi includono il luogo di partenza dei veicoli, la durata del periodo per cui si effettua la schedulazione (ad esempio, non bisogna

trascurare il fatto che un conducente non può guidare più di un certo numero di ore), il rispetto delle finestre temporali indicate dai clienti, il carico dei veicoli in relazione al loro volume, alla loro portata, al tragitto che percorreranno (i mezzi potrebbero dover attraversare ponti o viadotti per i quali è prevista una portata massima)...

Le caratteristiche fisiche dei veicoli non incidono solo sul problema della schedulazione e instradamento; infatti, nel caso in cui i veicoli siano dotati di scompartimenti o vani, si aggiunge spesso un secondo problema: quello della gestione ottima del carico (*loading problem*); a questo proposito, è necessario massimizzare la capacità dei vani che rimangono non occupati in modo tale da riuscire a garantire la soddisfacibilità di richieste successive.

Un altro aspetto del problema dell'instradamento è costituito dalla rete stradale impiegata dai veicoli per i loro viaggi: essa è normalmente rappresentata mediante un grafo i cui nodi corrispondono a punti rilevanti della rete stessa (incroci stradali, locazione di clienti e depositi) e i cui archi rappresentano le connessioni stradali. Agli archi è associato un peso che può consistere nel costo di transito (distanza geografica, importo di pedaggi...) o nel tempo di percorrenza.

Nei paragrafi seguenti (1.3.2 e 1.3.3), viene illustrato in che modo TRAVEL affronta il VRP: in particolare, si mostra il processo di schedulazione eseguito dall'applicazione e si presentano alcuni dettagli relativi all'algoritmo impiegato.

1.3.2 La schedulazione dei viaggi in TRAVEL

TRAVEL si propone di risolvere il problema della pianificazione dei trasporti impiegando un algoritmo di ottimizzazione che produce un piano di viaggi sulla base degli ordini da evadere, i clienti da raggiungere e gli automezzi a disposizione nel periodo che si desidera schedulare, nel rispetto dei vincoli di contesto (si veda la figura 1.1). Per viaggio si intende il percorso fatto da un mezzo per distribuire n ordini a m clienti ed eventualmente tornare al deposito da cui è stata prelevata la merce.

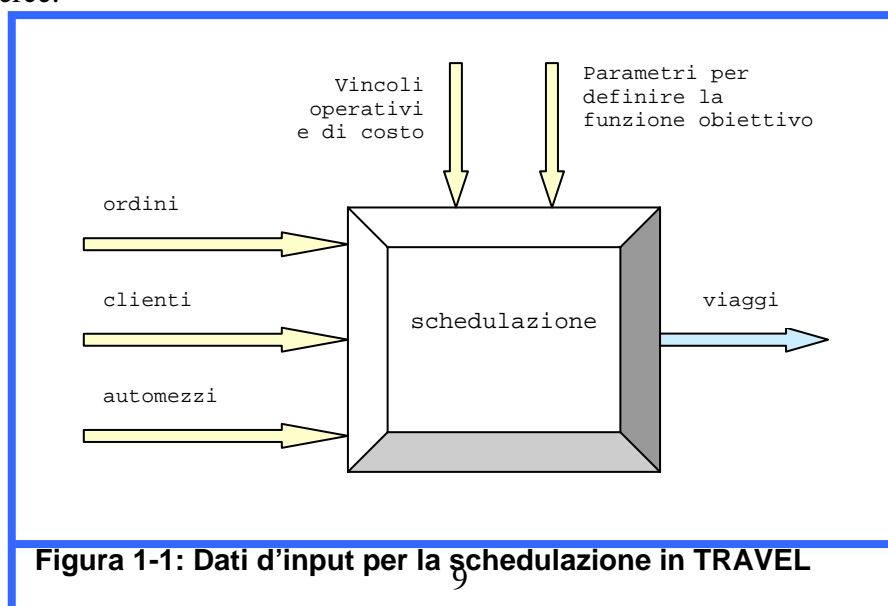


Figura 1-1: Dati d'input per la schedulazione in TRAVEL

L'operazione preliminare è quella di impostare correttamente tutti i dati d'input attraverso le apposite interfacce; i dati d'input includono determinati parametri che entrano in gioco nella definizione della funzione obiettivo come:

- Costo minimo: valore numerico che corrisponde al peso che la minimizzazione dei costi deve avere nel calcolo della funzione obiettivo.
- Saturazione equilibrata: valore numerico corrispondente al peso che la ripartizione equilibrata delle ore lavorative tra i mezzi deve avere; inserendo un valore alto per questo parametro si ottiene come obiettivo principale un'equa assegnazione dei viaggi tra i mezzi disponibili.
- Funzione di bontà: valore numerico che indica quanto incide la massimizzazione della saturazione specificata.
- Minimo numero di viaggi: valore corrispondente al peso che la minimizzazione del numero dei viaggi deve avere nella funzione obiettivo.
- Minimizzazione del percorso in base alla distanza (in questo caso, verrà ricercato il percorso più breve) o al tempo (in tal caso, sarà determinato il percorso più veloce).

Pertanto, la funzione obiettivo si compone attribuendo un coefficiente di peso a ciascuno degli obiettivi sintetizzati nel paragrafo 1.1, decidendo quale privilegiare secondo la politica aziendale applicata.

Dopo questa fase d'impostazione dei parametri, ha inizio il vero e proprio processo di ottimizzazione che è suddiviso in due fasi:

1. Generazione di tutti i viaggi possibili compatibili coi vincoli imposti.
2. Scelta della combinazione ottimale dei viaggi in base alla funzione obiettivo selezionata dall'utente.

Nella prima fase vengono determinati e memorizzati tutti i viaggi possibili che soddisfano le seguenti condizioni o vincoli:

- Compatibilità tra tipo di mezzo e caratteristiche degli impianti del singolo cliente.
- Percentuale di carico superiore al valore scelto dall'utente.
- Rispetto della massima capacità di carico dei mezzi (sia in volume che in peso).
- Tempo complessivo (inclusi i tempi di carico e scarico della merce) inferiore alla disponibilità temporale del mezzo.

Nella seconda fase si esplora lo spazio dei viaggi generati alla ricerca di una soluzione ottimale; l'obiettivo da raggiungere è stabilito dall'utente tramite l'impostazione dei parametri, per cui in un caso una soluzione verrà considerata migliore della precedente se ha un costo minore, in un altro caso se distribuisce meglio i viaggi tra i vari mezzi e così via. Anche in questa fase una condizione necessaria è che il tempo totale dei viaggi assegnati a un certo mezzo non superi il tempo massimo disponibile per quel mezzo. I vincoli da soddisfare riguardano la compatibilità tra gli orari di apertura del cliente, le fasce orarie lavorative del mezzo e il tempo di percorrenza di ogni viaggio; per raggiungere quest'obiettivo possono essere aggiunti dei tempi di attesa tra una tappa e l'altra.

L'utente esercita un pieno controllo sulla soluzione ottimizzata; infatti ogni viaggio proposto dall'algorithm può essere cambiato tramite un'apposita interfaccia che permette la modifica e/o la costruzione di un viaggio manualmente.

Quindi, l'applicazione può essere utilizzata nelle seguenti tre modalità:

1. Automatica: i viaggi vengono proposti automaticamente dal sistema sulla base degli obiettivi e dei vincoli impostati dall'utente;
2. Manuale: l'utente viene guidato dal sistema nella costruzione dei viaggi passo per passo, con opportune segnalazioni dei vincoli da rispettare;
3. Semi-automatica: i viaggi, dopo essere stati proposti automaticamente, vengono rivisti ed eventualmente modificati dall'utente.

L'utente può passare da una modalità all'altra in qualsiasi momento, ad esempio annullando un viaggio proposto dal sistema e ricostruendolo manualmente.

Inoltre, è possibile effettuare alcune simulazioni cambiando l'ampiezza e la disponibilità del parco automezzi oppure modificando alcuni parametri di ottimizzazione, in modo da valutare gli effetti che alcuni cambiamenti nella gestione o negli obiettivi avrebbero nella risoluzione del problema dei trasporti; in questo modo, è possibile ottenere una pianificazione strategica della distribuzione, determinando, ad esempio, il corretto dimensionamento del parco veicoli.

Il percorso relativo ai viaggi determinati dal processo di schedulazione o costruiti manualmente può essere visualizzato sulla mappa stradale di cui TRAVEL dispone.

1.3.3 L'algoritmo di schedulazione

In letteratura esiste un vasto numero di algoritmi creati per risolvere problemi di *vehicle routing*. Ogni algoritmo ha un proprio dominio applicativo dove riesce a fornire buoni risultati, tuttavia il limite che spesso presentano questi algoritmi consiste nella loro specificità: introducendo un nuovo vincolo o modificando la funzione obiettivo, tali algoritmi diminuiscono la loro capacità di ottimizzazione. Per questa ragione, in TRAVEL si utilizza un algoritmo meno specifico, che è pertanto in grado di funzionare correttamente su un vasto insieme di casi.

L'approccio algoritmico utilizzato è di tipo *enumerativo* (o *a esaurimento*), ovvero si opera per enumerazione o esaurimento di tutte le soluzioni possibili, alla ricerca della soluzione ottima in accordo alla funzione obiettivo definita; più precisamente, si esplora l'insieme o albero delle soluzioni fino alla determinazione di un nodo foglia che corrisponde a una soluzione ottima.

In particolare, l'algoritmo consta dei seguenti tre passi:

1. Costruzione di tutti i viaggi possibili compatibili coi vincoli imposti.
2. Selezione e ordinamento dei viaggi ritenuti accettabili in base alla funzione di bontà.
3. Scelta di quell'insieme di viaggi che massimizzi una funzione di costo parametrizzabile sulla base delle esigenze del particolare contesto applicativo.

Il primo e il terzo passo utilizzano un'esplorazione dell'insieme delle soluzioni di tipo *branch and bound*. Il secondo passo usa l'algoritmo di ordinamento *QuickSort*.

In riferimento al primo passo, la tecnica del *branch and bound* (o interruzione) consiste nell'arresto della computazione lungo ogni ramo dell'albero delle soluzioni che risulti incapace di generare una soluzione ammissibile (cioè compatibile coi vincoli imposti).

Il terzo passo che normalmente viene risolto utilizzando tecniche di programmazione intera lineare o quadratica, è invece risolto tramite un'esplorazione combinatoria incompleta dell'albero delle soluzioni. Tale esplorazione utilizza un limite che permette di scartare i sottoalberi riconosciuti incapaci di fornire soluzioni interessanti, ed è diretta da funzioni statistiche in grado di riconoscere i sottoalberi più ricchi di soluzioni, nei quali viene pertanto intensificata la ricerca.

È necessario notare che questa metodologia di risoluzione non presenta le tipiche instabilità riscontrabili utilizzando tecniche di programmazione intera lineare o quadratica. Inoltre, l'impiego di queste ultime comporterebbe dei problemi in relazione alle diverse politiche aziendali applicabili; tra questi problemi risultano importanti la difficile parametrizzazione e personalizzazione dei vincoli, la funzione obiettivo lineare ecc.

1.3.3.1 La risoluzione del *loading problem*

Si è dichiarato come al problema di schedulazione e instradamento se ne può aggiungere un secondo, nel caso in cui il veicolo sia dotato di vani o scompartimenti: quello relativo al caricamento della merce (*loading problem*). Per questo problema, l'obiettivo di ottimizzazione è quello di massimizzare il volume complessivo dei vani rimasti non occupati dopo il caricamento di una data quantità di merce.

Il procedimento algoritmico risolutivo si basa sulla selezione del minimo numero di scompartimenti sufficiente a contenere i prodotti in conformità con l'obiettivo prefissato.

Perciò, si ordinano i vani in modo crescente e si selezionano i primi n vani (partendo con $n = 1$) sufficienti a garantire il caricamento della merce. Nel caso in cui non sia possibile disporre i prodotti in un solo vano si procede a calcolare le possibili *combinazioni* di n ($n \geq 2$) scompartimenti. L'elaborazione si arresta al primo valore di n accettabile.

1.4 L'architettura di TRAVEL

TRAVEL è un'applicazione costituita da due principali componenti:

- Un modulo gestionale, relativo alla gestione dei dati necessari per il funzionamento del sistema.
- Un modulo operativo, da cui è possibile avviare la schedulazione dei viaggi.

I dati impiegati dall'applicazione risiedono in una base di dati in cui si possono distinguere due insiemi di dati: il primo è l'insieme dei dati geografici che contiene le informazioni necessarie alla

rappresentazione della mappa stradale di cui il sistema è dotato (tali informazioni includono i nodi corrispondenti alle varie località geografiche e le connessioni tra i nodi stessi, ovvero le strade).

Il secondo insieme include i dati d'input per la schedulazione dei viaggi: i dati relativi agli automezzi, ai clienti, agli ordini che devono essere evasi e agli articoli inclusi negli ordini. Le informazioni relative agli ordini, agli articoli e ai clienti sono prelevate (eventualmente) dalla base di dati aziendale.

L'architettura del sistema è riassunta in figura 1.2.

Nei prossimi due paragrafi si forniscono ulteriori dettagli relativi a tali dati; più precisamente, si descrivono le tabelle da cui si ricavano i dati di carattere geografico e si presentano alcuni particolari relativi alle tabelle del secondo insieme di dati.

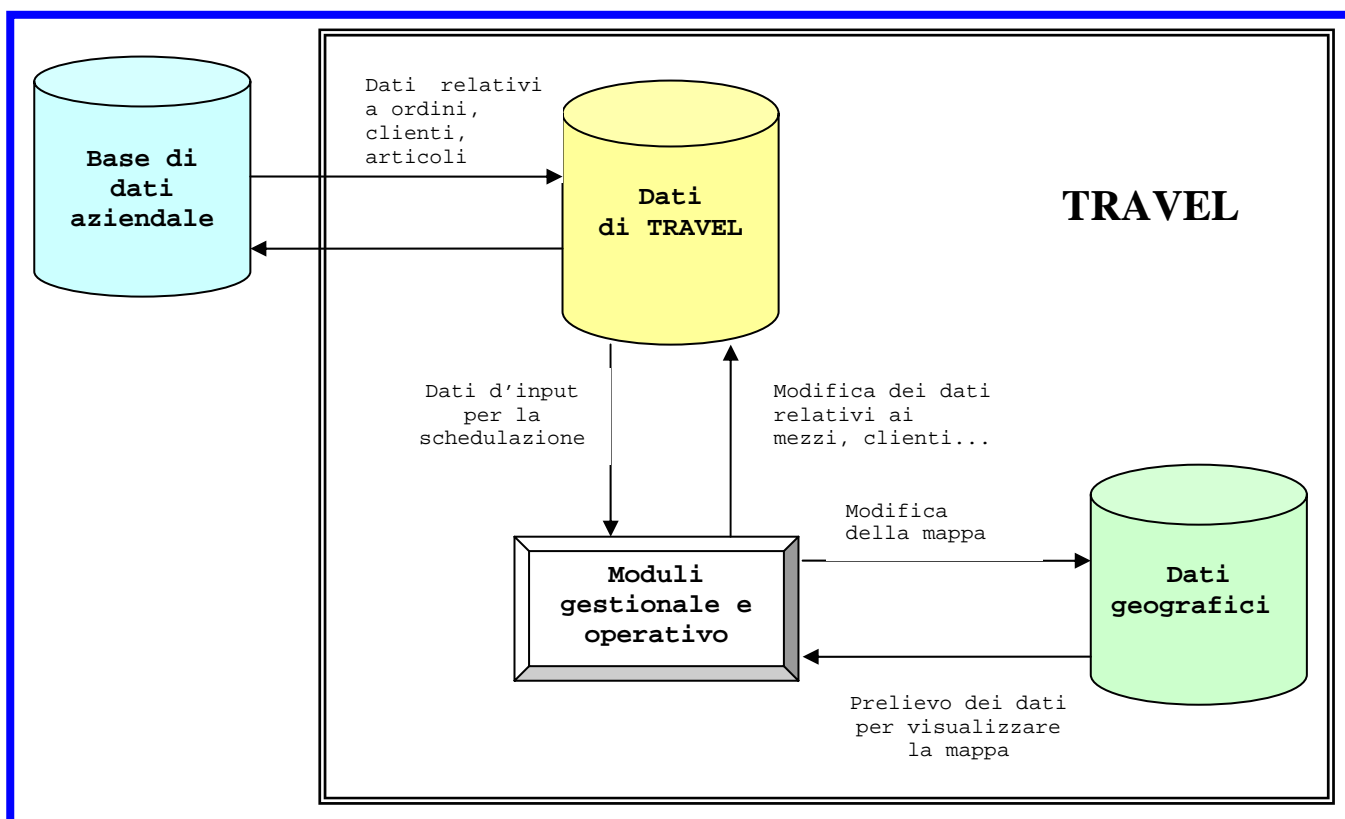


Figura 1.2 : TRAVEL: architettura del sistema

Le tabelle contenenti i dati per la rappresentazione della mappa sono tre (Figura 1.3):

- KNOTS: tabella delle località (nodi).
- NET: tabella delle strade (archi).
- NETTYPE: tabella dei tipi di strada.

Nella tabella KNOTS per ogni nodo sono indicati il suo identificativo, la sua importanza relativa (per esempio, se si tratta di un capoluogo di regione, di provincia o di un comune), dati di carattere generale (nome della località rappresentata, il codice di avviamento postale, la sigla della provincia e il nome della regione di appartenenza, il codice ISTAT associato alla località, il fatto se si tratta di un capoluogo) e informazioni legate alla mappa, come il valore dell'ascissa e dell'ordinata del nodo

e l'identificativo del suo settore di appartenenza; il settore è un'unità di suddivisione della mappa, atta a rendere più rapida l'identificazione di un nodo a seguito di un clic su un punto della mappa stessa.

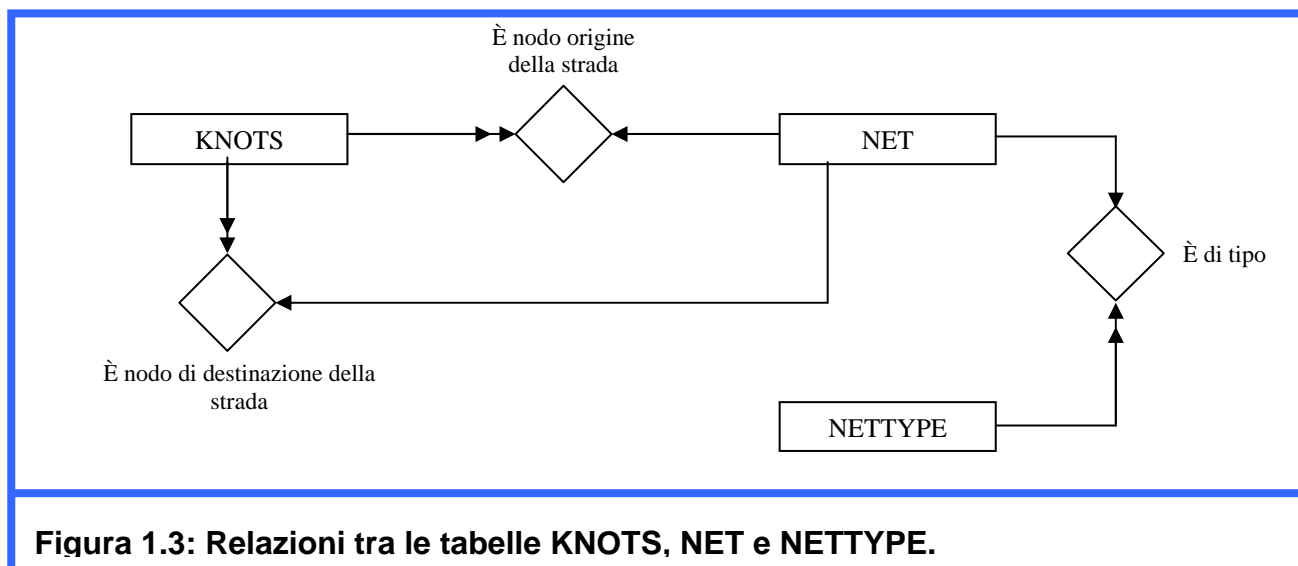


Figura 1.3: Relazioni tra le tabelle KNOTS, NET e NETTYPE.

Nella tabella NET, per ogni arco (strada), si includono il suo identificativo, il nodo di origine e di destinazione (con riferimento alla tabella KNOTS), la sua lunghezza (espressa in ettometri) e il suo tipo (con riferimento alla tabella NETTYPE).

Infine, in NETTYPE, per ogni tipologia di strada, si rappresentano le informazioni seguenti: il suo identificativo, la sua descrizione (per esempio, autostrada, urbana, secondaria...), la relativa velocità media di percorrenza e il colore con cui viene visualizzata sulla mappa.

1.4.2 Dati d'input per la schedulazione

In questo paragrafo si fornisce una descrizione delle tabelle principali da cui sono prelevati i dati in ingresso per la schedulazione. Tali tabelle sono raggruppabili in due insiemi: l'uno contenente i dati relativi a clienti, ordini e articoli; l'altro riguardante i mezzi.

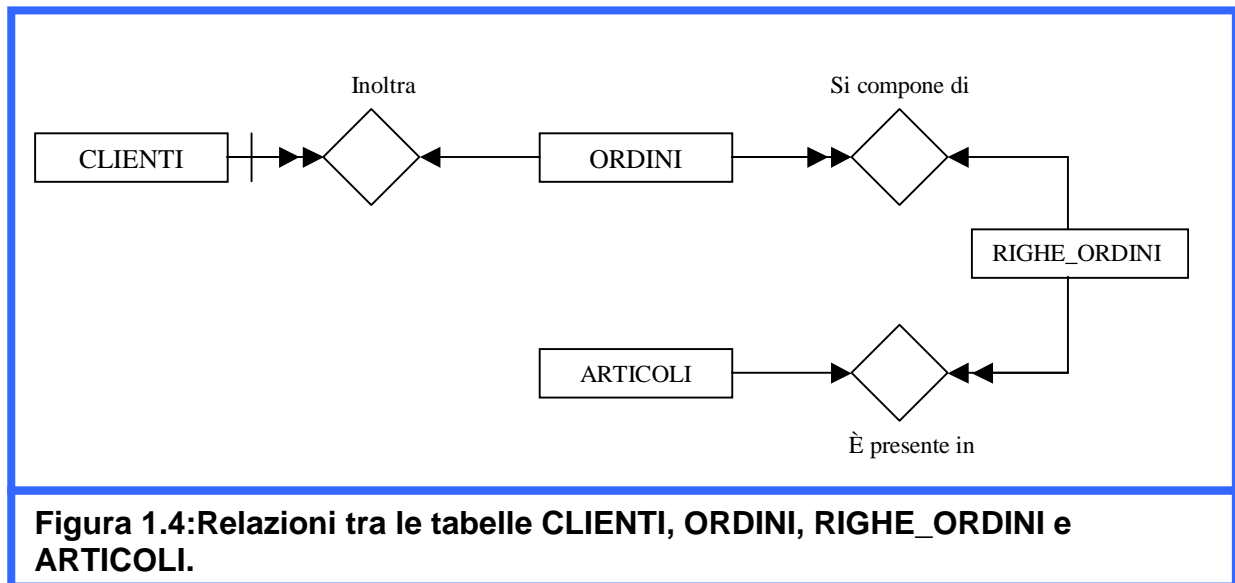
Il primo gruppo include le tabelle CLIENTI, ORDINI, RIGHE_ORDINI e ARTICOLI (Figura 1.4); il secondo le tabelle MEZZI, TIPI_MEZZI e TARIFFE (Figura 1.5).

1.4.2.1 I dati relativi a clienti, ordini e articoli

Ogni cliente è collocato in modo univoco all'interno della mappa. Le caratteristiche considerate per i clienti includono:

- la tipologia di impianto intesa come modalità di scarico consentita, altezza, larghezza e peso massimi consentiti per i mezzi che consegnano la merce;
- i dati anagrafici;

- le fasce orarie di ricevimento della merce per ogni giorno della settimana;
- il tempo di scarico inteso come tempo forfetario necessario a adempiere alle formalità richieste per la consegna della merce presso quel cliente;
- eventuali incompatibilità con i mezzi appartenenti alla flotta aziendali, riconducibili a fattori “umani” e non suscettibili ad alcuna schematizzazione.



Per quanto riguarda gli ordini, ogni ordine è relativo a un unico cliente e può essere costituito da n righe d'ordine; per ogni riga d'ordine si conoscono:

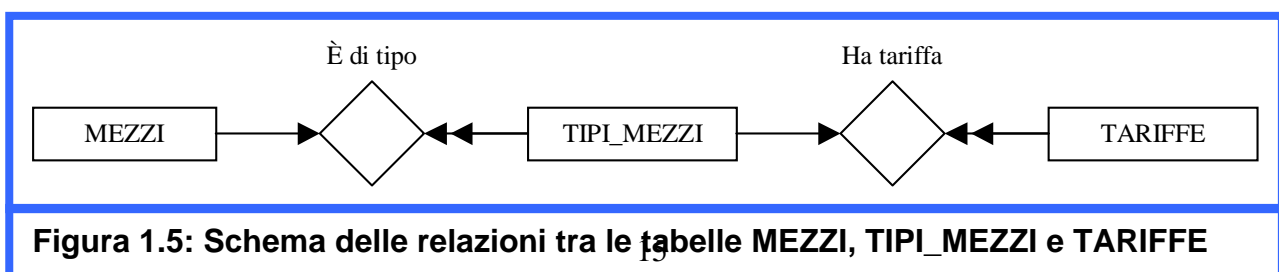
- il prodotto a cui si riferisce (unico per ogni riga);
- la quantità richiesta di tale prodotto in peso e in volume;
- la data di consegna.

Ogni riga d'ordine fa riferimento a un certo articolo del quale si prende in considerazione il peso specifico.

1.4.2.2 I dati relativi ai mezzi

I veicoli sono considerati per tipologia (tabella TIPI_MEZZI). Per ogni tipo di mezzo si prendono in esame le caratteristiche seguenti:

- portata massima;
- modalità di scarico (per esempio, pneumatico);
- altezza e larghezza;
- velocità media su ogni tipo di strada presente tra i dati geografici;
- presenza di un rimorchio;
- tipologia di tariffa applicata;
- tempo di carico/scarico (espresso in quintali per minuto).



Se il veicolo è impiegato per il trasporto alla rinfusa ed è pertanto dotato di scompartimenti, è necessario conoscere la portata e la capacità volumetrica di ogni scompartimento.

Ad ogni tipologia di veicolo possono appartenere n mezzi disponibili (tabella MEZZI). Per ciascuno di essi si prendono in considerazione le informazioni seguenti:

- la tipologia di appartenenza;
- l'autista eventualmente collegato al mezzo;
- il deposito di appartenenza;
- il numero massimo di viaggi effettuabili giornalmente;
- il calendario lavorativo;
- eventuali limitazioni geografiche.

Gli aspetti considerati per le tariffe sono i seguenti:

- La tipologia della merce trasportata (per esempio, in relazione a farinacei o mangimi zootecnici, potrebbero essere trasportati sfusi oppure in sacchi o scatole: la tariffa può variare conseguentemente).
- I limiti inferiore e superiore della fascia chilometrica in questione.
- L'importo della tariffa stessa.
- Ulteriori informazioni che incidono sull'ammontare della tariffa. A questo proposito, si può specificare se il pagamento è effettuato in base alla portata reale del mezzo o a quella massima, se il pagamento è in relazione al numero di quintali trasportati per chilometro percorso e se la tariffa include anche il viaggio di ritorno al deposito nella fascia chilometrica contemplata.

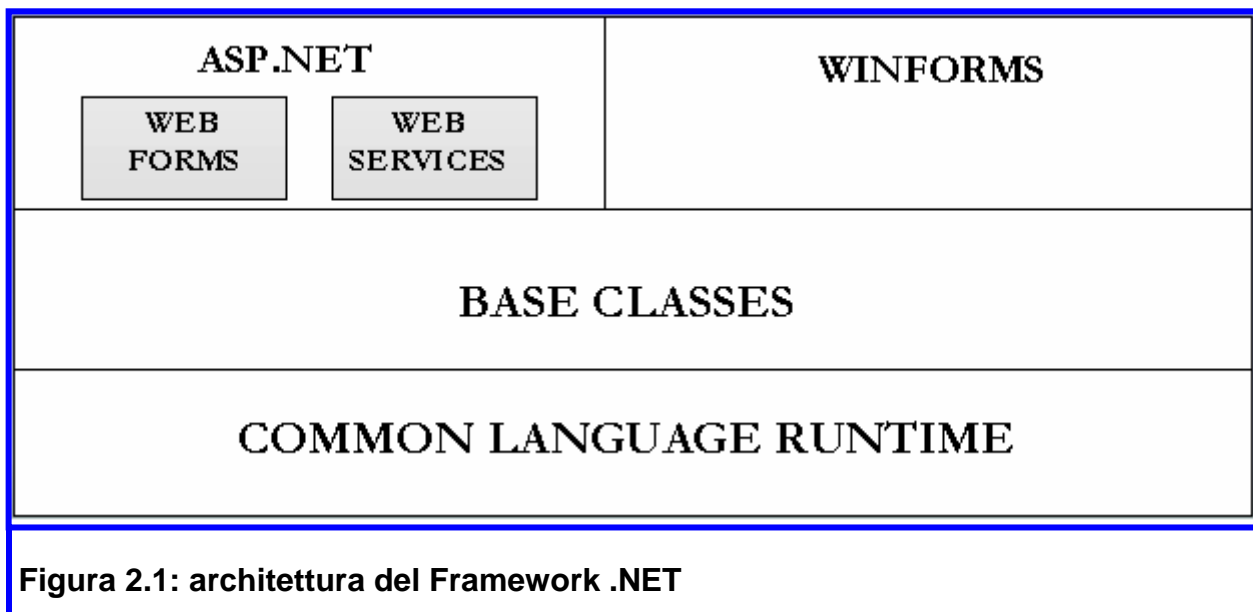
Il tipo di tariffario applicato comporta riduzioni più o meno significative nella minimizzazione dei costi.

CAPITOLO 2

La tecnologia impiegata

In questo capitolo vengono trattati gli strumenti utilizzati per lo sviluppo del progetto. La trattazione riguarda: ASP.NET per quanto riguarda le pagine Web, C# per quanto riguarda il linguaggio alla base delle pagine, attraverso cui vengono gestite e il framework .NET per quanto riguarda l'ambiente usato.

2.1 ASP.NET



2.1.1 Introduzione

Molti prodotti sono distribuiti come “un grande passo avanti” o come “rivoluzionari”, ma sono in effetti soltanto dei miglioramenti di versioni precedenti. ASP.NET non è come questi, ed è stato creato da nulla per garantire un mezzo ricco e flessibile per creare applicazioni Internet. Non solo offre nuove funzionalità, ma cambia anche lo stesso modo in cui bisogna pensare per costruire applicazioni per il Web.

Molti di questi cambiamenti derivano dal fatto che l'architettura di ASP.NET è molto più modularizzata ed è basata sul principio di componenti. Ogni pagina diventa un oggetto pienamente compilato, e trae vantaggio da tecniche tipiche di linguaggi orientati agli oggetti, dalla compilazione al volo subito prima dell'esecuzione (compilazione “just-in-time” o JIT) e dal caching dinamico.

Allo stesso tempo, la natura compatibile di ASP.NET significa che le pagine esistenti e le applicazioni sono ancora trattate alla vecchia maniera, quindi non c'è bisogno di riscriverle.

Uno dei maggiori obiettivi di ASP.NET è un immenso miglioramento del modo in cui le applicazioni possono essere installate, configurate e aggiornate. I componenti non devono più essere registrati nel server Web, e una intera applicazione può essere mossa da un server a un altro soltanto usando un comando di copiatura di file, FTP o applicazioni specializzate come FrontPage Server Extension.

2.1.2 Motivazioni sulla nuova versione

Come già detto, ASP.NET è stato creato da nulla. Ciò ha portato a una rottura nella compatibilità col vecchio ASP, ma ciò in fondo è stata una cosa buona, perché significa che così ASP.NET garantisce una piattaforma più potente per creare applicazioni, è ciò comporta molti più benefici.

Per quanto riguarda il problema della compatibilità, installare ASP.NET non interrompe il funzionamento delle applicazioni esistenti. Questo perché le pagine ASP esistenti sono ancora gestite dal solito meccanismo che le gestiva prima, e il nuovo Framework gestisce invece le nuove pagine ASP.NET, distinguendole dall'estensione del file (`.aspx` invece che `.asp`).

Tuttavia ASP.NET risolve alcuni dei problemi che invece esistevano nel vecchio ASP. Tra questi:

- ASP è un linguaggio di script, basato principalmente su VBScript o Jscript. Altri linguaggi sono disponibili se installiamo un interprete, ma sono ancora interpretati. Ciò porta a due svantaggi: la mancanza di una forte tipizzazione e la mancanza di un ambiente compilato. Il fatto che sia interpretato quindi porta inevitabilmente a problemi di performance.
- ASP non garantisce una struttura inerente per le applicazioni. Il problema era di mescolare i contenuti col codice, che è un problema se si ha un team di sviluppo misto in cui una parte lavora sui contenuti e l'altra sul codice. In più c'era il problema dei codici complessi, che portava alla creazione di file molto grandi. L'inclusione di file permetteva una certa struttura e un riutilizzo del codice, ma non era mai una grande soluzione.
- Bisognava scrivere codice ASP per fare molte cose: per esempio per controllare la correttezza di una stringa inserita in un campo bisognava scrivere codice per controllare che una stringa fosse stata scritta, poi per mantenere lo stato di un form bisognava scrivere codice e così via tutto necessitava di codice. Anche aggiungere controlli richiedeva scrivere HTML grezzo nella pagina.
- Il mondo delle compatibilità tra i browser è mutato in quello delle compatibilità tra i congegni da cui si richiedono le pagine Web. Se si vuole raggiungere il massimo numero di visitatori bisogna considerare anche questi congegni, e questo significa scrivere codice per individuare il congegno e restituire il contenuto appropriato.
- Anche la compatibilità con gli standard gioca una parte importante nello sviluppo del Web. XHTML sta generalmente diventando sempre più accettato, e anche l'XML è ora generalmente utilizzato, e parlando di congegni mobili può essere significativo anche il supporto per il WML. Ciò implica che per supportare questi standard ASP deve riuscire a lavorare con quelli esistenti, ma deve essere facilmente aggiornabile per riuscire a gestire gli standard futuri.

Quindi lo sviluppo di ASP.NET, risolvendo questi problemi (e non solo questi), ha avuto i seguenti obiettivi:

- Produrre codice pulito
- Migliorare la divisione del codice, la scalabilità, la sicurezza e l'affidabilità
- Garantire un miglior supporto per browser e congegni differenti
- Abilitare una nuova varietà di applicazioni Web

Alcuni di questi supporti non sono garantiti direttamente, ma attraverso il Common Language Runtime (CLR), che verrà discusso nel paragrafo dedicato al Framework .NET.

Adesso verranno esaminati i miglioramenti principali portati da ASP.NET nella creazione delle pagine Web.

2.1.3 Linguaggi multipli

ASP è stato limitato ai linguaggi di script, cioè VBScript e Jscript. Il Framework .NET supporta linguaggi multipli, per cui si può usare quello che sembra più comodo. Microsoft supporterà VB.NET, C# e Jscript (tutti compilati), ma ce ne sono molti altri da usare (come il C++). Visto che questa disponibilità di linguaggi è garantita dal Framework, non importa quale linguaggio un programmatore, o altri nel suo gruppo, usano.

Questo supporto a linguaggi multipli non è solo limitato a ciò che è disponibile, ma anche a ciò che è usato. È infatti possibile scrivere componenti in un linguaggio, e usarli (o riusarli) con un altro linguaggio.

2.1.4 Elaborazione lato Server

Uno dei grandi problemi di ASP è che le pagine erano definite da una grande funzione, che iniziava in cima alla pagina e finiva in fondo. I contenuti della pagina venivano resi nell'ordine della pagina, sia che fossero direttamente HTML oppure HTML generato da ASP. Tuttavia non c'è modo di indirizzare un controllo HTML ad eccezione di renderlo come parte di un flusso. Ogni cosa necessita di codice, e ciò include anche l'output degli elementi HTML.

ASP.NET risolve il problema introducendo un modello per i controlli dichiarativo e basato sul server. I controlli sono dichiarati sul server, possono essere programmati sul server ma accettano eventi anche dal client. Per rendere un normale controllo HTML in un controllo sul server dobbiamo soltanto aggiungere l'attributo "runat="server"".

Creare un controllo affinché venga fatto girare sul server ci permette di usare l'attributo ID per identificarlo direttamente. Ciò permette di creare codice più leggibile, e rende lo sviluppo di pagine molto più semplice.

2.1.5 I controlli delle Web Form

Convertire controlli HTML esistenti in quelli lato server è semplice, ma ci sono ancora alcuni problemi con questo approccio:

- **Consistenza.** Siamo ancora bloccati dalla natura ancora non intuitiva di alcuni controlli HTML (esempio: perché non fare un unico controllo che raggruppi i tag `INPUT` e `TEXTAREA`, la cui differenza sta solo nel numero di righe in cui si può scrivere?)
- **Esperienza dell'utente.** Non abbiamo la possibilità di scrivere facilmente pagine che vengano correttamente visualizzate da tutti i browser (siano a più alto o più basso livello), perché l'HTML non ha l'abilità di cambiare il contenuto in base al browser. Dobbiamo scrivere codice per far ciò.
- **Apparecchi.** Come scrivere siti per far fronte ai congegni oltre che ai browser? Telefoni WAP, PDA e anche i frigoriferi hanno browser oggi. Come nel caso dei browser, abbiamo bisogno di scrivere codice.

Per alleviare questi problemi è stato creato un set di server control, identificati dal prefisso `asp:`. ASP.NET affronta i problemi citati coi server control nei seguenti modi:

- Garantendo uno standard di nomi consistente. Per esempio, tutti i campi per l'immissione di testo sono gestiti dal controllo `TextBox`. Per le differenti modalità (linee multiple, password, etc etc) basta specificare alcuni attributi.
- Garantendo consistenti proprietà. Tutti i controlli lato server usano un gruppo consistente di proprietà, rendendole più facili da ricordare. Ad esempio, il campo `Text` di una `TextBox` è più intuitivo di un campo `Value`.
- Garantendo un consistente modello ad eventi. L'ASP tradizionale spesso aveva delle grandi quantità di codice per gestire l'invio dei dati, specialmente quando una pagina gestiva più comandi. Con ASP.NET si collegano i controlli alle procedure a eventi, dando al codice lato server più struttura.
- Emittendo HTML puro o con JavaScript lato client. Il controllo al server fa emettere HTML 3.2 di default, compatibile con molti browser. Ovviamente si può scegliere di emettere HTML 4.0 e DHTML, compatibile con una minor parte di browser, ma che garantisce una interfaccia più ricca. Tutti gli utenti vedranno comunque il contenuto HTML e non il controllo al server.
- Emittendo codice specifico per i diversi apparecchi. Certi controlli restituiscono HTML quando richiesti da un browser ma WML quando richiesti da un cellulare WAP. Il controllo gestisce l'individuazione del congegno e genera il linguaggio adatto.

Per quanto riguarda la gestione degli eventi, un controllo dichiara quali sono gli eventi che può generare. Nella pagina ASP.NET comparirà un attributo che lega l'evento al metodo che lo gestisce. La procedura riceve due parametri: il controllo che ha generato l'evento e altri eventuali argomenti utili alla procedura stessa. All'interno della procedura possiamo accedere a qualsiasi altro controllo, incluso i dati inviati durante un postback.

2.1.6 La gerarchia dei Server Control

I server control sono logicamente divisi in un gruppo di famiglie:

- **Server Control HTML**, che sono l'equivalente lato server degli elementi HTML.

- **Web Form Control**, che descrivono da vicino gli elementi HTML individuali.
- **Controlli a lista**, che descrivono gruppi di elementi HTML per produrre tabelle (grid) o controlli simili.
- **Controlli ricchi**, che producono contenuti ricchi e incapsulano complesse funzionalità, e restituiscono HTML puro o con script. Un buon esempio di questa famiglia di controlli è il Calendario, che permette all'utente di avere un calendario attraverso un'unica linea di codice.
- **Controlli di convalida (validator)**, ovvero controlli invisibili, ma che permettono un facile utilizzo di mezzi di convalida sia lato server (attraverso il linguaggio scelto) sia lato client (attraverso uno script).

2.1.7 I Web Control

Vi sono una serie di Web Control già utilizzabili in ASP.NET, in più, come già detto, c'è la possibilità di crearne altri per venire incontro alle proprie esigenze.

I Web Control più comuni sono:

- **TextBox**: i classici campi di testo compilabili, utili per mandare stringhe al server. Sono rappresentati dal controllo `input` o `textarea` dell'HTML.
- **CheckBox**: le caselle da contrassegnare con una X. Hanno lo stesso nome anche come tag HTML.
- **Validator**: servono per convalidare i dati inseriti dagli utenti, e generalmente vengono resi in JavaScript per lasciare la convalida sul lato client. I più utilizzati sono il `RequiredFieldValidator`, che controlla che un campo non sia lasciato vuoto, e il `RegularExpressionValidator`, che prende come attributo una espressione regolare e controlla che il valore inserito sia verificato da una espressione regolare.
- **DropDownList**: le caselle combinate con elenco a discesa, ovvero quelle caselle che, se selezionate, permettono di visualizzare un elenco di opzioni da cui scegliere. L'elenco può essere inserito a mano, tratto da un database oppure una combinazione delle due.
- **DataGrid**: è un tipo di controllo che genera una tabella compilata con i dati presi da un database. Bisogna indicare, tra le proprietà, una collezione di colonne (che possono essere formate da campi di testo, pulsanti o possono essere personalizzate), ognuna delle quali deve avere specificato con quale campo della tabella del database verrà compilata.
- **Calendar**: questo è un controllo molto complesso che genera un calendario (il cui aspetto può essere personalizzato a piacere) i cui pulsanti possono essere collegati a delle funzioni JavaScript (gran parte del controllo è formato da codice JavaScript).

2.1.8 La separazione tra codice e contenuto

Con ASP.NET si è giunti a una separazione del codice dai contenuti. Questo viene fatto in due modi: il codice può essere mantenuto ancora all'interno della pagina ASP.NET, ma non è mescolato all'HTML. È facile separare il codice e i contenuti in due sezioni, e questa separazione è possibile grazie alla nuova architettura basata sui server control, che permette l'accesso ai controlli HTML dal codice sul server.

Il secondo modo per separare codice da contenuto è il modello "code-behind", dove il codice è completamente rimosso in un file separato. Una speciale direttiva viene aggiunta per comunicare al Common Language Runtime che la pagina corrente eredita il suo codice dal file nominato.

Il codice rimane lo stesso e, visto che stiamo lavorando in un ambiente compilato, non c'è nemmeno perdita di performance.

2.3 Il linguaggio C#

2.3.1 Introduzione e visione di insieme

Nelle scorse due decadi C e C++ sono stati i due linguaggi più usati per sviluppare software commerciale. Mentre entrambi i linguaggi forniscono allo sviluppatore un tremendo numero di controlli diversificati, questa flessibilità viene raggiunta a costo della produttività. Paragonato a una applicazione scritta in VB una applicazione in C o C++ impiega molto più tempo per essere scritta. A causa della complessità e dei lunghi tempi di sviluppo associati a questi linguaggi, molti sviluppatori C e C++ hanno cercato un linguaggio che offrisse un maggior bilancio tra potenza e produttività.

Ci sono oggi linguaggi che incrementano la produttività a sfavore della flessibilità che spesso richiedono gli sviluppatori C e C++. Queste soluzioni limitano molto lo sviluppatore (per esempio omettendo un meccanismo per un controllo sul codice a basso livello) e garantiscono le capacità a “minimo comune denominatore”. Non interagiscono facilmente con sistemi preesistenti e spesso non si legano bene con la pratica del moderno sviluppo Web.

La soluzione ideale per i programmatori C e C++ dovrebbe essere rapido sviluppo combinato con le potenzialità di accesso a tutte le funzioni della piattaforma sottostante. Vogliono un ambiente che sia perfettamente sincronizzato con gli standard emergenti per il Web e che offra una facile integrazione con le applicazioni esistenti. In più gli sviluppatori C e C++ vorrebbero avere la possibilità di scrivere codice di basso livello qualora ne abbiano bisogno.

2.3.2 La Microsoft introduce il C#

La soluzione della Microsoft a questo problema è un linguaggio chiamato C# (pronunciato “C sharp”), moderno, orientato agli oggetti e che permette ai programmatori di costruire una vasta gamma di applicazioni per la nuova piattaforma .NET, che offre strumenti e servizi che sfruttano pienamente sia la potenza di calcolo che le comunicazioni.

Grazie al suo elegante design orientato agli oggetti, il C# è un'ottima scelta per l'architettura di una vasta gamma di componenti, da oggetti per il commercio a alto livello fino a applicazioni. Usando semplici costrutti del linguaggio C#, queste componenti possono essere convertite in servizi Web XML, permettendo l'accesso via Internet attraverso qualunque linguaggio su ogni sistema operativo.

Più che ogni cosa, C# è costruito per portare uno sviluppo rapido ai programmatori C e C++. A causa della sua eredità, C# ha una grande fedeltà con C e C++. Sviluppatori che conoscono questi linguaggi possono essere presto produttivi con C#.

2.3.3 Produttività e sicurezza

La nuova Web-Economy, dove i concorrenti sono lontani appena un click, sta forzando il commercio a rispondere alle minacce competitive più velocemente di quanto lo facesse prima. Gli sviluppatori sono chiamati ad accorciare i tempi previsti e a produrre più versioni incrementali di un programma, anzi che una unica versione monumentale.

C# è stato creato con queste considerazioni in testa. Il linguaggio è stato realizzato per aiutare gli sviluppatori a fare più con meno linee di codice e minori opportunità di errore.

2.3.3.1 Abbracciare gli standard emergenti della programmazione Web

Il nuovo modello per lo sviluppo di applicazioni significa che più e più soluzioni richiedono l'uso degli standard emergenti come l'Hypertext Markup Language (HTML), l'Extensible Markup Language (XML) e il SOAP. I tool di sviluppo esistenti furono creati prima di Internet o quando il Web come noi lo conosciamo era nella sua infanzia. Come risultato di ciò, spesso non offrono il meglio per lavorare con le nuove tecnologie Web.

I programmatori C# possono far leva su un framework esteso per scrivere applicazioni sulla piattaforma .NET. C# include il supporto per convertire ogni componente in un servizio Web XML che può essere invocato attraverso Internet da qualunque applicazione eseguita su qualunque piattaforma.

Ancora meglio, il framework dei servizi Web XML può far apparire al programmatore ogni servizio XML esistente come un nativo oggetto C# , quindi può permettere agli sviluppatori di far leva sulle conoscenze che già hanno sui linguaggi orientati agli oggetti per gestire i servizi Web XML.

Ci sono molte sottili funzionalità che rendono il C# un grande mezzo per la programmazione in Internet. Per esempio, XML sta emergendo come standard per passare dati strutturati su Internet. Queste unità di dati sono spesso molto piccole. Per aumentare la performance, C# permette di mappare i dati XML direttamente in un tipo di dato strutturato invece che in una classe. Questa è una via molto più efficiente per gestire piccole quantità di dati.

2.3.3.2 Eliminare costosi errori di programmazione

Anche il più esperto programmatore C++ può fare il più semplice degli errori (ad esempio dimenticare di inizializzare una variabile) e spesso questi semplici errori possono creare problemi imprevedibili che possono rimanere nascosti per lunghi periodi di tempo. Una volta che un programma è in produzione, può essere molto dispendioso eliminare anche il più semplice errore.

Il design moderno del C# elimina i più comuni errori della programmazione C++. Per esempio:

- Il Garbage Collector solleva il programmatore dal carico di una gestione manuale della memoria.
- Le variabili in C# sono automaticamente inizializzate dall'ambiente.
- Le variabili sono sicure per quanto riguarda i tipi.

Il risultato finale è un linguaggio che rende molto più facile per gli sviluppatori il compito di scrivere e di mantenere programmi che risolvono complessi problemi.

2.3.3.3 Ridurre i costi di sviluppo in corso col supporto per la versione

Aggiornare componenti del software è un compito incline agli errori. La revisione fatta al codice può cambiare non intenzionalmente il significato di un programma esistente. Per assistere lo sviluppatore con questo problema, C# include il supporto per la versione nel linguaggio. Per esempio, i metodi devono essere sovrascritti esplicitamente, e non può succedere che ciò avvenga inavvertitamente come in C++ o Java. Ciò aiuta a prevenire gli errori di codice e a preservare la flessibilità della versione. Una funzione correlata è il supporto nativo per le interfacce e per l'eredità da una interfaccia. Queste funzionalità permettono a complessi framework di essere sviluppati e di evolversi nel tempo.

Messe assieme, queste funzionalità rendono il processo di sviluppo di versioni seguenti di un progetto più robusto e in più riducono il costo totale di sviluppo per le versioni successive.

2.3.4 Potenza, espressività e flessibilità

2.3.4.1 Interazione estesa

L'ambiente gestito e controllato sui tipi è appropriato per molte applicazioni di impresa. Ma l'esperienza del mondo reale mostra che alcune applicazioni continuano a richiedere codice "nativo", sia per ragioni di performance oppure per interagire con le esistenti interfacce per la programmazione di applicazioni (API). Tali scenari possono forzare gli sviluppatori a usare il C++ anche quando preferirebbero usare un ambiente di sviluppo più produttivo.

Il C# affronta questi problemi attraverso:

- L'inclusione del supporto nativo per le API basate sul Component Object Model (COM) e di Windows
- Il permesso limitato a usare puntatori nativi.

Con C#, ogni oggetto è automaticamente un oggetto COM. Gli sviluppatori non devono più implementare esplicitamente la IUnknown o altre interfacce COM. Invece, queste funzionalità sono già pronte. Similmente, i programmi C# possono usare in maniera nativa gli oggetti COM esistenti, senza preoccuparsi di quale linguaggio venga usato per scriverli.

Per gli sviluppatori che lo richiedono, C# include funzioni speciali che permettono ad un programma di chiamare API native. Dentro a uno speciale blocco di codice, gli sviluppatori hanno

il permesso di usare puntatori e funzioni tradizionali C/C++ come i puntatori aritmetici e le funzioni per gestire manualmente la memoria. Questi sono vantaggi enormi rispetto agli altri ambienti. Significa che i programmatori C# possono sviluppare sopra il loro vecchio codice C e C++, invece che scartarlo.

In entrambi i casi (supporto a COM e accesso alle native API) lo scopo è offrire allo sviluppatore il potere essenziale e il controllo senza dovergli fare abbandonare l'ambiente C#

2.2 Il Framework .NET

2.2.1 Cosa è .NET?

Come molte persone possiamo essere un po' confusi circa la capacità di .NET. Cosa è precisamente .NET? Quale tecnologie e prodotti ne fanno parte? .NET è un rimpiazzo di COM o è costruito a partire da COM?

Non esiste una risposta semplice per dire cosa sia .NET. La Microsoft ha confuso molto i confini di .NET e ha rimarcato molti dei loro prodotti per server (come SQL Server 2000 e BizTalk Server 2000) come “.NET Enterprise Servers”.

2.2.2 Le parti di .NET

Un modo per eliminare tutta questa confusione è quello di dividere .NET in 3 pezzi principali:

- **Il punto di vista di .NET** – l'idea fondamentale è che tutti i dispositivi un giorno saranno connessi assieme da un network a banda larga (che sarà Internet), e che il software sarà un servizio distribuito in questo network.
- **Il Framework .NET** – nuove tecnologie come ASP.NET che rendono .NET più di un semplice punto di vista, garantendo tecnologie e servizi concreti così che gli sviluppatori possano costruire applicazioni per venire incontro alle necessità degli utenti connessi a Internet oggi.
- **Il .NET Enterprise Servers** – prodotti server come SQL 2000 e BizTalk 2000 che sono usati dalle applicazioni del Framework .NET, ma non sono ancora scritte usando il Framework .NET. Tutte le future versioni di questi prodotti server supporteranno .NET, ma non saranno necessariamente riscritte usando .NET.

Per gli sviluppatori un altro importante pezzo della piattaforma .NET sono i tool di sviluppo. La Microsoft ha creato un grande aggiornamento del Visual Studio chiamato appunto Visual Studio .NET. Tuttavia si possono ancora sviluppare applicazioni .NET col Notepad, ad esempio, o con un altro qualunque editor.

2.2.3 Vantaggi e svantaggi della piattaforma .NET

Per programmare applicazioni per Windows oggi giorno ci sono miriadi di linguaggi o di tecnologie utilizzabili. In funzione del linguaggio scelto sono disponibili tecnologie che in genere sono tra loro molto differenti, e sono spesso molto restrittive. Per esempio, un programmatore C/C++ che deve creare una applicazione GUI (Graphical User Interface) può usare le MFC (Microsoft Foundation

Classes), le WTL (Windows Template Library) oppure le API (Application Program Interface) WIN32 a un più basso livello. Un programmatore VB dovrà utilizzare i pacchetti VB Forms.

I problemi con questo approccio sono:

- Microsoft spende più tempo a sviluppare più tecnologie che non a migliorarne una sola
- La disponibilità di più tecnologie con la stessa funzione confonde la gente
- Sviluppatori che conoscono più linguaggi devono imparare più tecnologie per avere gli stessi risultati
- Le compagnie devono investire in predominanza su un linguaggio, visto che un insegnamento incrociato può essere dispendioso per tempo e denaro
- Non tutti i linguaggio espongono necessariamente le stesse funzionalità, o sono tutti produttivi allo stesso modo

Questi problemi spesso portano confusione, e sono molte le compagnie fallite per aver scelto un linguaggio/tecnologia sbagliato. Il Framework .NET aiuta a risolvere questi problemi.

Con .NET c'è ora un solo chiaro modo orientato agli oggetti per accedere alle funzionalità del Framework e costruire applicazioni. Tutte le funzionalità migliori e più comunemente utilizzate sono state riunite assieme in un singolo Framework.

Tuttavia ci sono anche degli aspetti negativi che riguardano la performance dei programmi: se si stanno creando applicazioni che richiedono una risposta assolutamente veloce, tipo applicazioni real-time o SQL Server, .NET potrebbe non fare al caso nostro. Sebbene .NET dia enormi benefici agli sviluppatori, semplicemente non ha la performance di una applicazione C/C++ ben scritta, nonostante alcuni aspetti di una applicazione .NET (come l'allocazione di memoria) sono più veloci che in C/C++.

2.2.4 Sulle funzionalità dei linguaggi

In .NET tutti i linguaggi di programmazione sono "cittadini di prima classe". Questo significa che si possono implementare soluzioni in un linguaggio di programmazione in cui un gruppo di sviluppatori sono produttivi, senza nessuna penalità.

Sono quattro i linguaggi supportati fin dall'inizio da Microsoft: VB.NET, C#, Jscript.NET e Visual C++.NET. Non ci sono differenze significative tra questi linguaggi, per cui la scelta è ancora lasciata a preferenze personali o benefici delle compagnie. Una cosa da notare è che la performance di alcuni linguaggi può essere migliore marginalmente (circa il 5%) rispetto ad altri.

2.2.5 Il "cuore" di .NET: il Common Language Runtime (CLR)

Il CLR è una delle funzionalità più radicali di .NET. I moderni linguaggi di programmazione come VC++ e VB hanno sempre avuto il runtime.

Le regole del runtime di un linguaggio cambiano a seconda del linguaggio stesso: possono eseguire il codice (come nel caso di Java), o in caso di linguaggi compilati (come C/C++), il runtime

garantisce funzionalità comuni usate dalle applicazioni. Alcune di queste funzionalità possono essere usate direttamente da una applicazione, tipo cercare una sequenza di caratteri in una stringa, o indirettamente, tipo un compilatore che aggiunge codice durante il processo di compilazione per gestire situazioni di errore e eccezioni.

Il CLR è un runtime per tutti i linguaggi .NET. È responsabile di eseguire e gestire tutto il codice scritto in qualsiasi linguaggio che giri sulla piattaforma .NET.

Il ruolo del CLR è in qualche modo simile alla Java Virtual Machine (JVM) di Sun: è responsabile dell'esecuzione del codice sviluppato usando linguaggi .NET. Tuttavia, il punto critico che differenzia il CLR è che compila tutti i codici.

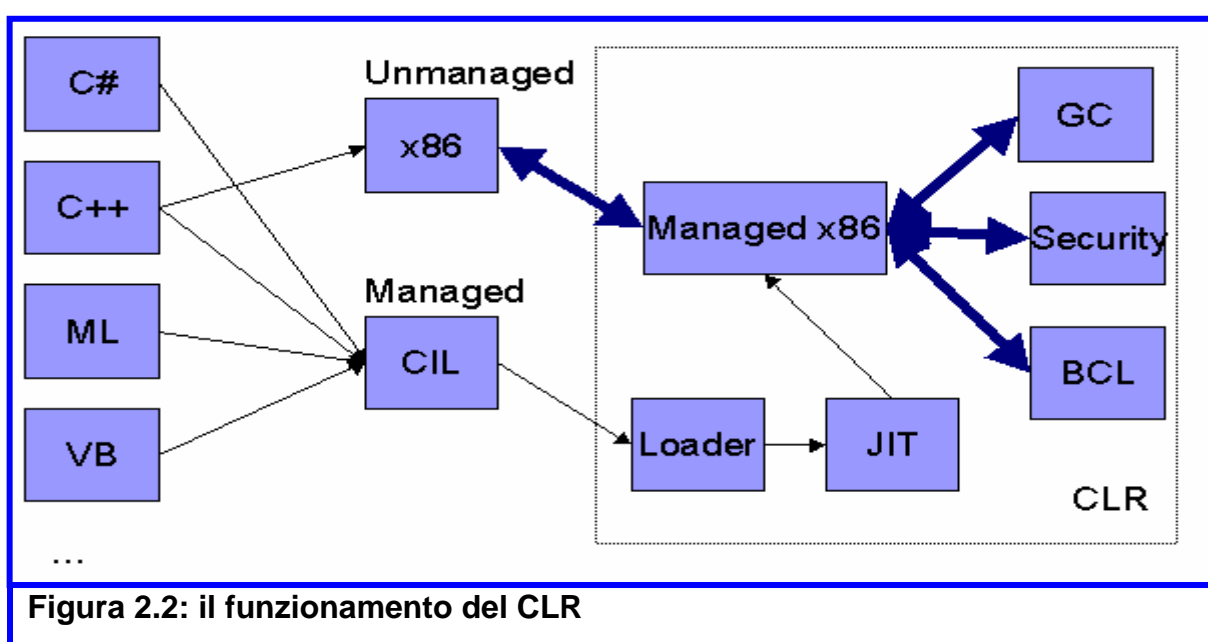


Figura 2.2: il funzionamento del CLR

I compilatori .NET emettono un Linguaggio Intermedio (IL) anziché codice macchina, il IL è compilato al momento della richiesta (“just-in-time”, JIT) prima che il codice sia eseguito. Il Linguaggio Intermedio non è interpretato, e non è un bytecode come il p-code di VB o il bytecode di Java, ma è un linguaggio vero e proprio, compilato, convertito in codice macchina e quindi eseguito. Il risultato è una applicazione di .NET che viene eseguita sul CLR e ha una performance eccezionalmente buona.

Per rendere completo il Linguaggio Intermedio, i compilatori che hanno come obiettivo il CLR emettono anche metadati ricchi che descrivono i tipi contenuti con una DLL o un EXE (simili alle librerie di tipi COM, ma più ricche) e informazioni su versione/dipendenze. Questi metadati permettono al CLR di risolvere intelligentemente i riferimenti tra differenti file di applicazioni a runtime, e oltretutto rimuovere le dipendenze dal registro di sistema.

2.2.6 I servizi del CLR

Il CLR garantisce molti servizi di base per le applicazioni quali il Garbage Collector, la verifica del codice e la sicurezza sull'accesso del codice. Il CLR può fornire questi servizi grazie al modo in cui gestisce l'esecuzione delle applicazioni e il fatto che può avere informazioni su tutti i tipi usati ricavandole dai ricchi metadati prodotti dai compilatori.

Il Garbage Collector è una funzionalità del CLR che automaticamente gestisce la memoria per conto di una applicazione. Si creano e usano oggetti, ma non si distruggono esplicitamente. Il CLR rilascia gli oggetti quando non sono più referenziati e in uso. Ciò affievolisce il problema della mancanza di memoria nelle applicazioni. Il momento in cui viene rilasciata la memoria è non deterministico: un effetto di questa funzionalità è che uno sviluppatore può pensare che un oggetto venga eliminato liberando memoria non appena diventa inutile.

La verifica sul codice è un processo che rende sicuro tutto il codice prima di eseguirlo. Questa funzione richiede sicurezza sui tipi e in più evita che il codice esegua operazioni illegali come ad esempio accedere a blocchi di memoria invalidi. Grazie a questo servizio non deve essere possibile scrivere codice che porti a un blocco nella applicazione. Se il codice fa qualcosa di male, il CLR lancia una eccezione prima che possa essere fatto alcun danno. Tali eccezioni possono essere catturate e gestite dall'applicazione.

La sicurezza sull'accesso del codice permette di garantire o negare al codice stesso i permessi per compiere determinate azioni, in funzione della configurazione della sicurezza per una macchina data, dell'origine del codice e dei metadati associati ai tipi che l'applicazione sta cercando di usare. Lo scopo principale di questa funzione è di proteggere l'utente da codice malevolo che prova a accedere a altro codice residente nella macchina.

2.2.7 Il Sistema di Tipi Comune (Common Type System – CTS)

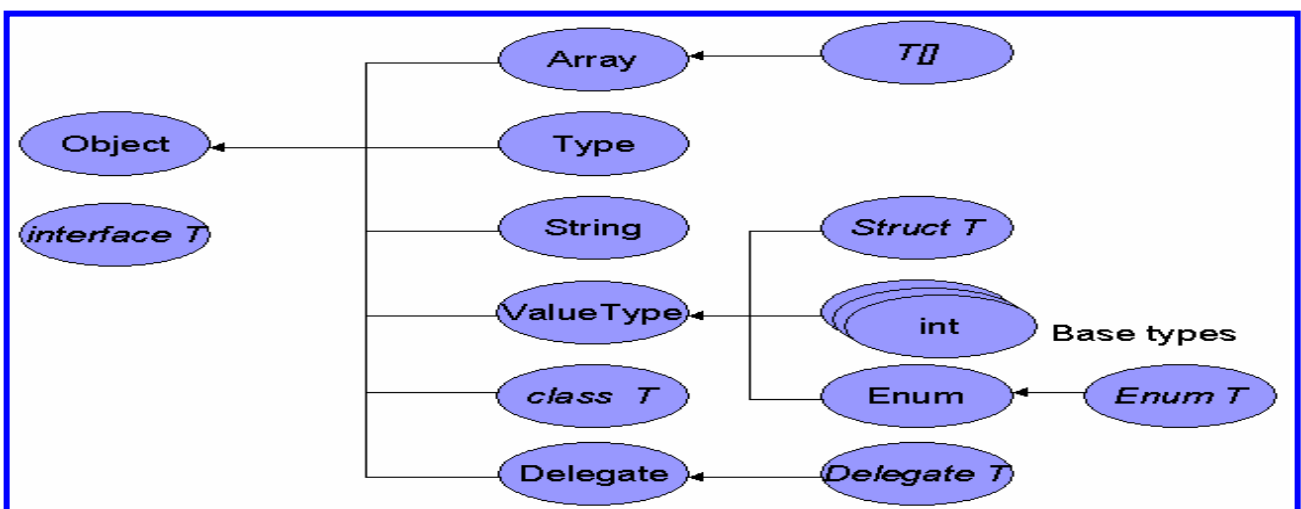


Figura 2.3: il Sistema di Tipi Comune del CLR

Per avere una interoperabilità regolare tra i vari linguaggi, il CLR ha un sistema di tipi comune. Linguaggi come VB hanno tipi primitivi come Integer, String e Double. C++ ha tipi come long, ulong e char*. Tuttavia questi tipi spesso non sono compatibili. Per far sì che il CLR possa integrare linguaggi incrociati così uniformemente, tutti questi linguaggi devono usare uno stesso sistema di tipi comune.

Linguaggi differenti usano differenti parole chiave per esporre questi tipi. Ciò rende il lavoro sui vari linguaggi più naturale e intuitivo, senza compromettere lo scopo del CLR. Si può comunque dichiarare il nome del tipo nativo del CLR in qualunque linguaggio, ma generalmente questa cosa non viene fatta a meno che il linguaggio non abbia la propria corrispondenza tra tipi. Questa è un'altra grande funzionalità del CLR. Se un linguaggio non dispone di una determinata funzione, si potrà solitamente trovare una classe del framework .NET che lo farà. Tutti i tipi derivano da System.Object. Questa classe ha quattro metodi, tutti disponibili per ogni tipo: Equals (che confronta due oggetti per uguaglianza; molte classi del CLR sovrascrivono questo metodo per darne una implementazione più adatta), GetHashCode (restituisce un codice hash per l'oggetto, e tale codice può essere utilizzato come identità unica per l'oggetto), GetType (restituisce un oggetto Type che può essere usato per controllare i metodi, le proprietà e gli eventi di un tipo; questa funzione nel CLR è chiamata **Reflection**) e ToString (restituisce una rappresentazione del tipo attraverso una stringa, e viene sovrascritto per molti tipi del CLR).

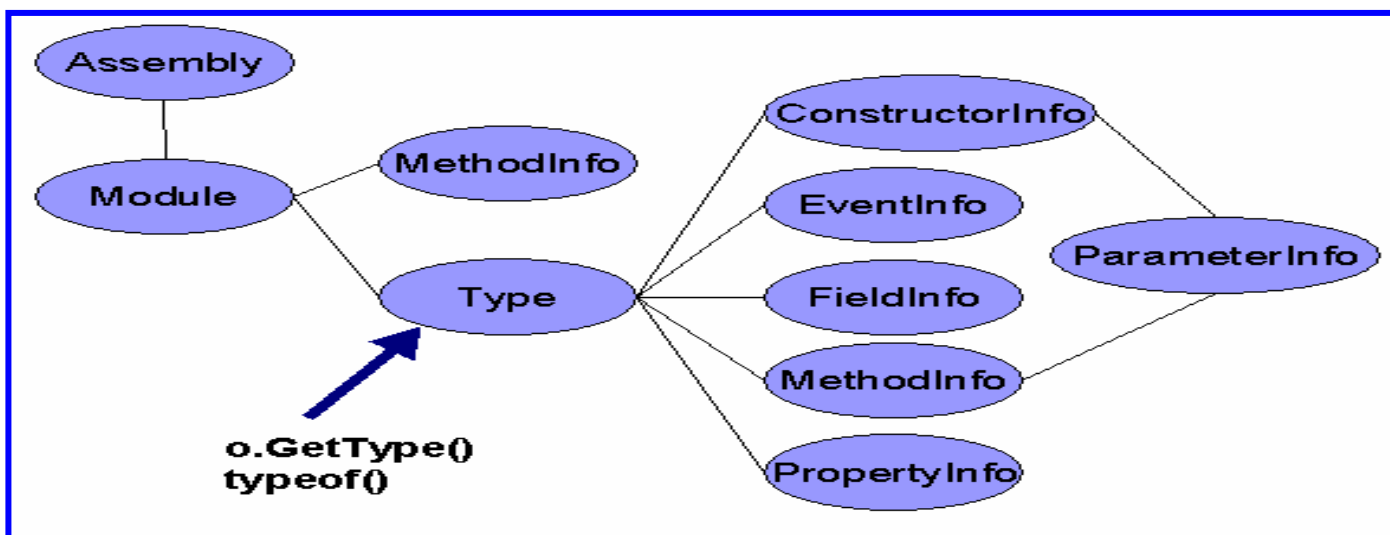


Figura 2.4: Struttura della Reflection

2.2.8 Il Common Language Specification (CLS)

Alcuni linguaggi possono richiedere tipi che altri linguaggi non supportano. Per garantire ciò ci sono tipi del CLR addizionali, che supportano le funzionalità richieste per linguaggi specifici, come

ad esempio il C#. Tuttavia, un'altra diffida nel condividere le classi del CLR create con differenti linguaggi significa che c'è potenzialmente la possibilità che una classe non possa essere usata in certi linguaggi perché i suoi tipi non sono compresi. Comunque il problema non sussiste se tali tipi vengono usati nelle definizioni di membri non pubbliche. Per aiutare ad assicurare che i tipi creati da linguaggi differenti siano compatibili, esiste un gruppo comune di tipi base che assicurano l'interoperabilità tra i linguaggi. Questo è parte di quello che viene chiamato il Common Language Specification (CLS). Molti compilatori del CLR hanno opzioni che 'avvisano' il programmatore, in caso che venga usato un tipo non presente nel CLS.

2.2.9 Gli Assembly

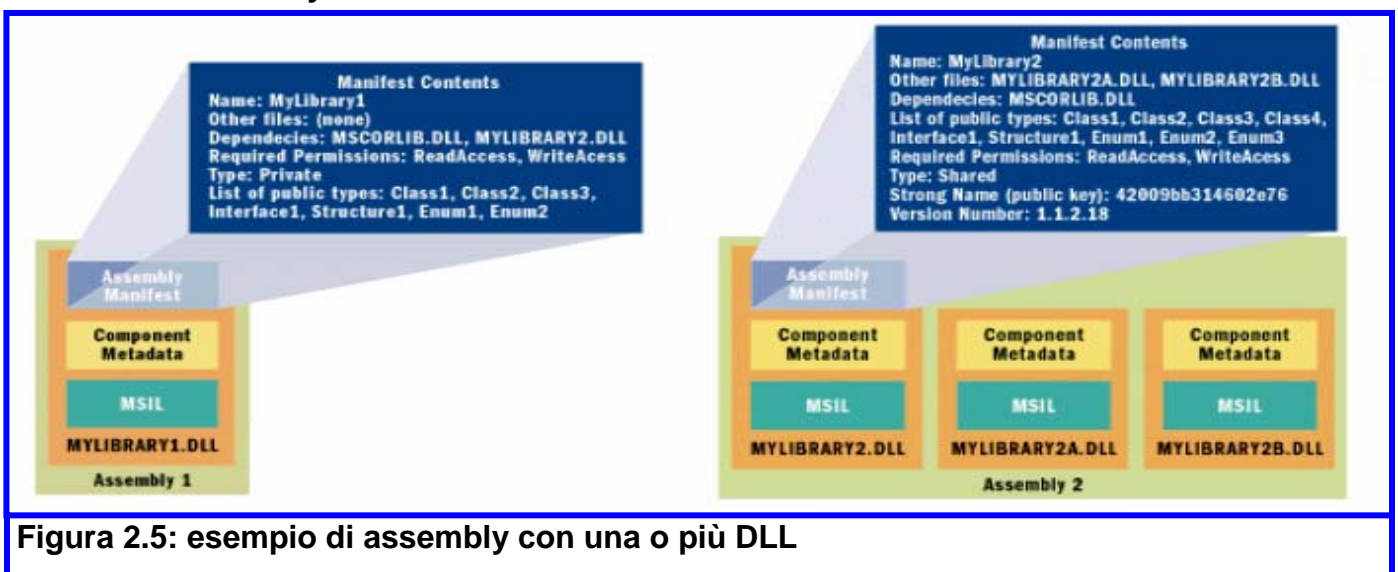


Figura 2.5: esempio di assembly con una o più DLL

Prima di .NET una applicazione poteva potenzialmente rendere non funzionabile un'altra applicazione sovrascrivendo accidentalmente una DLL di sistema, o rimuovendo una componente installata. Per evitare questi problemi il CLR usa gli assembly.

Un assembly è una collezione di uno o più file, con uno di questi file (sia un DLL o un EXE) contenente alcuni metadati speciali chiamati **manifesto dell'assembly**. Questo manifesto definisce quali sono le richieste di versione per l'assembly, chi ne è l'autore, quali permessi di sicurezza necessita per essere eseguito e quali file ne fanno parte.

Un assembly è creato di default quando si costruisce una DLL. Si possono esaminare i dettagli del manifesto nel programma attraverso le classi locate nel namespace `System.Reflection`.

Si può pensare che una DLL e un assembly siano due cose molto simili. Molti sviluppatori probabilmente li useranno allo stesso modo, ma in casi speciali si possono usare le funzionalità più avanzate per creare assembly che raggruppano più DLL, un singolo EXE, e che contengono o legano assieme uno o più file di risorse.

Tutti i file interni a un assembly sono chiamati **moduli**. Un namespace può aggregare uno o più moduli. La struttura fisica di un assembly non ha impatto o relazione sulla struttura del namespace. La ragione per creare assembly con più file è varia. Può essere appropriato agire in tal modo se si sta lavorando in un grande gruppo e si vuole inserire i moduli dei singoli sviluppatori in un unico assembly la cui versione sia controllata. O, in modo più pratico, si può volere gestire il consumo di memoria. Se si è a conoscenza che alcune classi non sono necessarie con frequenza per l'attività dell'applicazione, metterle in un file separato significa che il CLR può ritardare il caricamento del file finché i tipi in esso contenuti non vengono richiesti.

Capitolo 3

Il lavoro svolto

Il progetto che ho svolto ha riguardato il porting dell'applicazione TRAVEL sulla rete, in modo da creare una applicazione ASP e fornire uno strumento utilizzabile da tutte quelle aziende che per loro tipologia hanno necessità di spedire merce trasportata su gomma.

Il lavoro è stato diviso in due parti: la prima riguardante la costruzione di alcune delle pagine principali utili per l'applicazione, la seconda riguardante la creazione di alcuni web controls, utili per rendere più modulari le pagine create.

Non è stato descritto invece il lavoro eseguito per creare le tabelle del database e le Stored Procedure per interagire con queste, in quanto si è trattato di un lavoro ripetitivo e molto poco creativo, in cui le differenze tra le varie tabelle (o le Stored Procedure ad esse collegate) consistevano soltanto nel nome dei campi che le costituivano.

3.1 Scopo del progetto

Il progetto su cui ho lavorato fa parte di un progetto più grande della azienda Optisoft chiamato "*netT*".

L'obiettivo di tale progetto è quello di fornire uno strumento in rete che permetta un utilizzo ottimizzato delle risorse disponibili da parte di tutti gli operatori del settore, con il conseguente raggiungimento di due vantaggi: notevoli risparmi economici e miglior utilizzo di tutte le strutture (veicoli, strade) a disposizione.

Il portale netT dovrebbe diventare il riferimento del mondo dei trasporti, cercando di aggredire e risolvere i problemi cronici di tale mercato, quello che Amazons rappresenta per i libri o Yahoo per la ricerca su web.

Il progetto completo prevede l'utilizzo di tutte le tecnologie legate alla rete: utilizzo dei telefoni cellulari con protocollo WAP, di quelli con incorporata la tecnologia GPS, impiego degli small device.

In particolare si vuole offrire: informazioni sul traffico in tempo reale, localizzazione dei mezzi di una flotta di veicoli con l'utilizzo dei cellulari dotati di tecnologia GPS , possibilità di richiedere la rotta ed altre informazioni (ad esempio locazione di officine, alberghi) con telefoni mobili dotati di protocollo WAP da bordo dei veicoli, utilizzo di netT per la ricerca e la fornitura di automezzi, utilizzo della tecnologia futura di accesso ad Internet, basata sui terminali UMTS, pianificazione dei viaggi (ottimizzazione dei trasporti con la generazione di rotta e piano di carico), realizzazione del

sistema di pianificazione dei trasporti cosiddetti ‘a domanda’ in ambiente urbano (dove è concentrata la maggior parte della mobilità nazionale), tracking delle merci in movimento, creazione di una borsa dei trasporti, con l’equazione: ‘Trasporto merce: da costo ad opportunità’, possibilità di offrire corsi di formazione online (quest’ultima attività sviluppata in cooperazione con la Scuola Nazionale dei Trasporti e della Logistica), possibilità alle piccole aziende di trasporti di conoscersi attraverso il portale e di consorziarsi.

3.2 Le pagine ASP.NET

Per quanto riguarda le pagine ASP.NET create, esse riguardavano principalmente la registrazione di una azienda al servizio in rete (con autenticazione per accedere al servizio), l’inserimento di tutti i dati relativi alle varie tipologie di mezzi utilizzati dall’azienda per il trasporto (con la possibilità di modificare o aggiornare i dati inseriti) e l’inserimento dei dati di tutti i clienti dell’azienda registrata.

Adesso descriverò il lavoro svolto per ogni pagina creata.

3.2.1 La pagina di registrazione di una azienda

3.2.1.1 Descrizione della pagina

La prima pagina riguardava la registrazione di una azienda al servizio in rete. Per potersi registrare un utente deve inserire alcuni dati generali dell’azienda, tra cui: la ragione sociale, l’indirizzo (comprensivo di città, provincia e stato) della sede, un eventuale altro indirizzo, alcune informazioni per essere reperibili (numero di telefono fisso, numero di telefono mobile, fax e due indirizzi e-mail di cui il secondo facoltativo), un codice (chiave primaria nel database del programma, per cui diverso per ogni azienda) e per finire una password utile per l’accesso.

Alcuni dati inseriti da un utente che si vuole registrare devono essere considerati validi prima di essere immessi nel database. Per alcuni campi basta soltanto accertarsi che sia stato inserito un valore, ma per altri bisogna anche verificare che il valore inserito sia plausibile: tra questi il codice di avviamento postale (CAP) che deve essere di 5 numeri, il codice fiscale, l’indirizzo e-mail e la password. In più il campo “riconferma password” deve essere compilato con lo stesso testo del campo password.

Ovviamente la pagina di registrazione deve funzionare anche come pagina di aggiornamento dei dati, visto che i campi utilizzati per l’aggiornamento sono chiaramente gli stessi di quelli usati per la registrazione.

3.2.1.2 Descrizione del codice

Il codice della pagina di registrazione di una azienda è diviso principalmente nei seguenti metodi: la `Page_Load` (che gestisce l'evento di caricamento della pagina), la `btnConferma_Click` (che gestisce l'evento del click del mouse sul bottone di conferma), il metodo `CheckDati` (che controlla se il codice con cui si vuole registrare un utente non sia già stato utilizzato da qualcuno), i due metodi `BindProv` e `BindNazione` (che servono per compilare le caselle combinate con elenco a discesa (`DropDownList`) sia per le province che per le nazioni) e per finire `Button1_Click` (che gestisce l'evento del click del mouse sul bottone di ritorno alla pagina principale).

Nel metodo `Page_Load` prima di tutto si compilano le 4 `DropDownList` (due per indicare le province e due per le nazioni), aggiungendo a quelle che specificano l'indirizzo opzionale il campo "Non presente". Poi si va a controllare se nella richiesta della pagina è stato passato il campo `CodAz`, che indica che si sta procedendo all'aggiornamento dei dati dell'azienda il cui codice è passato in quel campo.

Se tale campo contiene un valore, il codice creerà un comando SQL invocando una `Stored Procedure` chiamata "`SPSelectDati`" che andrà a prendere i dati dell'azienda direttamente nel database (il codice dell'azienda è, come già detto, chiave del database), e con questi compilerà i vari campi della pagina, permettendo all'utente di modificarli.

Altrimenti la pagina verrà visualizzata vuota, in modo da permettere l'inserzione di tutti i dati di una nuova azienda.

Il metodo `btnConferma_Click` funziona in modo analogo: prepara il comando SQL e fa iniziare una transazione (si avrà comunque bisogno di due comandi da inviare al database, e se uno dei due fallisce devono essere cancellati gli effetti di entrambi), poi controlla se il campo del codice dell'azienda è vuoto o no. Nel caso in cui sia vuoto prima si invocherà il metodo `CheckDati` e se questo dà risultato negativo non verrà effettuato nessun comando SQL, altrimenti si eseguirà una `INSERT` sul database delle aziende e una su quello delle login. Nel caso in cui il campo del codice non sia vuoto verranno eseguiti due aggiornamenti (sia per l'inserzione che per l'aggiornamento si userà una `Stored Procedure`).

Il metodo `CheckDati` fa un controllo sul database per vedere se esiste già una azienda registrata con un determinato codice. Usa una funzione SQL (praticamente una `Stored Procedure` che però restituisce un unico valore, nel nostro caso "1" se esiste già il codice passato come argomento, "0" altrimenti) e restituisce un valore booleano in base al valore restituito dalla funzione.

I due metodi `BindProv` e `BindNazione` funzionano allo stesso modo: usano un comando SQL per ottenere i dati da utilizzare per compilare le `DropDownList` che vengono passate come parametri. La differenza tra i due metodi sta nella `Stored Procedure` utilizzata.

Per finire il metodo `Button1_Click` fa tornare l'utente alla pagina principale attraverso il metodo `Redirect`.

3.2.2 La pagina di autenticazione

3.2.2.1 Descrizione della pagina

È la pagina di ingresso ai servizi offerti nel sito. È composta solo da due campi di testo, uno che raccoglie la username e uno per la password.

Il funzionamento è semplice: una volta che vengono compilati i dati, viene fatto un accesso al database delle login per vedere se esiste un campo che corrisponde ai dati inseriti. Se non corrisponde viene visualizzato un messaggio di errore, altrimenti si registra un cookie sul computer dell'utente che vuole accedere per mantenere la username, utile come chiave di accesso a tutti i dati dell'utente.

3.2.2.2 Descrizione del codice

La pagina svolge tutte le sue funzioni durante il caricamento. La prima volta che viene caricata, il programma non fa niente di più che non visualizzare i due campi di testo (username e password) e il pulsante di invio.

Durante i caricamenti successivi al primo (postback), che vengono realizzati soltanto attraverso la pressione sul tasto di invio, il programma esegue un comando SQL sul database delle login. Se la risposta dal database è vuota vuol dire che non ci sono aziende con quel codice, altrimenti si controlla che la password dell'azienda trovata corrisponda a quella inserita dall'utente.

Se anche le password coincidono, l'utente è indirizzato sulla pagina principale, e viene registrato sul suo computer un cookie contenente la sua username.

3.2.3 La pagina principale

3.2.3.1 Descrizione della pagina

È la pagina in cui sono presenti tutti i pulsanti per le altre relative pagine. Da qui si possono raggiungere le pagine per: aggiungere una nuova azienda, inserire un mezzo o aggiornare la propria lista di mezzi, modificare i propri dati, inserire un cliente o aggiornare i dati di un cliente inserito.

La pagina è composta da una serie di link a altre pagine. Per distinguere tra una pagina i cui campi di testo sono compilati con i dati richiesti e una i cui campi sono lasciati vuoti (tipicamente il caso

di una pagina usata sia per la registrazione che per l'aggiornamento dei dati), nella richiesta della pagina vengono aggiunti alcuni campi divisi da un punto interrogativo. Ogni campo è composto da un nome, il simbolo di uguaglianza e il valore.

A questo punto nelle varie pagine basta vedere se questi campi sono o no nulli: se lo sono, la pagina deve essere restituita coi campi vuoti, altrimenti generalmente si fa un accesso al database per trovare le informazioni necessarie a compilare la pagina.

3.2.3.2 Descrizione del codice

Il codice C# che svolge le funzioni della pagina non è particolarmente complesso. Si tratta soltanto di fare, durante il caricamento della pagina stessa, un controllo sul database per vedere se l'utente è iscritto come azienda non trasportatrice. Se questo controllo risultasse negativo, allora automaticamente l'utente sarebbe iscritto come azienda trasportatrice.

Questo va a influenzare i collegamenti per aggiornare i dati dell'utente, poiché cambia la pagina a cui il link fa riferimento.

Tutti gli altri collegamenti rimangono invariati per entrambi i tipi di azienda, e mandano alle varie pagine inserendo nella richiesta della pagina gli eventuali campi utili per distinguere tra la modalità di inserzione e quella di aggiornamento.

3.2.4 La pagina di inserimento dei mezzi

3.2.4.1 Descrizione della pagina

Questa pagina è stata copiata direttamente dal programma TRAVEL, ed è composta da una serie di campi da compilare riguardanti i dati di un mezzo. Serve per inserire le varie tipologie di automezzo, ovvero la sua descrizione con caratteristiche specifiche di trasporto.

Nella pagina sono richieste informazioni riguardanti: il tipo di mezzo (identificativo, descrizione, società), le caratteristiche (le dimensioni esterne, il tempo di carico e scarico, la percentuale minima di caricamento, la tolleranza, ovvero la quantità di merce eccedente la portata massima che è possibile caricare, il tipo di velocità e i tempi massimi di viaggio), la portata (relativa alla sola motrice, massima, il volume del mezzo, la portata limite utilizzata per il calcolo della tariffa nel caso in cui il mezzo abbia il rimorchio e nel caso in cui sia senza).

La pagina viene utilizzata, anche in questo caso, sia per l'inserzione che per l'aggiornamento. In caso di aggiornamento però sono disponibili anche i pulsanti per scorrere i vari tipi di mezzo inseriti dall'azienda, e il pulsante per inserirne i vani.

Anche in questo caso alcuni dei campi, prima di essere inseriti o aggiornati, devono essere controllati con gli appositi validator (generalmente i campi più importanti sono obbligatori).

3.2.4.2 Descrizione del codice

Il codice di questa pagina è composto dal metodo che gestisce l'evento di caricamento della pagina (`Page_Load`), dai sei metodi che gestiscono gli eventi di pressione dei sei rispettivi pulsanti (`btnConferma_Click` per il pulsante di conferma, `btnSuccessivo_Click` e `btnPrecedente_Click` per i bottoni che servono a scorrere tra i vari mazzi registrati, `btnDelete_Click` per il bottone di cancellazione di un mezzo, `btnVani_Click` per il bottone che visualizza i vani registrati per il mezzo e infine `Button1_Click` per il bottone di ritorno alla pagina principale) e dal metodo `CompDati` che serve per compilare i campi della pagina.

Durante il caricamento (`Page_Load`) se nella stringa di richiesta della pagina non compare il campo `Mod` significa che la pagina va presa in modalità di inserzione. In questo caso viene visualizzato un testo in cui viene scritto il numero di mezzi già registrati sotto quella azienda (tutto ciò fatto mediante una richiesta al database e un conteggio dei campi restituiti).

Altrimenti vengono messi a zero due campi (che verranno resi come invisibili nella pagina html) il cui valore viene utilizzato al fine di mantenere il numero del mezzo visualizzato e di poter gestire la visualizzazione dei vari mezzi attraverso i due appositi pulsanti. Fatto ciò si richiama il metodo `CompDati` utile per compilare i vari campi della pagina in caso di aggiornamento.

Il metodo `CompDati` utilizza i valori dei due campi citati prima per mantenere in uno il numero di pagina da visualizzare, nell'altro il numero di pagina estratto dall'oggetto `DataReader` restituito dalla richiesta fatta al database. Un ciclo esegue le estrazioni dal `DataReader`, e quando i due valori sono uguali vengono visualizzati i dati del mezzo estratto. Un po' di controlli permettono di visualizzare determinati messaggi quando ad esempio si tenta di accedere a un oggetto precedente al primo o di seguito all'ultimo.

I metodi `btnSuccessivo_Click` e `btnPrecedente_Click` hanno quindi soltanto il compito di aggiornare il valore del numero di pagina da visualizzare, il primo incrementandolo e il secondo decrementandolo di uno. Alcuni controlli servono per evitare che il valore diventi invalido sia in eccesso che in difetto.

Il metodo `btnConferma_Click` fa il controllo per vedere se la modalità è di inserzione o di aggiornamento, dopo di che richiama la `Stored Procedure` adeguata alla situazione (cioè quella di inserzione o quella di aggiornamento) passando al comando `SQL` i relativi parametri.

Anche il metodo `btnDelete_Click` serve a invocare una `Stored Procedure`, il cui compito è quello di rimuovere tutti i dati relativi al mezzo in questione.

Infine abbiamo il solito metodo `Button1_Click` che rimanda alla pagina principale.

3.2.5 La pagina di inserimento dei vani di un mezzo

3.2.5.1 Descrizione della pagina

Per quanto riguarda la pagina di registrazione/modifica dei vani di un mezzo, i campi da compilare sono solo quattro: il numero del vano, il volume, la portata e se il vano appartiene o no al rimorchio. Potendo però esserci un numero abbastanza elevato di vani su ogni mezzo, la pagina è stata creata mettendo i campi per l'aggiunta di un nuovo vano seguiti da un DataGrid, che viene compilato con i dati di tutti i vani, uno per ogni riga.

Per potere aggiornare i dati inseriti, viene aggiunta una colonna a quelle della tabella. Questa è una colonna di bottoni, la cui funzione è quella di identificare quale riga deve essere evidenziata e resa modificabile. Infatti una volta premuto un pulsante tutti i campi di testo della riga selezionata diventano TextBox già compilati con i dati della relativa cella, e le CheckBox della colonna di appartenenza al rimorchio passa dallo stato disabilitato (cioè si può vedere il valore, ma non può essere modificato) a quello di abilitato.

In seguito al passaggio in modalità di modifica, il tasto della colonna dei bottoni cambia la sua funzione, e diventa un comune tasto di sottomissione di dati al server. Ricompilati i campi, l'utente premerà il tasto per attivare un comando SQL che aggiornerà i dati relativi al vano scelto, nel database.

3.2.5.2 Descrizione del codice

La pagina è composta dai campi che vanno compilati, da un bottone di invio e da un DataGrid che mostra i vani già inseriti per quel mezzo.

I metodi gestiti dal codice C# sono i seguenti: il `Page_Load` che gestisce l'evento di caricamento della pagina, il `BindDataGrid` che compila i dati della tabella, `EventEdit`, `EventCancel` e `EventUpdate` che gestiscono gli eventi associati alla tabella (il primo per quello del tasto di selezione di una riga della tabella, il secondo per quello della cancellazione di un vano e il terzo per gestirne quello di aggiornamento) e per finire, oltre al solito `Button1_Click` per gestire il ritorno alla pagina principale, il `btnConf_Click` per inserire i dati di un nuovo vano per il mezzo selezionato.

Nella `Page_Load` non si fa niente di più se non chiamare il metodo `BindDataGrid` e visualizzare il nome del mezzo i cui vani verranno gestiti dalla pagina.

La `BindDataGrid` invece, utilizzando le informazioni passate attraverso i cookie (quindi il codice dell'azienda e il nome del mezzo in questione), genera un comando SQL che, per mezzo di una Stored Procedure, va a leggere dal database i dati relativi ai vani del mezzo e con questi

compila il DataGrid invocando il metodo `DataBind`. Se la richiesta non restituisce nessun campo, il DataGrid non viene nemmeno visualizzato.

I tre metodi che gestiscono gli eventi del DataGrid ricevono come argomento, oltre a un puntatore all'oggetto che ha invocato l'evento, un `DataGridCommandEventArgs`, utile per sapere su quale riga della tabella è stato invocato l'evento.

Il metodo `EventEdit` non fa altro se non impostare l'attributo `EditItemIndex` del DataGrid all'indice di riga passato dall'argomento, segnarsi il numero del vano (chiave del database dei vani) in una variabile nascosta (servirà nel caso di aggiornamento) e invocare un nuovo `BindDataGrid` per visualizzare la tabella con la riga selezionata pronta per essere modificata.

Il metodo `EventCancel` invece imposta quell'attributo a "-1", ovvero nessuna riga verrà visualizzata in maniera da poter essere modificata, invoca attraverso un comando SQL una Stored Procedure per cancellare la riga selezionata e alla fine invoca nuovamente la `BindDataGrid` per rivisualizzare gli effetti della cancellazione appena effettuata della riga.

Il metodo `EventUpdate` si comporta approssimativamente come l'`EventCancel` solo che invece che cancellare la riga, il comando SQL la aggiorna usando il vecchio numero del vano (salvato durante l'`EventEdit`) per sapere su quale riga fare l'aggiornamento.

Infine, il metodo `btnConf_Click` non fa altro se non una `INSERT` nel database dei vani. L'inserzione, come l'aggiornamento, può generare delle eccezioni (ad esempio se si inserisce un vano il cui numero esiste già) che vengono gestite visualizzando un messaggio di errore.

3.2.6 La pagina di registrazione dei clienti

3.2.6.1 Descrizione della pagina

La pagina di registrazione dei clienti contiene molti campi utili per avere una descrizione dettagliata di ogni singolo cliente gestito dall'azienda dell'utente. Oltre ai dati anagrafici (ragione sociale, indirizzo, CAP etc etc), si possono inserire le caratteristiche del cliente (denominazione del suo sito, portata massima di un mezzo (espressa in quintali) consentita per tale cliente, l'altezza e la larghezza massima consentita per i mezzi, la modalità di scarico del mezzo in base ai vincoli imposti dal cliente stesso e il tempo massimo consentito per lo scarico, le fasce orarie di consegna ammesse a cui fare riferimento, la regione di appartenenza), le caratteristiche particolari del cliente inserito (il gruppo di appartenenza, l'eventuale giro di consegne di cui fa parte, l'eventuale distanza aggiuntiva che deve essere percorsa dal mezzo (ad esempio distanza da percorrere per effettuare la pesa), la tariffa di consegna applicata al cliente, il numero della cella che gli appartiene, l'eventuale piattaforma a cui è collegato, se il cliente deve fare parte di un viaggio costituito da clienti

omogenei, un valore che indica la sua priorità di consegna, se il cliente necessita di un mezzo coperto, la sua zona di appartenenza e le note aggiuntive).

Anche in questo caso la pagina di registrazione, con qualche modifica data dal codice ASP, viene utilizzata anche per l'aggiornamento dei dati. Tuttavia, essendo generalmente tanti i clienti di una azienda, per selezionare quello di cui si vuole aggiornare i dati è stato necessario creare una pagina per ricercarne uno mediante l'inserimento di alcuni dati. Di questa pagina parlerò nel paragrafo 3.2.7.

3.2.6.2 Descrizione del codice

Oltre ai metodi per gestire l'evento di caricamento della pagina (`Page_Load`), quello per la pressione sul bottone di conferma (`btnConf_Click`), sul bottone per cancellare il cliente selezionato (`btnDelete_Click`) e sul bottone di ritorno alla pagina principale (`Button1_Click`), il codice è composto da altri due metodi per compilare le `DropDownList` per le province (`BindProv`) e per le nazioni (`BindReg`) e da uno per eseguire le `Stored Procedure` (`useStore`).

Questa pagina si comporta in maniera molto simile alle altre: al caricamento della pagina si controlla un cookie che contiene il codice di identificazione del cliente selezionato. Se è nullo vuol dire che la pagina è in modalità di inserimento, altrimenti di aggiornamento. Nel secondo caso nei campi compariranno i dati del cliente selezionato.

La `useStore` è un metodo che genera un comando SQL prendendo come argomento il nome della `Stored Procedure` da richiamare. In questa pagina ho optato per creare questo metodo soltanto per rendere più comprensibile e compatto il codice.

Gli altri metodi si comportano esattamente come i corrispondenti metodi già visti nelle altre pagine.

3.2.7 La pagina di ricerca di un cliente

3.2.7.1 Descrizione della pagina

In questa pagina l'utente può cercare un cliente immettendo alcune sue informazioni. Tra queste informazioni ci sono il codice del cliente (univoco), la provincia di appartenenza, la città di appartenenza, la priorità di consegna e la ragione.

Il funzionamento è particolarmente semplice: ci sono i campi da compilare e un bottone per inviare i dati al server. I dati inviati vanno a creare una query SQL in maniera dinamica, dove i vincoli sono dati dall'uguaglianza con i dati inviati. Quindi se, per esempio, un utente invia un certo codice cliente, nella query SQL comparirà "WHERE codice = @valore" dove @valore è il valore inviato.

Il risultato di questa query servirà a compilare un DataGrid limitato alle sole informazioni di cui si può fare la ricerca. Il DataGrid avrà anche una colonna di bottoni che serviranno per selezionare un cliente. Una volta premuto uno di questi bottoni, i dati scelti andranno a compilare la pagina di registrazione dei clienti, e si potranno aggiornare e rispedire al server che li userà per un aggiornamento del database.

In più, se il risultato della query consiste in un unico elemento (è il caso, ad esempio, di ricerca per mezzo del codice del cliente), il datagrid non viene nemmeno mostrato, ma si passa direttamente a compilare la pagina di registrazione coi dati dell'elemento trovato.

3.2.7.2 Descrizione del codice

Al caricamento (`Page_Load`) non si fa altro che compilare la `DropDownList` con le province (prese da un database attraverso il metodo `BindProv`) e aggiungerci il campo "Non presente" con valore nullo.

Oltre a questi due metodi (il cui comportamento nei dettagli è già stato discusso per le pagine precedenti), e quello che gestisce l'evento di pressione del bottone per tornare alla pagina principale, vi sono poi i due metodi per gestire la pressione del bottone di ricerca e quello per la gestione della pressione di uno dei bottoni di aggiornamento presenti nella colonna aggiunta al DataGrid di cui ho parlato nella descrizione della pagina.

La ricerca di un cliente avviene compilando i dati della pagina. Alla pressione del tasto di ricerca, il metodo che gestisce questo evento crea un comando SQL e genera una stringa che sarà il testo del comando da inviare al database.

In questa stringa all'inizio, dopo il `WHERE`, ci sarà soltanto la clausola di uguaglianza tra il codice dell'utente e il codice di registrazione dell'azienda. Poi, per ogni campo compilato nella pagina, verrà aggiunta una ulteriore clausola di uguaglianza tra il campo nella pagina e quello nel database.

Questo significa che se viene premuto il tasto di ricerca senza compilare nessun campo, il comando restituirà tutti i clienti registrati dall'utente, mentre se vengono compilati alcuni campi, tra tutti i clienti verranno restituiti solo quelli che rispettano tali parametri.

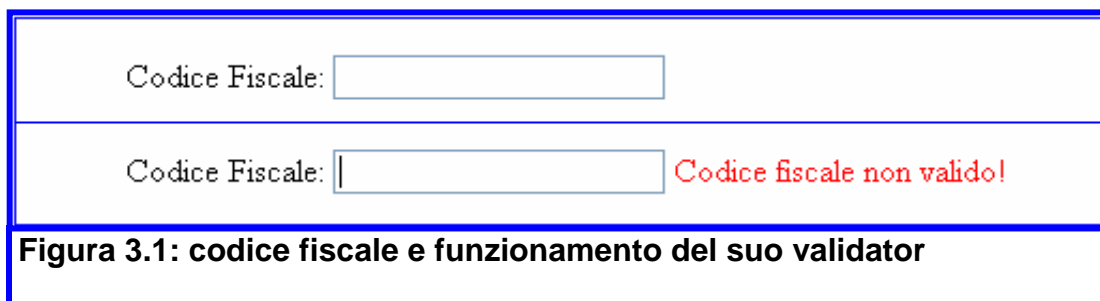
Infine il metodo usa la risposta dell'esecuzione della richiesta al database per compilare il DataGrid. Nel caso in cui la ricerca non dia nessun risultato, la pagina viene visualizzata priva della tabella, altrimenti se il risultato è un solo cliente questo viene passato direttamente alla pagina di aggiornamento dei clienti (aggiornando il cookie che contiene il codice del cliente selezionato e caricando la pagina di registrazione dei clienti). Nel caso di un risultato che prevede più clienti, verrà compilato e reso visibile il DataGrid.

A questo punto, per aggiornare un cliente basterà un click sul pulsante posto all'inizio della riga che contiene i suoi dati. Questo aggiornerà il cookie e caricherà la pagina come descritto in precedenza nel caso di un unico cliente trovato.

3.3 I web-control creati

La seconda parte del mio lavoro è stata costituita dalla creazione di web-control per migliorare le pagine già create e per avere validi mezzi per creare le pagine future. I web-control creati sono qui descritti dettagliatamente.

3.3.1 Il codice fiscale



3.3.1.1 Descrizione del Web Control

Il primo controllo creato serve per avere una TextBox già predisposta a verificare la correttezza di un codice fiscale inserito. In pratica il controllo prevede la creazione di una scritta (che generalmente sarà “Codice Fiscale:”, ma che può essere passata come attributo), di un campo di testo da poter compilare, e di due validator: uno per controllare che venga compilato il campo (quindi un RequiredFieldValidator) e uno per controllare che il codice fiscale immesso possa essere valido (quindi un RegularExpressionValidator).

3.3.1.2 Descrizione del codice

Questo Web Control utilizza 4 oggetti, che vengono dichiarati come globali: un Label (che farà comparire la scritta che, come affermato prima, generalmente sarà “Codice Fiscale”), un campo di testo (per inserire i dati) e due validator (uno che controlla che il campo non sia vuoto: RequiredFieldValidator, l’altro che controlla che il campo sia compilato come previsto da una espressione regolare: RegularExpressionValidator).

Le proprietà pubbliche di questo Web Control sono: il testo presente nell’apposito campo di testo, il testo della Label, il testo dell’errore generato dal RequiredFieldValidator e quello del

RegularExpressionValidator, l'espressione regolare da controllare e un intero che indica la posizione (in pixel) da sinistra degli errori (che servirà per sovrapporli).

Il costruttore assegna a un metodo (il Page_Load) la gestione dell'evento di caricamento del controllo. Durante il caricamento vengono aggiunti al contenitore dei controlli il campo di testo e la Label, viene messo come ControlToValidate (proprietà dei due validator che determina quale è il nome del Web Control che deve essere controllato) dei due validator il campo di testo e infine vengono aggiunti anche loro ai controlli.

Poi viene sovrascritto (override) il metodo che viene invocato quando il Web Control deve essere riprodotto col codice HTML. Per questo motivo si crea una tabella con tre campi. Attraverso il metodo RenderControl ereditato da System.Web.UI.WebControls.WebControl viene restituito il codice del Label, che viene messa nel primo campo della tabella, e della TextBox, che viene messa nel secondo campo.

Per sovrapporre i due validator nel terzo campo, uno dei due viene reso normalmente, l'altro invece, copiando gli attributi di stile e modificando la posizione da sinistra attraverso il metodo modifica oppure immettendo il valore direttamente modificando l'attributo di cui ho parlato all'inizio, viene sovrapposto al primo.

Il metodo modifica infatti prende la stringa contenente la posizione in cui verrà generato il controllo, fa una stima dello spazio occupato dagli altri due campi della tabella e pone approssimativamente il comando all'inizio del terzo campo. Il metodo però si affida a una stima delle dimensioni (la TextBox viene stimata grossa 140 punti, e ogni carattere del Label 8 punti) perciò, se venisse visualizzato non del tutto sovrapposto, è comunque possibile usare la proprietà pubblica per immettere a quanti punti da sinistra posizionare il messaggio.

3.3.2 Il calendario

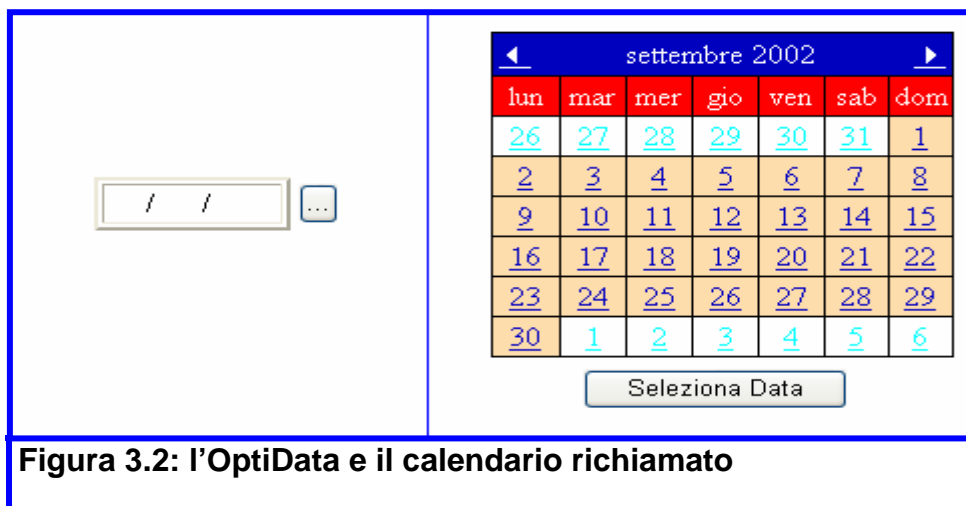


Figura 3.2: l'OptiData e il calendario richiamato

3.3.2.1 Descrizione del Web Control

Questo web-control è stato scomposto in tre parti. Il primo controllo creato, chiamato “OptiDataControl”, era fondamentalmente un campo formato da tre TextBox. Le TextBox erano raggruppate in una tabella e separate da un separatore (di solito il simbolo “/”, ma un separatore diverso può essere impostato cambiando la omonima proprietà).

La tabella che raggruppa tutto ha i margini esterni visibili, ma quelli tra i vari campi invisibili, in modo da far sembrare tutto un unico campo.

Dovendo fornire un campo per una data, le proprietà che permettono di accedere al testo delle TextBox sono degli interi (quindi se una persona vuole inserire un valore diverso da un intero non può farlo), in più è possibile leggere una proprietà del controllo che restituisce indietro un oggetto di tipo DateTime, che altro non è se non una stringa già in forma di data.

Il secondo controllo altro non è se non un validator particolare (chiamato “OptiDataValidator”), che deve fare il controllo sui tre campi di un oggetto di tipo “OptiDataControl” tenendo conto l’uno dell’altro. Infatti una data per essere valida deve avere: il valore dell’anno compreso tra 1 e 9999, il campo del mese compreso tra 1 e 12 e il campo del giorno compreso tra 1 e 30 se il mese è 4(aprile), 6(giugno), 9(settembre), 11(novembre), tra 1 e 31 se il mese è 1(gennaio), 3(marzo), 5(maggio), 7(luglio), 8(agosto), 10(ottobre), 12(dicembre), invece deve essere tra 1 e 28 se il mese è 2(febbraio) e l’anno non è bisestile (un anno è bisestile se in modulo 4 il suo valore è zero e se in modulo 100 il suo valore è diverso da zero), mentre se l’anno è bisestile deve essere tra 1 e 29.

Ovviamente, per rendere la cosa più rapida e risparmiare sulle richieste al server, la verifica sulla correttezza della data è stata fatta in javascript, anche se è possibile lasciarla lato server, in modo che l’utente possa farla funzionare sul proprio pc (quindi lato client).

Il terzo controllo è formato da un oggetto di tipo OptiDataControl, da uno di tipo OptiDataValidator e da un pulsante. In pratica il web-control permette di immettere una data e, premendo il bottone, la data viene controllata. Se valida, viene aperta una pagina ASP contenente solamente un calendario e un pulsante. Il calendario mostra la data selezionata, ma è possibile cambiare la data usando i vari tasti presenti nel calendario (vedere il controllo Calendar).

Il pulsante serve per tornare alla pagina principale, e per modificare il valore scritto nell’OptiDataControl nel caso sia selezionata una data diversa a quella immessa in precedenza.

3.3.2.2 Descrizione del codice

Come già detto questo Control è diviso in più parti: `OptiDataControl` che genera i campi di testo separati da un apposito separatore, `OptiDataValidator` che controlla che la data sia valida (utilizzando anche tutta una serie di JavaScript salvati nel file `Validator.js`) e per finire

la `OptiData` che contiene gli altri due e implementa tutte le funzioni necessarie. Per una descrizione esauriente del controllo farò una analisi distaccata dei tre codici.

Il primo (`OptiDataControl`) ha le seguenti proprietà: la stringa `Separatore`, che di default è il carattere “/”, e che appunto serve per separare il campo “giorno” da quello “mese”, e il campo “mese” da quello “anno”, le stringhe `Giorno`, `Mese` e `Anno`, con cui è possibile ottenere o impostare gli omonimi campi della data, e per finire il campo `Data`, ovvero un oggetto di tipo `DateTime` utile in alcuni casi, che viene restituito creandolo a partire dai 3 campi che compongono la data.

Durante il caricamento della pagina vengono assegnati tre identificativi unici ai tre campi (praticamente all’identificativo del controllo, che è unico di suo tra i controlli che costituiscono una pagina, vengono aggiunti in fondo un paio di lettere: “gg” per giorno, “mm” per mese e “yr” per anno).

Nella `Render` invece si crea una tabella i cui bordi sono visibili, si assegna una dimensione per i tre campi e un massimo numero lettere con cui possono essere compilati, poi si rende il codice in modo che i tre campi vengano di seguito, senza bordi e separati dal `Separatore`.

Il secondo, l’`OptiDataValidator`, deriva da `Label` e dall’interfaccia `Ivalidator`, e quindi non da `WebControl` come tutti gli altri.

Ha le proprietà tipiche di un `validator`, e la possibilità di essere reso attraverso uno script in modo da funzionare lato client. Questo control è formato da alcuni metodi il cui compito è registrare sulla pagina lo script nel caso che la proprietà `EnableClientScript` sia impostata a “vero”. Lo script è mantenuto in parte nel codice `c#`, in parte nel file `Validator.js` di cui parlavamo prima. Poi ci sono i metodi per la convalida lato server (utili nel caso del `EnableClientScript` impostato a “falso”). I controlli da fare per vedere se una data è valida sono stati spiegati nella descrizione della pagina.

Per descrivere rapidamente il funzionamento dello javascript, caricato dal file `Validator.js`, questo controlla che la proprietà `ControlToValidate` del `validator` punti a un oggetto della pagina. A questo punto accede ai 3 campi e ne fa una stringa unica e la passa alla funzione di controllo. Questa, attraverso una espressione regolare, divide i campi del giorno, del mese e dell’anno e esegue esattamente lo stesso controllo che avrebbe eseguito il codice `c#` lato server.

Se la data è valida, viene aperta la pagina `Calendario.aspx`, contenente un calendario il cui codice, attraverso un proprio javascript, permette di modificare la data inserita selezionandola direttamente e premendo un bottone di conferma. A pressione avvenuta il javascript copia i valori della data nei tre appositi campi (nella pagina contenente il controllo) e chiude la finestra del calendario.

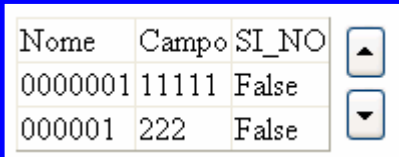
Infine c'è l'ultima parte, la `OptiData`. Questo controllo dichiara tre oggetti globali: un `OptiDataControl`, un `OptiDataValidator`, e un bottone. Le proprietà sono le stesse della `OptiDataControl` più la stringa `Errore` che contiene appunto il messaggio di errore che verrà visualizzato in caso di data invalida.

Il costruttore affida al metodo `Page_Load` la gestione dell'evento di caricamento e mette il testo composto da tre puntini sul bottone.

Nel caricamento del controllo vengono aggiunti alla collezione dei controlli tutti gli oggetti globali, dando ad ognuno di essi un identificativo unico e compilando il campo `ControlToValidate` del validator con l'identificativo dell'oggetto `OptiDataControl`.

Infine, nella `Render`, i tre controlli vengono messi in una tabella uno dietro l'altro, occupando i tre campi.

3.3.3 La *OptiDataGrid*



| Nome | Campo | SI_NO | |
|---------|-------|-------|---|
| 0000001 | 11111 | False | ▲ |
| 000001 | 222 | False | ▼ |

Figura 3.3: La *OptiDataGrid*

3.3.3.1 Descrizione del Web Control

Di questo web-control ne sono state implementate due versioni: la differenza tra le due risiede nel modo in cui vengono passati alcuni parametri utili per il funzionamento.

Fondamentalmente questo control altro non è se non un `DataGrid` in cui, tra i parametri, viene messa la stringa per accedere al database, il nome della tabella del database, le colonne che comporranno il `DataGrid` e il numero di righe visualizzate per ogni pagina.

A fianco del `DataGrid` sono stati messi due pulsanti che permettono di scorrere tra le pagine della tabella.

La seconda versione (il cui nome è "`OptiXMLGrid`") prende tutti i dati relativi alla richiesta al database (quindi la stringa di accesso, il nome della tabella e il nome delle colonne) direttamente da un file XML. Tramite le proprietà quindi bisogna indicare il nome del file XML (che verrà cercato nella cartella in cui risiede tutto il progetto) e il numero di righe per pagina.

Il `DataGrid` all'interno del controllo verrà compilato durante il caricamento della pagina.

3.3.3.2 Descrizione del codice

Vista l'enorme somiglianza tra i due codici, prenderò in esame soltanto quello riguardante la `OptiXMLGrid`.

Le proprietà di questo Web Control sono: la stringa `NomeFileXML` per indicare il nome del file XML in cui sono raccolti i dati utili per la compilazione della pagina, la stringa `Filtro` che serve per il campo “WHERE” del comando SQL che dovrà essere generato, e per finire l’intero `ItemXPage` che determina il numero di elementi visualizzati contemporaneamente nella `DataGrid`. Il costruttore aggiunge alla lista di controlli due bottoni (per scorrere le pagine del `DataGrid`) e il `DataGrid` stesso e affida gli eventi dei click sui due bottoni e del caricamento del controllo a tre metodi.

Durante il caricamento vengono compilati alcuni attributi del `DataGrid`, poi si genera un oggetto `DataSet` e gli si fa leggere il file XML (passandogli il nome del file completo di path in cui il file si trova). Nella `ViewState` si registra un campo “log” con la stringa di connessione al database (che deve essere inserita nel file XML) e un campo “query” con il testo del comando SQL da invocare. Per creare questo testo viene invocato il metodo `GetQueryString`, il quale costruisce, usando il `DataSet` con i dati del file XML, la stringa richiesta (mettendo le virgole al posto giusto e stando quindi attento alla sintassi).

Poi viene invocata la `BuildColumnsGrid` che genera una colonna per ogni campo “Fields” che compare nel file XML, e tale colonna, sempre in base ai dati del file, viene resa con un campo di testo o con una `CheckBox` e compilata con gli opportuni valori restituiti dai comandi SQL (inserendo nella proprietà `DataField` da quale campo del database deve prendere i dati). In più alla fine si aggiunge la colonna alla lista dei controlli del `DataGrid`.

Poi, sempre nel caricamento della pagina, si invoca il metodo `GenerateData` che esegue il comando SQL, e si utilizza un ciclo per eliminare le prime righe (tante quante servono per fare vedere la pagina selezionata). Questo metodo per visualizzare le pagine del `DataGrid` rende possibile assegnare ai due pulsanti creati le funzioni per cambiare pagina.

Il controllo comprende altri metodi: i due che gestiscono gli eventi dei pulsanti, che non fanno altro se non incrementare o decrementare il contatore che segnala il numero di pagina da visualizzare, e la `Render` che non fa altro che rendere il controllo nel formato HTML (usando la solita tabella per dare una forma al controllo).

3.3.4 La OptiFullGrid

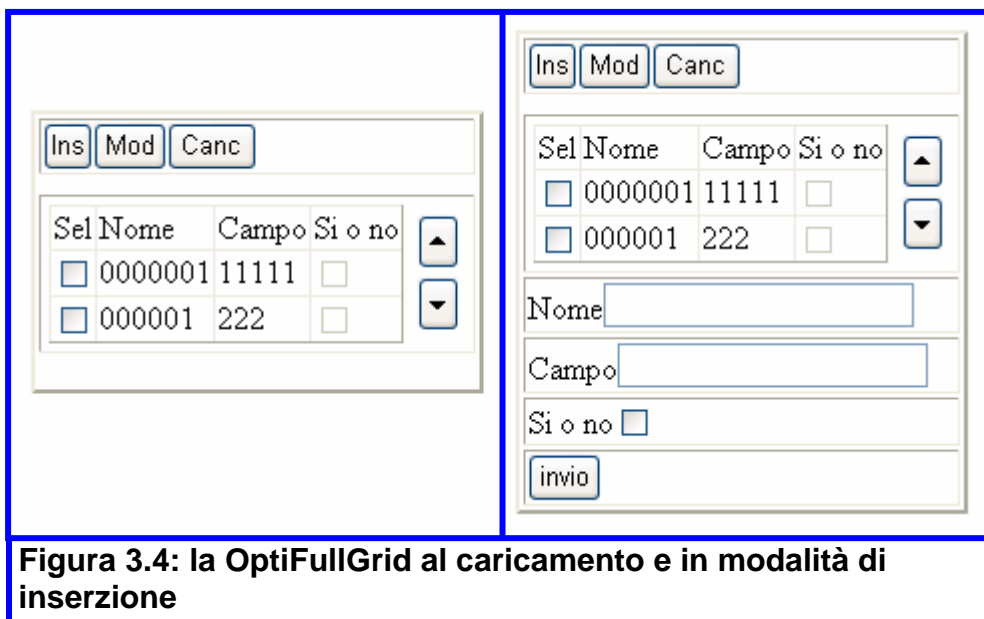


Figura 3.4: la OptiFullGrid al caricamento e in modalità di inserzione

3.3.4.1 Descrizione del Web Control

Questo controllo riprende la OptiXMLGrid, rendendola un web-control molto più complesso e utile.

In pratica il controllo è formato da tre bottoni (il bottone di inserzione, di modifica e di cancellazione), sotto i quali vi è un oggetto OptiXMLGrid, il cui DataGrid viene espanso con una colonna di CheckBox utili per selezionare le varie righe.

Tra le proprietà, oltre a quelle da passare direttamente all'oggetto OptiXMLGrid, c'è da mettere quale dei campi selezionati sia da considerare "chiave" delle tabelle scelte.

Il funzionamento del control prevede che durante il caricamento della pagina, sia inizializzata la OptiXMLGrid mostrando i valori delle colonne scelte. Le pagine della tabella possono essere navigate esattamente come nel controllo unico.

La pressione del tasto di inserzione fa comparire dei campi di testo (uno per ogni colonna presente nella tabella) e un tasto di invio. Dopo aver compilato i campi e aver premuto "invio" viene generato un comando SQL di inserzione, viene ricaricata la pagina ed è possibile vedere la nuova riga presente nella tabella.

La pressione del tasto di modifica invece controlla quale CheckBox della colonna aggiunta alla tabella dell'OptiXMLGrid è stata selezionata (viene considerata soltanto l'ultima CheckBox che ha il campo Checked a "vero") e visualizza i campi come nel caso dell'inserzione, ma questa volta compilati con i dati della riga scelti. Il tasto di invio genererà un comando di aggiornamento e lo invierà al database.

Infine il tasto di cancellazione genererà un comando SQL che cancellerà tutti i campi della tabella selezionati tramite le CheckBox di selezione.

3.3.4.2 Descrizione del codice

Per quanto riguarda le proprietà, sono le stesse della `OptiXMLGrid` con l'aggiunta di un campo di testo che indica quale sia la chiave della tabella..

Il costruttore invece imposta due valori booleani nella `ViewState`, che servono a determinare se il controllo si trova in modalità di inserzione o di modifica. All'inizio non si trova in nessuno dei due stati, per cui nel costruttore i due valori sono messi a "falso". Poi viene inserita una colonna di `CheckBox` nel datagrid dell'oggetto `OptiXMLGrid`. Servirà per selezionare le righe su cui apportare modifiche o da cancellare. Infine, nella lista di controlli, vengono aggiunti quattro bottoni (uno per l'inserzione, uno per la modifica, uno per la cancellazione e uno, all'inizio invisibile, per confermare l'invio dei dati) e vengono assegnati quattro metodi ai relativi quattro eventi di pressione sui bottoni, e uno per l'evento di caricamento del controllo.

Nella `Page_Load` invocata per il caricamento, si usa un `DataSet` per leggere il file XML e si invoca il metodo `GenerateFields` per generare un array di controlli (che possono essere o `TextBox` o `CheckBox`) utili in caso di inserimento o aggiornamento dei dati.

Questo metodo genera anche i vari `Label` in cui mettere la descrizione del campo, presa sempre dal solito file XML. Il funzionamento del metodo è affidato a un ciclo `for` che continua finché non sono stati generati tutti i campi descritti, e nel `DataSet` il campo `Type` descrive se si tratta di un campo di testo (e quindi se va creata una `TextBox`) o di un campo booleano (gestito quindi da una `CheckBox`).

Nella `Render` si crea il codice HTML: una tabella racchiude tutti i campi principali (l'oggetto di tipo `OptiXMLGrid`, i vari bottoni...). Se poi una delle variabili indica che la modalità è quella di inserimento o modifica, vengono mostrati anche gli altri campi (quelli generati tramite `GenerateFields`), compilati, nel caso di modifica, con i dati presi dal `DataGrid`, e il bottone di invio.

Per quanto riguarda il metodo che gestisce l'evento sia del bottone di inserimento che di modifica, non deve fare altro che modificare il corrispondente valore della `ViewState` a "vero". In più, nel caso di modifica verrà tenuto in una variabile l'indice della riga selezionata per la modifica (se ce ne è più di una selezionata, verrà tenuto l'indice di valore maggiore).

Per i due metodi, l'invio dei dati al server è regolato dalla pressione sul tasto "Invio". Il metodo che gestisce questo evento crea un comando SQL e invoca il metodo `GenerateQuery` che genera la stringa SQL. Questo metodo, attraverso una serie di cicli, riesce a generare la stringa adatta a ognuna delle tre modalità, basandosi su due parametri passati: la stringa contenente il tipo di comando (inserzione, aggiornamento o cancellazione) e il `DataSet`.

Nel caso di cancellazione tuttavia non si deve nemmeno premere il tasto “Invio”. La pressione sul tasto di cancellazione genera un comando SQL di cancellazione (con la stessa modalità illustrata prima) per ogni riga del DataGrid selezionata.

Bibliografia

I seguenti sono testi ed altro materiale consultato per la stesura della relazione:

- *TRAVEL – Ottimizzazione e pianificazione dei trasporti*, tesi di laurea di Ettore Nardone – Università degli Studi di Pisa
- Travel Versione 2.0 - Manuale per l'utente
- *Scienze delle decisioni per i trasporti*, a cura di Stefano Pallottino e Anna Sciomachen, CNR-Progetto finalizzato Trasporti 2, collana Trasporti, editore Franco Angeli
- “*Sviluppo e realizzazione di procedure risolutive euristiche per la soluzione del Vehicle Routing Problem*” - Tesi di Laurea di Massimo Lodigiani - Università degli Studi di Milano.
- *La struttura degli algoritmi* di F. Luccio, 1988, Bollati Boringhieri, pp. 146, 158.
- *.NET Framework Developer's Guide, Visual Basic and Visual C# Concepts* dal manuale in linea di Microsoft Visual Studio.NET 7.0
- *Professional ASP.NET*, di R. Anderson, B. Francis, A. Homer, R. Howard, D. Sussman, K. Watson 2001, Wrox Press Ltd.
- *Programming Microsoft .NET*, di Jeff Prosise, edito dalla Microsoft Press
- *Executive Summary Plan e Business Plan* del progetto netT
- *C# Introduction and Overview*, pagina Web del sito della Microsoft dedicato al Visual Studio .NET
- *Java2C#*, dispense per il corso di “Programmazione di Interfacce Grafiche” di Antonio Cisternino
- *ASP.NET*, presentazione dell'azienda Optisoft