Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in

Ingegneria Informatica

Applicazione web per la gestione dei dati utilizzando il framework Java GWT

Relazione finale di tirocinio

Laureando : Roberto Zampieri

Relatore : Prof. S.Congiu

Anno Accademico : 2011-2012

Indice

Introduzione

Questa tesi nasce da un'esperienza di stage effettuata presso un'azienda di informatica che risponde in maniera completa alle esigenze del cliente: privacy, informazioni ed assistenza ad utenti che hanno problemi nella gestione di un prodotto o di un servizio software e hardware, servizi ASP ovvero l'erogazione di software tramite Internet, outsourcing, sicurezza informatica, assistenza sistemistica, formazione e post vendita.

La scelta di includere un'esperienza di stage nel percorso di studi è derivata da una voglia di avere un contatto diretto con il mondo del lavoro e vedere come ciò che ho studiato finora si integra nella realtà aziendale. Durante il percorso di studi, infatti, si apprendono moltissimi concetti nuovi ma talvolta può essere difficile riuscire ad inquadrarli in un contesto lavorativo e quindi un'esperienza di lavoro può risultare davvero utile a questo scopo. Ritengo quindi che la possibilità di compiere una esperienza di stage durante la carriera universitaria di uno studente sia un modo per avere una conferma che quanto fatto finora sia servito per il proprio futuro e per la futura carriera lavorativa.

Anticipo fin da subito che mi ritengo pienamente soddisfatto dell'esperienza che ho avuto, sia perchè mi è piaciuto il progetto che ho portato a termine sia per l'ambiente di lavoro che ho trovato; infatti credo che uno stage possa essere utile, oltre che per vedere applicato ciò che prima era solo teorico, anche perchè permette di conoscere la realtà lavorativa molto diversa da quella scolastica.

L'attività di tirocinio ha avuto come oggetto la progettazione e l'implementazione di un'applicazione Web per la gestione dei dati utilizzando un framework. Per capire meglio di cosa si tratta, il primo capitolo, oltre alla presentazione dell'azienda, descrive il progetto di stage, utilizzando introduzioni teoriche in modo che i lettori meno esperti capiscano di cosa si parli e l'ambito nel quale si colloca il lavoro da svolgere.

L'obiettivo iniziale del tirocinio è stato quello di acquisire sufficienti conoscenze riguardanti le applicazioni Web e tutto ciò che è a loro collegato come l'architettura JEE, i linguaggi di programmazione ed il pattern utilizzato. Tra tutti questi argomenti, presenti nel secondo capitolo, molto importante è la piattaforma standard di sviluppo JEE. JEE è un'architettura piuttosto

complessa ed è composta da un elevato numero di componenti, ognuna delle quali offre uno specifico servizio. Per questo motivo si sono approfondite solo le parti delle tecnologie di questa piattaforma necessarie per lo sviluppo del progetto.

Il terzo capitolo è dedicato esclusivamente al framework Java GWT, mentre nel quarto capitolo si descrive dettagliatamente ed in modo completo il codice sorgente utilizzato per la realizzazione del progetto, dando così un aspetto pratico alle nozioni teoriche apprese durante l'attività di tirocinio. Per vedere i risultati pratici di tale implementazione sono state inserite alcune immagini.

Capitolo 1

Presentazione

1.1 Presentazione dell'azienda

L'azienda presso la quale ho volto lo stage si chiama Scp ed ha sedi operative a Belluno e Villorba (TV). Nata nel 1981, Scp è un'azienda di informatica composta da un gruppo di 60 professionisti ed esperti che lavora per interpretare e soddisfare i bisogni del proprio partner principale: il cliente. La volontà di offrire un servizio completo ed affidabile ha portato alla suddivisione dell'organico in aree specializzate, composte da uno staff di persone che operano al fine di garantire ai propri clienti un servizio informatico completo: dalla fase iniziale di contatto, analisi e progettazione, sino alle fasi esecutive di post-vendita ed assistenza.

Scp ha organizzato al suo interno una rete di servizi che risponde in maniera completa e professionale alle esigenze di cliente. I servizi offerti sono disponibili attraverso un contatto diretto con il team di persone che coopera nelle differenti aree di supporto: servizio commerciale, servizio clienti e servizio tecnico.

Il servizio commerciale si avvale di un gruppo di persone specializzate in distinte aree di business ed il valore aggiunto è dato dalla quotidiana collaborazione con il reparto tecnico: la condivisione ed il continuo scambio di competenze innovative garantiscono al cliente una notevole professionalità e qualità del servizio offerto.

Il servizio clienti si occupa dell'assistenza software, manutenzione hardware e assistenza sistemistica su PC, reti locali e periferiche; per ridurre i tempi di intervento e di trasferta ed eliminare i costi delle spese di viaggio è stata introdotta l'assistenza telematica che unita all'assistenza on site, grazie alla quale i tecnici possono svolgere attività di installazione di prodotti e di aggiornamenti, di consulenza e le di personalizzazioni direttamente presso il cliente, sono un notevole punto di forza dell'azienda.

Il servizio tecnico, composto da un team di sviluppatori coordinati da un

responsabile del progetto, opera al fine di realizzare la soluzione ad hoc per ogni specifica esigenza del cliente. L'attività svolta dal servizio tecnico comprende analisi e progettazione, sviluppo personalizzato, consulenza e formazione.

Scp offre un'ampia gamma di prodotti, dedicati ad aziende di piccole, medie e grandi dimensioni, diversificati a seconda del settore di lavoro occupato dall'azienda. Una piccola azienda artigianale o commerciale avrà delle necessità molto diverse rispetto ad un'azienda medio grande; come del resto un software gestionale pensato per una ditta commerciale non potrà essere ugualmente utile per un'azienda di autotrasporti o che opera nell'edilizia. Per questo motivo Scp ha sviluppato una linea di prodotti molto variegata che possa soddisfare le specifiche richieste da ogni tipo di cliente.

Molta importanza viene data ai prodotti dedicati al Web. La comunicazione di massa del ventunesimo secolo trova quale maggiore canale di diffusione il mondo Web. Tantissime informazioni di qualsiasi tipo vengono condivise e scambiate ed esiste un elevato rischio di accesso da parte di utenti non autorizzati. Diventa fondamentale quindi l'assunzione di strumenti che garantiscono protezione, riservatezza e filtri adeguati.

La ditta garantisce al cliente la massima riservatezza e protezione offrendo:

- Antivirus locali;
- Dispositivi anti-intrusione;
- Filtraggio posta elettronica (antispam);
- Filtraggio dei siti non desiderati ;

Scp, inoltre, fornisce servizi, strumenti e risorse per la creazione di siti Web funzionali attraverso le tecnologie più innovative ed avanzate, fornendo soluzioni personalizzate. Alcuni servizi :

- Analisi e progettazione del sito ;
- Registrazione e mantenimento dei domini ;
- Servizi avanzati di posta elettronica ;
- Servizi di Hosting ;

Alcune applicazioni Web realizzate:

- Portale Web per strutture alberghiere ed agenzie immobiliari ;
- Consultazione dati di magazzino e gestione ordini aggiornati in tempo reale ;
- Raccolta ordini integrata con il sistema aziendale del cliente.

1.2 Presentazione del progetto di stage

Il tirocinio si è svolto a partire dal giorno 02/05/2011 fino al 27/07/2011 ed ha riguardato prevalentemente l'inserimento del progetto "Rivenditori-Tipologie" nell'area di servizio tecnico reparto programmazione. L'obiettivo principale dello stage è stato quello di ricerca ed impostazione per lo sviluppo di un applicativo Web rivolto alla gestione dei dati (e quindi un database) utilizzando il framework Java GWT (GoogleWebToolkit); ciò ha richiesto inizialmente lo studio del framework (struttura e funzionamento) e, una volta acquisite queste conoscenze, è stato necessario apprendere i diversi linguaggi di scripting o di programmazione richiesti al fine di sviluppare il database richiesto.
Il Database richiesto dovrà contenere le "tabelle" rivenditori e tipologie collegate tra loro attraverso una associazione uno a molti. Ognuna di esse deve disporre di specifici attributi : Id, Codice, Descrizione per rivenditori e Id, Codice e Descrizione per tipologia. In seguito verrà spiegato a che cosa si riferisce il nome "tabella", perchè viene utilizzato, che cosa sono gli attributi e che cos'è e cosa implica una relazione tra tabelle.

Nei capitoli successivi verrà descritto nel dettaglio cos'è il framework GWT con pregi e difetti, la differenza tra lo sviluppo di applicazioni Web con e senza GWT, la struttura interna sulla quale è stata sviluppata l'applicazione, come viene resa "Web" e verrà fornito il codice sorgente dell'applicazione esaminando come vengono sfruttate le potenzialità messe a disposizione da GWT dal punto di vista della programmazione. Come si potrà constatare da una successiva analisi, un applicativo risulta composto da due parti: una chiamata "lato server" (server-side), utilizzata per poter usufruire dei servizi messi a disposizione dal server, l'altra indicata come "lato client" (client-side), impiegata per gestire le operazioni che possono essere effettuate dall'utente. Per quanto riguarda il lato client, è stato necessario installare un plugin compatibile per ogni browser utilizzato, potendo garantire in questo modo un corretto e facile utilizzo da parte degli utenti.

L'obiettivo principale dei successivi sottoparagrafi risulta essere quello di illustrare le fondamenta sulle quali si basa il progetto di stage usando un linguaggio esauriente e dettagliato ma allo stesso tempo comprensibile alla maggior parte dei lettori, compresi coloro che hanno poca confidenza con gli argomenti in esame.

1.2.1 Framework

Nella produzione del software, il framework è una struttura di supporto su cui un software può essere organizzato e progettato ed è definito da un insieme di classi astratte e dalle relazioni tra esse. Alla base di un framework c'è sempre una serie di librerie di codice, utilizzabili con uno o più linguaggi di programmazione, spesso corredate da una serie di strumenti di supporto allo sviluppo del software come ad esempio un IDE, un debugger, o altri

strumenti ideati per aumentare la velocità di sviluppo del prodotto finito. Lo scopo di un framework è quello di risparmiare allo sviluppatore la riscrittura di codice già scritto in precedenza per compiti simili. Ad esempio, il tipo di interazione con l'utente offerta da un menù a tendina sarà sempre la stessa indipendentemente dall'applicazione. Ciò permette al programmatore di concentrarsi sulle vere funzionalità dell'applicazione e di creare il contenuto vero e proprio.
Nel progetto in questione il framework utilizzato è GoogleWebToolkit (GWT) utilizzabile con il linguaggio di programmazione Java.

Figura 1.1: Logo di Java

Figura 1.2: Logo GWT

Tale linguaggio, nato nel 1991 in Sun Microsystem da un gruppo di progettisti guidato da James Gosling e Patrick Naughton, è definito linguaggio ad alto livello livello in quanto il programmatore esprime la sequenza di operazioni da compiere senza scendere al livello di dettaglio delle istruzioni macchina ed è stato progettato per essere sicuro e indipendente dalla CPU cioè consente di scrivere programmi trasferibili da un sistema operativo all'altro. La trasferibilità è sicuramente il vero punto di forza : si pensi che lo stesso programma, senza bisogno di modifiche, può essere eseguito indipendentemente su Windows, su Linux o su Machintosh.
Allo stato attuale, Java si è già imposto come uno dei più importanti linguaggi per la programmazione e per l'apprendimento dell'informatica, ma non è perfetto per tre ragioni:

- non è stato curato affinchè fosse veramente semplice da utilizzare per scrivere programmi elementari ed è necessaria una certa dose di tecnicismi per scrivere anche il più semplice dei programmi;

- per utilizzare una nuova versione di Java si deve configurare l'ambiente di programmazione;

- il linguaggio Java è relativamente semplice ma Java contiene un'ampia raccolta di pacchetti di libreria che sono utili per scrivere programmi. Si tratta di pacchetti per la grafica, per la costruzione di interfacce utente, per la crittografia, per la connessione in rete, per l'audio e per molti altri scopi e risulta impossibile conoscere il contenuto di tutti i pacchetti. Ci si limita a sapere quelli che interessano per un particolare progetto.

1.2.2 Database

Definizione. *Un sistema di gestione di basi di dati (Data Base Management System, abbreviato con DBMS) è un sistema software in grado di gestire collezioni di dati che siano grandi, condivise e persistenti, assicurando loro affidabilità e privatezza.*

Definizione. *Una base di dati o Database è una collezione di dati gestita da un DBMS.*

Analizziamo più in dettaglio le definizione appena introdotte.

- Le basi di dati possono essere grandi, nel senso che possono avere delle dimensioni davvero considerevoli. Si può affermare che al momento l'unico limite di grandezza di una base di dati è dato dalle dimensioni del disco sul quale risiede.
- Una base di dati deve essere condivisa, cioè devono poterci accedere diverse applicazioni e diversi utenti; ciò è molto importante perchè se esistessero varie copie degli stessi dati, queste potrebbero essere differenti tra loro; viceversa se i dati sono memorizzati in un calcolatore in modo univoco, non è possibile incorrere in disallineamenti.
- Le basi di dati si definiscono persistenti perchè hanno un tempo di vita limitato a quello delle singole esecuzioni dei programmi che le utilizzano. Al contrario, i dati gestiti da un programma in memoria centrale hanno vita che inizia e termina con l'esecuzione del programma.
- Il DBMS deve garantire affidabilità, ossia deve conservare il contenuto del database anche in caso di malfunzionamenti hardware e software. A questo scopo quindi deve permettere di effettuare backup e recupero dei dati.
- Il DBMS deve anche garantire la privatezza dei dati, cioè deve poter garantire privilegi diversi a seconda dell'utente che accede alla base di dati.

Quando si parla di Database spesso si sente parlare di modello dei dati utilizzato. Un modello di dati è un'insieme di concetti utilizzati per organizzare i dati e descriverne la struttura in modo che essa risulti comprensibile a un elaboratore.

Esistono cinque tipi di modello di dati:

- il modello relazionale, sviluppato da Codd nel 1970, attualmente è il più diffuso e consente di organizzare i dati in un insieme di record a struttura fissa. Una relazione viene spesso rappresentata per mezzo di una tabella, le cui righe rappresentano specifici record e le cui colonne

corrispondono ai campi del record; l'ordine delle righe e delle colonne è irrilevante;

- il modello gerarchico, basato sull'uso di strutture gerarchiche, definito durante gli anni Sessanta e tuttora ampiamente utilizzato;
- Il modello reticolare, basato sull'uso di grafi e sviluppano dopo il modello gerarchico (anni Settanta);
- il modello ad oggetti, sviluppato negli anni Ottanta come evoluzione del modello relazionale, estende alle basi di dati il concetto della programmazione ad oggetti;
- il modello XML, sviluppato negli anni Novanta come rivisitazione del modello gerarchico, in cui dati non devono sottostare ad una struttura logica.

studente

MATR	NOME	CITTA'	C-DIP
123	Carlo	Bologna	Inf
415	Paola	Torino	Inf
702	Antonio	Roma	Log

esame

MATR	COD-CORSO	DATA	VOTO
123	1	7-9-97	30
123	2	8-1-98	28
702	2	7-9-97	20

corso

COD-CORSO	TITOLO	DOCENTE
1	matematica	Barozzi
2	informatica	Meo

Figura 1.3: Esempio di base di dati relazionale

Tutti i modelli precedentemente elencati vengono detti modelli logici per sottolineare il fatto che le strutture utilizzate da questi modelli, pur essendo astratte, riflettono una particolare organizzazione (ad alberi, a grafi, a tabelle o a oggetti). Più recentemente, sono stati introdotti modelli di dati, detti concettuali, utilizzati per descrivere i dati in maniera completamente indipendentemente dalla scelta del modello logico. Il loro nome deriva dal fatto che essi tendono a descrivere i concetti del mondo reale, piuttosto che i dati utili a rappresentarli.

Nelle basi di dati esiste una parte invariabile nel tempo, detta schema della base di dati, costituita dalle caratteristiche dei dati, e una parte variabile nel tempo detta istanza della base di dati, costituita dai valori effettivi. Nell'esempio di Figura 1.3 le tabelle hanno una struttura fissa : la tabella studente ha 4 colonne (dette attributi) che non variano nel tempo e 3 righe (dette tuple) che variano nel tempo in quanto possono essere cancellate, modificate o può avvenire l'inserimento di una nuova tupla.

Modello relazionale e i suoi costrutti

Il modello relazionale si basa sul concetto matematico di relazione, definita come sottoinsieme del prodotto cartesiano D1 x D2 x .. x Dn di n insiemi di valori di tipo elementare, detti domini, ovvero come insieme di n-ple ordinate di valori d1, d2, ..., dn con dk con k appartenente a Dk , per ogni indice k.

Una relazione R definita su D1 x D2 x .. x Dn è un sottoinsieme di D1 x D2 x .. x Dn . La tabella è la rappresentazione grafica normalmente accettata per rappresentare la relazione.

Il grado di una relazione è il numero n delle componenti del prodotto cartesiano (numero di domini), mentre la cardinalità della relazione è il numero delle sue tuple.

L'attributo, come specificato precedentemente, è il termine usato per definire il nome di una colonna della tabella, quindi è il nome dato ad un dominio in una relazione.
Dando quindi delle definizioni informali, possiamo affermare che una relazione è una tabella, un attributo è una colonna, una tupla è una riga, il dominio è il tipo di dato, la cardinalità è il numero di righe, il grado è il numero delle colonne.

Vincoli di integrità e chiavi

La struttura del modello relazionale è sicuramente molto semplice ma allo stesso tempo impone un certo grado di rigidità per garantire che non esistano in un certo momento delle istanze che non rappresentano correttamente il mondo applicativo. Per questo esistono vincoli di integrità che stabiliscono quali sono i valori assumibili dagli attributi di ogni tupla. I vincoli quindi possono essere visti come proprietà che devono essere rispettate dalle istanze perchè queste possano essere considerate valide.

I principali vincoli di integrità sono quelli che specificano:

- quali attributi possono assumere valore nullo.
- quali attributi sono chiave.
- quali attributi sono chiave esterne.

I valori nulli

L'esigenza di ammettere nel dominio di definizione di un attributo il valore nullo deriva dal fatto che, per varie ragioni, può capitare che non sia possibile specificare il valore dell'attributo stesso. Questo accade molto

frequentemente, ad esempio quando non si conosce il valore dell'attributo, oppure quando il valore esiste ma non è disponibile.

Le chiavi

I vincoli di chiave sono i più importanti del modello relazionale e garantiscono l'univocità di ogni istanza di una relazione. Si possono definire tre tipi di chiavi:

- una superchiave è un sottoinsieme di attributi della relazione tale che in nessuna istanza valida della relazione possano esistere due tuple diverse che coincidono su tutti gli attributi superchiave. Una superchiave quindi identifica univocamente ogni tupla;
- una chiave di una relazione è una superchiave minimale, nel senso che se si elimina un attributo, i rimanenti non formano più una superchiave;
- una chiave primaria è una delle possibili chiavi, di solito quella con meno attributi.

E' evidente che una chiave primaria non può ammettere valori nulli nei suoi attributi, altrimenti non garantirebbe l'identificazione univoca di tutte le tuple della relazione.

Le chiavi esterne

Le chiavi esterne sono indispensabili per rappresentare le associazioni nel modello relazione. Il loro scopo è quindi quello di associare ad una tupla di una relazione quella tupla della relazione riferita che ha il valore della chiave primaria uguale al valore della chiave esterna.

In pratica, quando due tabelle vengono associate, la chiave primaria di una tabella viene inserita nell'altra tabella come attributo il cui nome, spesso, coincide con il nome della tabella a cui tale chiave primaria fa riferimento. Il vincolo di chiave esterna viene definito anche vincolo di integrità referenziale o di foreign key.

Modello Entità-Relazione e i suoi costrutti

Il modello Entità-Relazione (nel seguito utilizzeremo l'abbreviazione E-R) è un modello concettuale di dati e, come tale, fornisce una serie di strutture, dette costrutti, atte a descrivere la realtà di interesse. I costrutti principali del modello sono:

- Le entità rappresentano classi di oggetti che hanno proprietà comuni ed esistenza autonoma, hanno un nome che le identifica univocamente

e vengono rappresentate tramite rettangoli; ogni entità deve avere una chiave primaria.

- Le associazioni o relazioni rappresentano legami logici tra due o più entità, hanno un nome che le identificano univocamente e vengono rappresentate con dei rombi.
- Gli attributi descrivono le proprietà elementari di relazioni o entità, possono essere pensati come gli attributi del modello relazionale e vengono rappresentati con una linea unita ad un cerchio vuoto; se il cerchio è oscurato tale attributo è chiave primaria; se più attributi sono chiave primaria allora essi sono uniti da una linea.

Cardinalità delle associazioni

Vengono specificate per ciascuna partecipazione di entità a una associazione e dicono quante volte, un'occorrenza di una entità può essere legata a occorrenze delle entità coinvolte mediante associazione. In linea di principio è possibile assegnare un qualunque numero intero non negativo a una cardinalità di un'associazione. Nella realtà è sufficiente usare solo tre valori: zero, uno e il simbolo N che indica un numero maggiore di uno (viene chiamata anche molti).

Esistono tre tipi di cardinalità delle associazioni:

- uno a uno (1 – 1);
- uno a molti (1 – N);
- molti a molti (N – N).

Traduzione dello schema E-R verso il modello relazionale

Grazie alla realizzazione di uno schema E-R si è raggiunto l'obiettivo di rappresentare accuratamente i dati di interesse dal punto di vista del significato che hanno nell'applicazione. Per ottenere la base per l'effettiva realizzazione dell'applicazione è necessario tradurre lo schema E-R in schema logico, il quale deve essere in grado di descrivere, in maniera corretta ed efficace, tutte le informazioni di interesse presenti nello schema E-R.

Ad ogni relazione con una specifica cardinalità corrispondono precise regole per la traduzione da schema E-R a schema logico come per esempio la chiave primaria viene tradotta con la sottolineatura del nome dell'attributo. Nel nostro progetto siamo al cospetto di una relazione uno a molti tra l'entità Tipologia e l'entità Rivenditore, ognuna delle quali fornita di attributi. Di seguito verrà fornito uno schema E-R completo del progetto di stage.

Figura 1.4: Schema E-R

Lo schema logico associato è del tipo :

Tipologia (Id, Codice, Descrizione)

Rivenditore (Id, Codice, Descrizione, Tipologia)

1.2.3 Riepilogo

L'applicativo Web, rivolto alla gestione dei dati, dovrà contenere le entità Tipologia e Rivenditore con gli eventuali attributi specificati nello schema logico; una volta inserita un'entità si potranno modificare i valori degli attributi (tranne Id che verrà inserito in automatico come vederemo nei capitoli successivi) a proprio piacimento e in casi estremi potrà avvenire l'eliminazione dell'entità con la conseguente rimozione di tutti gli attributi ad essa associati.

Dalla traduzione dello schema E-R verso il modello logico si può notare la presenza dell'attributo Tipologia in Rivenditore. Tale attributo consente di creare una associazione tra le due entità. Secondo specifiche regole imposte dal modello relazionale il valore della chiave esterna deve coincidere con il valore della chiave primaria; nel nostro progetto però la chiave esterna Tipologia non assumerà il valore della chiave primaria ma bensì il valore dell'attributo Codice presente in Rivenditore. Questa scelta è stata decisa dal tutor aziendale in quanto nel mondo del lavoro, spesso, non si segue il modello teorico perchè troppo distante dalla realtà lavorativa e, per venire incontro alle esigenze dei clienti, si è costretti a modificare le regole fornite dalla teoria.

Capitolo 2

Applicazioni Web

Un'applicazione può essere intesa come un programma applicativo.

Definizione. *Un programma è una sequenza di istruzioni e di decisioni che il computer esegue per svolgere un'attività. Un programma indica al computer, nei minimi dettagli, la sequenza di passaggi che sono necessari per eseguire un determinato compito ed è composto da un numero enorme di operazioni elementari, che vengono eseguite dall'elaboratore ad altissima velocità.*

L'espressione Web-application, ovvero applicazione Web, in generale viene usata per indicare tutte le applicazioni che non risiedono direttamente sulle macchine che le usano, ma su server remoti collocati in ogni angolo del pianeta.

In pratica una Web-application, è un programma che non necessita di essere installato nel computer in quanto esso si rende disponibile su un server in rete e può essere fatto funzionare attraverso un normale Web browser (Internet Explorer, Mozilla Firefox, Opera, ecc.) in posizione di client. Risulta evidente l'architettura client-server sulla quale si basano gli applicativi Web: il client, dopo aver instaurato una connessione con il server, invia la richiesta per un servizio; il server dopo aver elaborato i dati necessari rende disponibile al client il servizio richiesto.

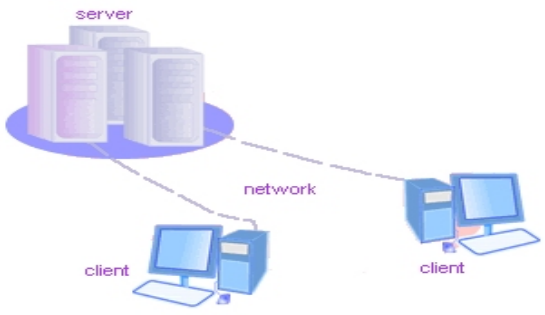

Figura 2.1: Architettura client–server

Troviamo applicazioni Web ovunque. Esempi pratici sono :

- Piattaforme social network come Facebook, Twitter.
- Wikipedia è un brillante esempio di applicazione Web.
- Molti siti Web.
- Motori di ricerca come Google.

Alcuni pregi :

- Gestibilità dei costi: è possibile provare l'applicazione nella versione base e decidere se passare poi a quella a pagamento, che richiede canoni molto abbordabili, con la possibilità di interrompere l'abbonamento nel momento in cui non si abbia più la necessità di utilizzare le funzionalità aggiuntive (e riprendendo il pagamento quando se ne ripresenti il bisogno).
- Facilità d'uso: alcune applicazioni non necessitano di alcuna installazione, ma l'installazione di runtime e plugin è facile, automatica e avviene direttamente dalla rete; in caso di errore o di intoppo può essere ripetuta con risoluzione immediata del problema. Tali piccoli software hanno poi dispositivi di aggiornamento automatico.
- Interoperabilità: le Web-application possono funzionare su qualsiasi sistema operativo. L'unica limitazione può essere data dal tipo di Web browser impiegato: in alcuni casi non è disponibile lo specifico programma aggiuntivo.
- Leggerezza: non avendo installazione o installando al massimo una runtime e un plugin le applicazioni non appesantiscono il sistema operativo.
- Disponibilità: Le applicazioni sono disponibili ovunque vi sia un computer connesso ad Internet. Non è dunque necessario portare con se chiavette USB o altri tipi di dispositivi per la memorizzazione di dati e portable software(programmi che funzionano dalla memoria USB senza bisogno di essere installati sul computer).
- Sicurezza dei dati : salvare i dati sui server è molto più sicuro che salvarli nel disco rigido del proprio computer o in altri supporti. I backup eseguiti dai sistemi online sono continui e altamente affidabili.
- Condivisione e collaborazione : i documenti archiviati negli spazi Web collegati alle applicazioni sono agevolmente condivisibili tra più persone, attraverso procedure di invito via e-mail. In molti casi è possibile inoltre partecipare alla realizzazione dell'elaborato: più persone possono contribuire alla sua realizzazione, utilizzando la medesima applicazione da luoghi diversi, in un atto di stesura collaborativa.

Alcuni difetti :

- Dipendenza dalla connessione a banda larga : l'uso delle applicazioni impone di essere connessi ad Internet e di disporre di una buona quantità di banda. La connessione permanente sta diventando la norma, ma non bisogna dimenticare che in alcune zone d'Italia non è ancora disponibile l'ADSL. E' ovvio, comunque, che eventuali lentezze della rete si ripercuotono negativamente sulle prestazioni delle applicazioni.
- Dipendenza dall'affidabilità del fornitore del servizio: è bene scegliere fornitori di applicazioni unanimemente riconosciuti come affidabili. Interruzioni anche brevi del servizio possono causare gravi danni all'attività. L'affidabilità del fornitore rassicura anche per ciò che riguarda la riservatezza e la tutela della privacy.

2.1 Linguaggi di programmazione

Di seguito sono riportati i linguaggi di programmazione utilizzati per la creazione di applicazioni Web; sono stati separati in due macro categorie, lato Server e lato Client, per definire con maggiore chiarezza il modo con il quale operano e per descrivere gli scopi del loro impiego.

2.1.1 Linguaggi lato client

I linguaggi lato client sono quei linguaggi la cui elaborazione e interpretazione vengono affidate alla macchina dell'utente (client). Di conseguenza una pagina Web può essere visualizzata correttamente solo se il browser, con cui la si sta visitando, supporta tutte le tecnologie utilizzate.

I linguaggi appartenenti a questa categoria sono:

- Html
- CSS
- JavaScript

Html

Html (HyperText Markup Lenguage) è un linguaggio di markup (non di programmazione) utilizzato per descrivere le modalità di impaginazione e visualizzazione grafica, testuale e non, del contenuto di una pagina Web. Viene interpretato dal browser il quale lo elabora e genera la visualizzazione della pagina richiesta.
Un documento HTML inizia con l'indicazione della definizione del tipo di documento (Document Type Definition), la quale segnala al browser l'URL

delle specifiche HTML utilizzate per il documento indicando quali elementi si possono utilizzare e a quale versione di HTML si fa riferimento. Successivamente il documento HTML presenta una struttura ad albero annidato, composta da sezioni delimitate da tag opportuni che al loro interno contengono sottosezioni più piccole, sempre delimitate dai tag.

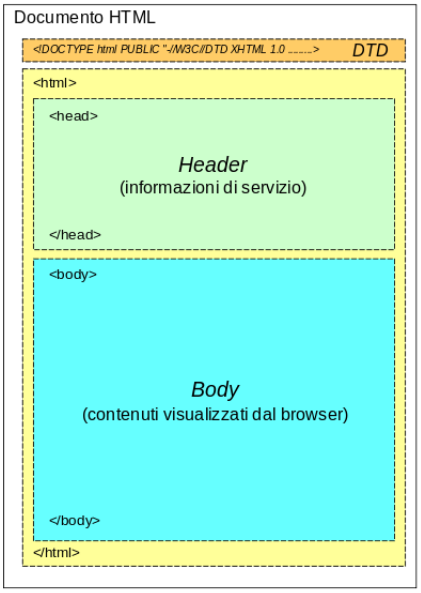

Figura 2.2: Struttura pagina HTML

La struttura più esterna è delimitata dai tag <html> e </html>. Al loro interno sono presenti due sezioni ben distinte:

- la sezione Header, delimitata dai tag <head> </head> che contiene informazioni di controllo non visualizzate dal browser ;
- la sezione Body, delimitata dai tag <body> </body> che contiene il testo, le immagini e i collegamenti che vengono visualizzati dal browser.

CSS

I CSS (Cascading Style Sheet – Fogli di stile a cascata), introdotti a partire dall'HTML 4.0, sono utilizzati per definire le modalità secondo le quali una pagina deve essere visualizzata; se l'HTML descrive il contenuto della pagina, i fogli di stile a cascata si occupano della sua formattazione (colori, caratteri, bordi, ecc . . .).
Esistono diversi vantaggi dovuti al loro utilizzo che sono principalmente:

- riutilizzo dello stesso tema per più pagine Web senza aggiungere nuovo codice;

- separazione del codice relativo al contenuto da quello relativo alla presentazone grafica, facilitando quindi la manutenzione.

L'unione degli elementi della pagina Web con gli stili avviene con l'utilizzo di un markup tag (associa uno stile agli elemti corrispondenti al tag html specificato), di un identificatore (associa lo stile ad un determinato elemento) oppure di una classe (associa lo stile ad un gruppo di elementi che utilizzano la classe specificata).

A ciascuno di questi selettori può essere unito un insieme di proprietà con lo scopo di definire un determinato stile. Qui di seguito si possono trovare elencate le più comuni:

PROPRIETA'	**DESCRIZIONE**
Background-color	Sfondo di un elemento
Text-color	Colore del testo
Text-align	Allineamento orizzontale del testo
Letter-spacing	Spazio tra caratteri
Font-family	Tipo di font
Font-size	Dimensione del font
Font-style	Forma del font (normale, obliqua, italica)
List-style	Tipo di elenco (puntato, numerato)
Border-style	Forma del bordo (linea continua, tratteggiata)
Cursor	Tipo di cursore del mouse
Position	Posizione rispetto alla pagina o all'elemento

Tabella 2.1: Proprietà CSS

JavaScript

JavaScript è un linguaggio di scripting interpretato (non è perciò necessaria alcuna operazione di compilazione), sviluppato per aumentare l'interattività con l'utente nelle pagine Web. L'attività di interpretazione del codice javascript non richiede elevate risorse hardware: questa caratteristica è di rilevante importanza in quanto, essendo un linguaggio client, l'elaborazione deve essere fatta nel minor tempo possibile per non appesantire la pagina e, di conseguenza, renderla poco usufruibile da parte dell'utilizzatore.

Le principali funzionalità di JavaScript sono :

- creazioni di componenti dinamiche all'interno della pagina Web;
- controlo degli eventi generati dall'utente quali il clic del mouse, la pressone di un pulsante, ecc. . . ;

- interazione con i moduli Web compilati, per esempio per il controllo dei valori inseriti;
- lattura e scrittura di cookie per il salvataggio di informazioni nel browser.

Compatibilità Browser

Come detto in precedenza, i linguaggi client-side devono essere interpretati dal browser; purtroppo non tutti i tipi i tipi di browser riescono a comprendere correttamente il codice scritto, rendendo nella peggiore delle ipotesi la pagina Web totalmente illeggibile e di conseguenza inutilizzabile. Durante la scrittura di applicazioni Web risulta doveroso rispettare alcuni standard imposti dal W3C (World Wide Web Consortium) ed è altamente sconsigliato utilizzare un codice appartenente solamente ad alcuni browser.
Ecco una lista dei principali browser la cui compatibiltà è attualmente richiesta:

- Mozzilla Frefox
- Google Chrome
- Opera
- Internet Explorer
- Apple Safari

Tra i linguaggi lato client descritti precedentemente, quello che ha creato più problematiche sotto questo punto di vista è senza ombra di dubbio JavaScript: ciò è dovuto alle attuazioni di JavaScript sviluppate all'interno dei diversi browser.
Per rimediare a questo problema si utilizza la libreria jQuery: è una libreria di funzioni javascript e ha come obbiettivo pincipale quello di rendere le pagine Web cross browser, cioè visualizzabili in modo corretto dalla maggior parte dei browser presenti. La libreria offre inoltre ulteriori servizi con lo scopo di diminuire la scrittura di codice per il raggiungimento di obiettivi quali la gestione degli eventi, la modifica del foglio di stile in modo dinamico, la creazione di effetti o animazioni, ecc. . . , e mette a disposizione un insiemedi funzionalità molto utili e veloci per l'utilizzo della tecnologia Ajax.

Ajax

Ajax (Asynchronous JavaScript and XML) è una tecnologia scritta in javascript per lo sviluppo di applicazioni Web dinamiche dedicate allo scambio

asincrono di informazioni tra il client e il server. La novità principale introdotta da questa tecnologia è proprio l'asincronia: quando si verifica una richiesta da parte del client verso il server non più necessario attendere che venga completata per effettuare altre operazioni.
Come si può vedere nella Figura 2.3 , l'impiego di questa tecnologia provoca la modifica del normale flusso di scambio dei dati previsto dal protocollo HTTP. Nel modello classico, infatti, per ogni richiesta effettuata dal client si attende la risposta del server (la risposta consiste nella pagina Web aggiornata), mentre nel modello con metodo Ajax, il client richiede delle informazioni al server e quest'ultimo risponde con dei dati scritti in formato XML o HTML, che verranno poi utilizzati per aggiornare la pagina Web.

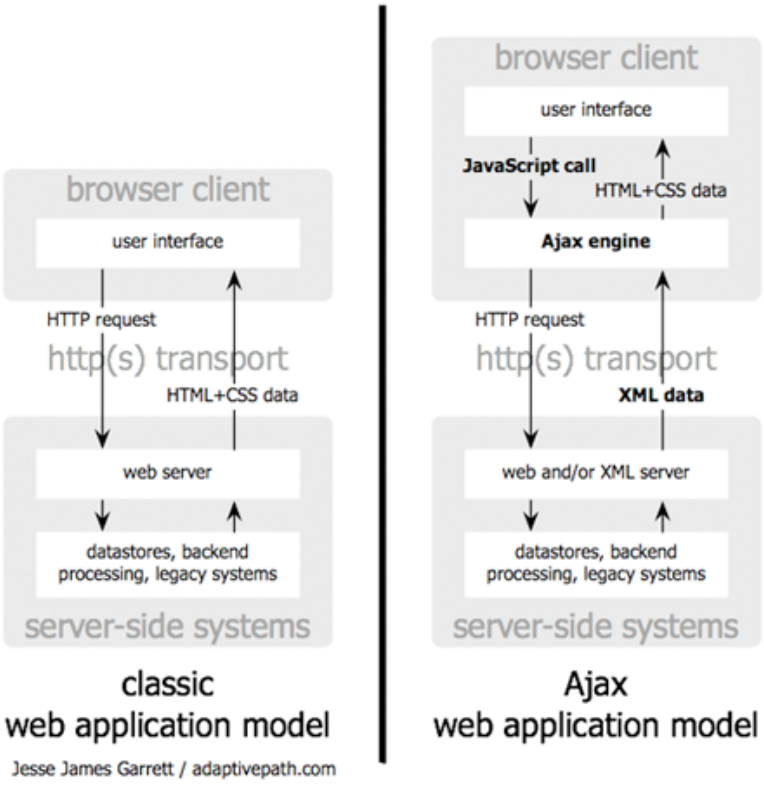

Figura 2.3: Differenza flusso dati Ajax- standard HTTP

I vantaggi derivanti dall'uso della tecnologia Ajax sono :

- minore quantità di traffico scambiato tra client e server: non è necessario attendere il ricaricamento di tutta la pagina ma solamente di una sua porzione;
- maggiore velocità di interazione con l'utente ;
- possibilità di richieste simultanee.

La possibilità di effettuare richieste simultanee costituisce un vantaggio ottenuto grazie al cambiamento del flusso dei dati. Questa funzionalità non

è disponibile nel modello classico in quanto, durante il tempo d'attesa per l'aggiornamento della pagina, non sono concesse ulteriori interazioni dell'utente con essa.
Nella Figura 2.4 si può osservare la maggiore efficienza temporale di Ajax; attraverso questa tecnologia è possibile continuare a utilizzare i servizi messi a disposizione per l'utente anche nell' intervallo tra una richiesta e l'aggiornamento della pagina, cosa non possibile con il modello classico.

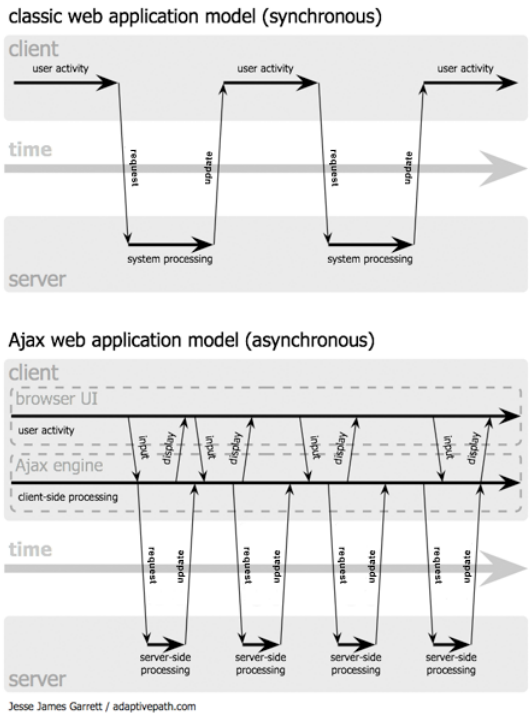

Figura 2.4: Ajax – diagramma temporale

2.1.2 Linguaggi lato server

I linguaggi lato server sono quei linguaggi che vengono elaborati dal server, il quale mette a disposizione un insieme di servizi utili per l'acquisizione di informazioni o funzionalità non disponibili nella macchina dell'utente. Ad ogni richiesta effettuata dall'utilizzatore verrà poi spedito il risultato prodotto dal server.

Esempi di linguaggio server-side possono essere:

- PHP (Pre Hypertext Processor) è un linguaggio open source completo di scripting, sofisticato e flessibile, che può girare praticamente su qualsiasi server Web, su qualsiasi sistema operativo (Windows o Unix/Linux, ma anche Mac, AS/400 e altri), e, soprattutto, consente di interagire con i principali tipi di database (MySql, PostgreSql, Sql

Server, Oracle, SyBase, ecc). PHP Permette di raccogliere dati, generare contenuti dinamici e interagire con i cookie e con altri servizi utilizzando i principali protocolli (IMAP, SNMP, NNTP, POP3).
Tra le applicazioni principali ricordiamo l'autenticazione degli utenti, la gestione di sistemi di conferenza, la gestione di template, la gestione di file XML, la creazione dinamica di immagini (tramite la libreria GD) e di documenti PDF (tramite la libreria PDF). La sua caratteristica principale è il supporto di un vasto numero di database ed una velocità di sviluppo veramente notevole. Azioni come interrogazioni di database o utilizzo di socket TCP possono essere gestite in modo molto semplice, almeno in confronto agli altri linguaggi.

- ASP (Active Server Pages) ovvero pagine attive lato Server. Una pagina ASP è un kit di istruzioni eseguibili dal server, il quale deve necessariamente essere un server Microsoft NT. Questo kit di istruzioni è integrato col codice HTML e permette di eseguire operazioni di manipolazioni dei dati e dei file inclusa la scrittura sul disco del server. La programmazione in ASP è la più usata per la realizzazione di siti e-commerce e deve essere scritta in uno dei seguenti linguaggi: VBScript , Jscript (diverso da JavaScript), PerlScript (diverso da Perl), Python.
  Dei quattro il linguaggio sicuramente più usato e supportato dai server è VBScript che è il linguaggio di riferimento per ASP. NT,ASP e VBSCRIPT, i quali sono tutti prodotti Microsoft perfettamente integrati.

2.2 JavaEE

Java EE (Enterprise Edition) è una piattaforma costituita da un insieme di specifiche per lo sviluppo di applicazioni per il lato server utilizzando il linguaggio Java. Rispetto alla versione Java SE (Standard Edition) vengono aggiunte delle librerie per applicare nuove caratteristiche quali :

- Java Servlet: insieme di API (Application Programming Interface) per lo sviluppo di Servlet ;
- Java Server Pages (JSP): tecnologia per la creazione di pagine Web dinamiche ;
- Java Persistence : framework utilizzato per il controllo dei dati mediante l'utilizzo delle classi java.

2.2.1 Application Server

Tra i più importanti elementi di un' applicazione Web, vi è l' Application Server, il quale, implementando le specifiche JEE, rappresenta uno strato software che fornisce i servizi e le interfacce per la comunicazione tra tutte

le componenti e permette all' applicazione di funzionare. Le applicazioni più semplici, quelle cioè che non possiedono una vera e propria logica di business, dovrebbero utilizzare Application Server con solo la proprietà di Web-Container (che ha la funzione principale di presentazione dei dati): sono chiamati Web Server e sono consigliati in queste circostanze in quanto, le risorse richieste sono di molto inferiori. Tra i principali componenti Java gestiti da un Web-Container vi sono Servlet e JSP. La scelta di quale Application Server utilizzare a sostegno delle proprie applicazioni è influenzata da numerosi fattori tra cui:

- Tecnologie preesistenti: alcuni case software dei principali DBMS commerciali propongono i propri Application Server. Optare per tale soluzione, solitamente, facilita l'integrazione tra le due componenti aumentando l'efficienza.
- Costo delle licenze: questo aspetto è chiaramente importante; normalmente però i costi delle licenze dei soli Application Server sono esigui perché legati ai costi del DBMS.
- Supporto a pagamento: un fattore importante soprattutto per le grandi aziende, che possono permettersi di sostenerne i costi .
- Supporto della rete: Application Server molto diffusi comportano una maggior quantità di informazioni in rete, con una maggior possibilità di reperire soluzioni a costi irrisori. E' il fattore principale delle piccole aziende.
- Portabilità: uno degli aspetti fondamentali del successo di Java per gli application Server JEE. Questa tecnologia permette di integrare componenti molto diverse tra loro, favorendo economie di scala.

I principali Application Server presenti nel panorama delle applicazioni Web e che implementano lo standard JEE sono:

- JBOSS: pioniere degli Application Server OpenSource, funziona sia da EJB-Container che da Web-Container grazie all'integrazione di Apache Tomcat.
- Apache Tomcat: funziona solo da Web-Container. Gestisce Java Servlet e Java Server Pages.
- Sun GlassFish Enterprise Server: essendo gestito direttamente da Sun implementa nel modo più completo e fedele le specifiche dello standard JEE; le versioni seguono di pari passo i nuovi rilasci delle specifiche JEE.
- BEA Weblogic: soluzione commerciale proposta da Oracle.
- IBM WebSphere.

2.2.2 Servlet

Definizione. *La servlet è una componente Web gestita da un contenitore, basata su tecnologia Java che genera pagine Web dinamiche. Come altri componenti basati sul linguaggio Java, il codice della servlet è presente all'interno di una classe e, una volta compilato, può essere caricato in qualsiasi server Web che supporta tale tecnologia.*

Le pagine Web generate in modo dinamico contengono al loro interno un codice sorgente che utilizza sia il linguaggio html sia quello java, per esempio per effettuare i controlli per l'autenticazione degli utenti, per sottoporre richieste alla base di dati, ecc.... Il difetto principale deriva quindi dall' unione nella stessa struttura di due tipologie diverse di linguaggio: html, dedicato alla visualizzazione delle informazioni, e java, rendendo complicate le operazioni di manutenzione e riducendo la leggibilità del codice stesso.
Per risolvere questo problema ci si serve delle Servlet: in questo modo gran parte del codice Java viene spostato in opportune classi Java, separando così il codice dedicato alla rappresentazione dei contenuti Web da quello relativo alle operazioni di gestione.

La struttura di base di una Servlet è rappresentata di seguito:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
  HttpServletResponse response)throws ServletException,
  IOException
  {
  //Overriding del metodo doGet
  }
  public void doPost(HttpServletRequest request,
  HttpServletResponse response)throws ServletException,
  IOException
  {
  //Overriding del metodo doPost
  }
}
```

Descriviamo ora il ciclo di vita di una Servlet. Queste fasi sono controllate dal Web-Container, in particolare, quando una richiesta viene indirizzata verso una servlet:

1. Se non esiste un'istanza della Servlet il Web-container:
   - **a)** Carica la classe Servlet
   - **b)** Crea un'istanza della classe
   - **c)** Inizializza la Servlet invocando il metodo init. Il metodo può essere sovrascritto passando come parametro un oggetto della classe ServletConfig al fine di permettere alla Servlet di accedere a dati di configurazione come password di file, cookies, parametri di

accesso al database o dati provenienti da richieste precedenti. Una Servlet che, per qualche motivo, non è in grado di terminare l'inizializzazione lancia l'eccezione UnavailableException

2. Il Web-container invoca il metodo service che identifica il tipo di richiesta del protocollo HTTP (GET, POST, PUT, DELETE) e invoca il corrispondente metodo Servlet doGet(), doPost(), doPut(), doDelete(). Altri metodi esistenti sono doOptions() e doTrace().

3. Se il container deve eliminare l'istanza della Servlet, sia per una scelta dell'amministratore dell'applicazione sia perché da tempo inutilizzata, chiama il metodo destroy, il quale effettua tutte le operazioni di chiusura tra cui la dismissione delle connessioni ai database.

All'interno di un server Web possono girare più servlet e sono gestite da un Servlet Container (o Servlet Engine) , un software capace di amministrarle rispettando le specifiche imposte dal linguaggio Java. Quando il server riceve una richiesta di una pagina Web, il Servlet Engine esegue un'analisi sintattica della pagina e quando si imbatte nel codice java lo interpreta sostituendolo con un codice html concatenando e mettendo insieme i frammenti di codice per poi restituire la pagina in formato html al client, il quale grazie al browser la interpreta e la visualizza nello schermo dell'elaboratore.

2.2.3 JSP

JavaServer Pages è una tecnologia sviluppata in Java per la creazione di pagine Web dinamiche. Per dinamiche s' intende che il contenuto delle pagine può essere creato includendo risultati di elaborazione, dati prelevati dal database, oppure parametri inseriti dall'utente. Le pagine JSP (.jsp) sono delle pagine Web composte da due tipi di codice: uno statico (html) e uno dinamico, formato quest'ultimo da elementi jsp che hanno il compito di generare il contenuto in modo dinamico.

Questo linguaggio di programmazione è strettamente legato alla tecnologia delle Servlet in quanto le pagine JSP vengono tradotte automaticamente da un compilatore JSP in servlet. Tutto ciò è reso possibile dalla presenza nel server Web di un motore JSP che, quindi, risulta essere indispensabile.

Le JSP si basano su tecnologia Java ereditandone i vantaggi garantiti dalla metodologia object oriented e dalla quasi totale portabilità multipiattaforma. Essere object oriented significa anche avere una predisposizione al riuso del codice: grazie ai Java Bean (componenti presenti nel Web- Container che utilizzano tecnologia Java)si possono includere porzioni di codice che semplificano l'implementazione di applicazioni, anche senza approfondite conoscenze di Java, e ne rendono più semplice la modifica e la manutenzione.

Gli elementi JSP possono essere di due tipi: direttive e script. Le direttive sono un insieme di comandi necessari per la corretta interpretazione della pagina da parte del motore JSP, mentre gli script sono elementi che contengono al proprio interno codice sorgente Java esono utilizzati per generare il contenuto dinamico oppure per effettuare dei controlli.

2.2.4 ORM

L'ORM (Object-Relational Mapping) è una tecnica di programmazione che fornisce la persistenza dei dati, il cui concetto fa riferimento alla caratteristica dei dati di sopravvivere all'esecuzione del programma che gli ha creati. Senza questa capacità i dati vengono salvati solo in memoria RAM e verranno persi allo spegnimento del computer. Con la parola mappping intendiamo il processo di trasformazione dei dati da semplici oggetti descritti nelle classi Java e veri e propri dati presenti nel database.

Per stabilire come questo mapping debba avvenire, alcuni servizi di ORM fanno uso di metadati di configurazione come ad esempio le annotazioni (presentate con il simbolo @ seguito dal nome dell' annotazione).

I principali vantaggi derivanti dall'uso di tale tecnica sono :

- risoluzione del problema dell'impedance mismatch che rende incompatibile il progetto orientato agli oggetti ed il modello relazionale sul quale è basata la maggior parte dei sistemi di gestione dei database relazionali (RDBMS);
- elevata portabilità rispetto alla tecnologia RDBMS utilizzata : cambiando DBMS non è necessario riscrivere il codice per implementare la persistenza dei dati;
- riduzione della qantità di codice da redigere: l'ORM maschera dietro semplici comandi le complesse attività di creazione, prelievo, aggiornamento ed eliminazione dei dati (CRUD Create Read Update Delete).

Java Persistence API

Java Pesistence API (JPA) è lo standard Java che fornisce agli sviluppatori un servizio ORM . Java Persistence è struturato in 4 aree:

1. Java Persistence API : pacchetto contenente l'insieme di metodi e classi utilizzabili;
2. Java Persistence Query Lenguage (JPQL) : linguaggio simile a SQL adoperato per realizzare interrogazioni o aggiornamenti delle entità del database;

3. Java Persistence Criteria API : insieme di operazioni impiegate per l'interrogazione o l' aggiornamento di dati; la differenza con JPQL è che queste operaioni vengono eseguite sugli oggetti e non sul database;

4. Metamodel API: area usata per la creare la relazione tra la classe ed una entità del database.

In JPA il mapping è dichiarativo, tutte le operazioni sui dati (CRUD) sono fornite dall'interfaccia Entity Manager che è la più importante visto che costituisce un ponte tra il mondo orientato agli oggetti (Java) e quello relazionale (database). Le principali annotazioni sono :

ANNOTAZIONI	**DESCRIZIONE**
@Entity	Rende persistente un oggetto Java (definito tramite una classe).
@Table	Indica il nome della tabella visualizzata nel database. Se non introdotto il nome della tabella coincide con il nome della classe Java.
@Id	Rappresenta univocamente una entità nel caso in cui l'identificatore sia costituito da un solo attributo.
@IdClass	Utilizzato nel caso in cui l'identificatore è composto da più attributi.
@GeneratedValue	Genera i valori di una chiave primaria; ne sistono di tre tipi
@OneToOne	Indica la relazione uno ad uno.
@OneToMany	Indica la relazione uno a molti.
@ManyToMany	Indica la relazione molti a molti.
@JoinColumn	Indica il nome della chiave esterna e il nome nella chiave primaria a cui fa riferimento.
@NotNull	Indica che l'attributo non può essere nullo.

Tabella 2.2: Annotazioni JPA

Hibernate

Hibernate è una piattaforma middlewere open source per lo sviluppo di applicazioni JAVA che fornisce un servizio l'Object Relational Mapping che semplifica e automatizza la gestione della corrispondenza tra il modello ad oggetti delle nostre applicazioni e quello relazionale dei database server ai quali queste ultime comunemente si interfacciano.
Il cuore di Hibernate è di circa 50.000 righe di codice tutte open e distribuite

con licenza LGPL, supporta numerosi Database come Oracle, DB2, PostgreSQL, MySQL ed il suo progetto è legato al famoso Application Server JBOSS.

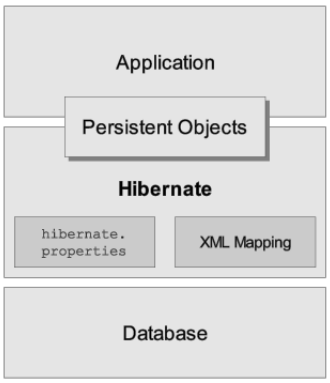

Figura 2.5: Architettura di Hibernate

Hibenate si pone tra il modello ad oggetti che si ricava dalla logica della nostra applicazione e l'insieme delle tabelle, chiavi primarie e vincoli relazionali presenti nel database relazionale. Permette di rendere facilmente persistenti le classe Java senza eccessivi vincoli ed uno o più file di configurazione , la cui estensione è hbm, stabiliscono la corrispondenza tra oggetti della nostra applicazione e le tabelle del database. Ogni oggetto, descritto dalla classe Java, viene progettato in fase di modellazione del dominio applicativo e per poter essere gestito da Hibernate deve semplicemente aderire alle fondamentali regole dei più pratici javabeans: metodi pubblici getXXX e setXXX per le proprietà persistenti.

La Figura 2.5 rappresenta il modo con cui Hibernate usa il database e i dati di configurazione per fornire servizi di persistenza (e oggetti persistenti) all'applicazione.
Uno dei concetti fondamentali dell'architettura di Hibernate è quello di classi persistenti, ovvero quelle classi che in un'applicazione implementano le entità del database (ad esempio Customer e Order in una applicazione di e-commerce). Hibernate funziona meglio se queste classi seguono alcune semplici regole, conosciute anche come il modello di programmazione dei cari vecchi oggetti java (in inglese e nel seguito si usa l'acronimo POJO che sta per Plain Old Java Object). Un POJO è più o meno come un JavaBean, con proprietà accessibili tramite metodi getter e setter (rispettivamente per recuperare e impostare gli attributi).

Un esempio di POJO:

```
public class Cat {
  private String id;
  private String name;
  private char sex;
  private float weight;
  public Cat() {
```

```
  }
  public String getId() {
  return id;
  }
  private void setId(String id) {
  this.id = id;
  }
  public String getName() {
  return name;
  }
  public void setName(String name) {
  this.name = name;
  }
  public char getSex() {
  return sex;
  }
  public void setSex(char sex) {
  this.sex = sex;
  }
  public float getWeight() {
  return weight;
  }
  public void setWeight(float weight) {
  this.weight = weight;
  }
}
```

Il mapping, a differenza di Jpa dove vengono usate annotazioni, viene definito in un documento XML progettato per essere leggibile e modificabile a mano; esistono un certo numero di strumenti per generare il documento di mapping ma è possibile effettuarlo anche attraverso un semplice editor di testo.
Ecco il file XML di mapping della classe descritta precedentemente:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate␣Mapping␣DTD␣2.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping package="eg">
<class name="Cat" table="CATS" discriminator-value="C">
<id name="id" column="uid" type="long">
<generator class="hilo"/>
</id>
<discriminator column="subclass" type="character"/>
<property name="birthdate" type="date"/>
<property name="color" not-null="true"/>
<property name="sex" not-null="true" update="false"/>
<property name="weight"/>
<many-to-one name="mate" column="mate_id"/>
<set name="kittens">
<key column="mother_id"/>
<one-to-many class="Cat"/>
</set>
</class>
<class name="Dog">
<!- qui potrebbe stare il mappaggio per la classe Dog ->
</class>
</hibernate-mapping>
```

I tag principali di un file XML sono:

- Doctype: indica l'effettivo DTD utilizzato che si trova nell'URL indicato.

- Hibernate-Mapping: specifica le tabelle a cui si fa riferimento nel mapping e vengono dichiarate le caratteristiche delle classi persistenti.

- L'elemento Hibernate-Mapping che si compone di vari sottoelementi e attributi opzionali.

DAO

Definizione. *Use a Data Access Object (DAO) to abstract and encapsulate all access to the data source. The DAO manages the connection with the data source to obtain and store data.*

L'accesso a sorgenti di dati implica la conoscenza e l'utilizzo delle relative modalità di accesso. Questo in applicazione distribuite causa un dipendenza tra la logica di business (Servlet, pagine JSP, ...) e la logica di accesso ai dispositivi di persistenza come database relazionali (RDBMS), file XML, ecc ....
L'intento del pattern DAO (Data Access Object) è di disaccoppiare la logica di business dalla logica di accesso ai dati. Questo si ottiene spostando la logica di accesso ai dati dai componenti di business ad una classe DAO rendendo gli EJB(Enterprise JavaBeans) indipendenti dalla natura del dispositivo di persistenza. Questo approccio garantisce che un eventuale cambiamento del dispositivo di persistenza non comporti modifiche sui componenti di business. Inoltre legando il componente EJB ad un particolare tipo di data repository, ne si limita il riuso in contesti differenti.

L'idea del pattern DAO si basa sulla possibilità di concentrare il codice per l'accesso al sistema di persistenza in una classe che si occupa di gestire la logica di accesso ai dati. Ad esempio nel caso di un accesso a DBMS si inserisce nel DAO tutto il codice per effettuare operazioni di creazione modifica aggiornamento e cancellazione dei dati presenti nel database.

JDO

Java Data Object (JDO) è un'interfaccia basata sulla definizione di persistenza degli oggetti per il linguaggio Java, che descrive l'archiviazione, l'interrogazione e il recupero di oggetti dalle basi di dati.

JDO è estremamente trasparente nella mappatura degli oggetti tra codice applicativo e database, permettendo quindi di archiviare direttamente gli oggetti model di Java sulla propria base di dati.

I vantaggi derivanti dall'utilizzo di JDO sono numerosi: innanzitutto abbiamo una notevole portabilità, cioè le applicazioni scritte con le API di JDO possono essere eseguite su piattaforme multiple senza dover essere ricompilate o dover sostituire parte del codice sorgente. Anche i Metadati, che

descrivono il comportamento della persistenza all'esterno del codice sorgente Java includendo le funzioni di mapping O/R, sono portabili. Un altro aspetto fondamentale è la totale indipendenza dal database: le applicazioni scritte con le API di JDO sono indipendenti dal database sottostante.
Inoltre, i dettagli della persistenza vengono delegati all'implementazione del JDO e abbiamo un' ottimizzazione dei modelli dei dati in accesso con un conseguente miglioramento delle prestazioni e una maggiore facilità d'uso. Ciò comporta che i programmatori possono concentrarsi sul loro dominio di oggetti model senza più doversi preoccupare dei dettagli della persistenza.
Ultimo vantaggio, ma non per ordine di importanza, è l'integrazione con EJB: le applicazioni possono trarre vantaggio dalle funzionalità di EJB come la gestione remota dei messaggi, la coordinazione automatica e distribuita delle transazioni e la sicurezza usando lo stesso dominio di oggetti model in tutto l'ambiente di sviluppo.

Le specifiche JDO definiscono una serie di interfacce che sovraintendono all'accesso delle funzionalità. Queste sono:

- PersistenceManagerFactory: restituisce oggetti di tipo PersistenceManager, ed è il punto di ingresso per il quale i provider di servizi JDO devono offrire implementazione;
- PersistenceManager: interfaccia implementata dai gestori del ciclo di vita degli oggetti persistenti, che permette di creare e rimuovere oggetti, effettuare la manipolazione di opzioni relative alle transazioni ed accedere alle query;
- PersistenceCapable: interfaccia di marcatura che tutti gli oggetti che devono essere resi persistenti devono implementare;
- Transaction: per ciascun PersistenceManager esiste un oggetto Transaction che gestisce il contesto transazionale entro il quale si sta eseguendo l'operazione di persistenza di un oggetto. In ambienti JEE questa interfaccia permette di accedere alle opzioni transazionali, mentre è il container a gestire la transazione. In ambienti non JEE l'interfaccia rappresenta un vero e proprio servizio transazionale, che deve essere invece implementato dal fornitore del servizio JDO;
- Query: consente di specificare una query di selezione sulla base dati degli oggetti persistenti. JDO fornisce supporto per un ottimo linguaggio object-oriented per le query chiamato JDOQL.

Le specifiche JDO inoltre definiscono 10 stati in cui l'oggetto coinvolto può passare (sette obbligatori e tre opzionali), in funzione delle diverse chiamate ed operazioni che è possibile eseguire. Questi sono :

- Transient: un oggetto è transiente quando non è considerato a livello JDO; esso diventa persistente (Persistent-new) nel momento in cui viene reso persistente dalla chiamata PersistenceManager.makePersistent()

oppure nel caso in cui venga referenziato da un campo persistente di un oggetto persistente, quando quest'ultimo viene reso persistente.

- Persistent: (-new) l'istanza è stata appena resa persistente e gli viene associata una identità; (-dirty) una istanza persistente, ancora non persistente, di cui sono stati cambiati i valori degli attributi; (-clean) un oggetto persistente, le cui copie in memoria e sulla base dati sono perfettamente allineate; (-deleted): un oggetto che rappresenta una specifica istanza persistente e che è stato cancellato nella transazione corrente; (-new-deleted) un oggetto appena creato e cancellato dalla transazione corrente; (-nontransactional) un oggetto che rappresenta una specifica istanza persistente i cui valori sono caricati ma non consistenti con lo stato della transazione .

- Hollow: una istanza di un oggetto persistito con JDO, ma i suoi attributi non sono ancora stati caricati dalla base dati.

- Transient: (-clean) oggetto transiente inserito in un contesto transazionale i quali valori non sono stati cambiati; (-dirty) oggetto transiente inserito in un contesto transazionale i quali valori sono stati cambiati .

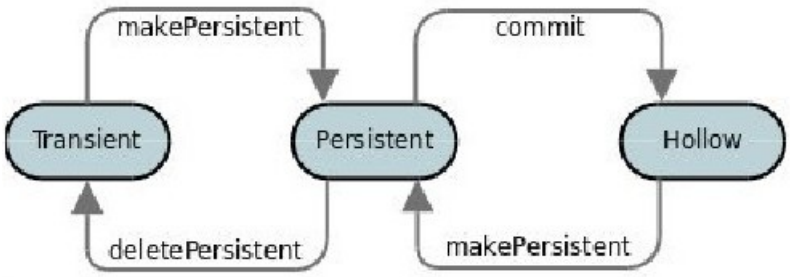

Figura 2.6: Esempio di stati in cui si può trovare un oggetto

2.2.5 Java Bean ed Enterprise Java Bean

Queste componenti sono elementi importantissimi per un'applicazione complessa che utilizzi tecnologie Java, non solo per quelle orientate al Web. Un JavaBean ha la funzione principale di contenere e manipolare le informazioni ed è rappresentata da una classe Java con determinati requisiti:

- lo stato dell'istanza deve avere proprietà accessibili solo all'interno della classe stessa (visibilità privata);

- devono esservi metodi pubblici specifici per poter ottenere (getNomeAttributo()) e manipolare (setNomeAttributo(valore)) le informazioni;

- il suo costruttore non deve avere argomenti;
- può comunicare i propri cambiamenti di stato con altre componenti mediante la generazione di eventi;
- l'istanza deve essere serializzabile (implementando l'interfaccia Serializable).

Di seguito, è riportato un semplice esempio d'implementazione di un JavaBean per la raccolta delle informazioni di un utente in navigazione all'interno di una generica applicazione. La proprietà pagineViste rappresenta il numero di pagine viste dall'utente durante la navigazione; la proprietà user, invece, identifica la login dell'utente.

```
public class InfoUtente implements java.io.Serializable{
  private String user;
  private int pagineViste;
  // Costruttore senza argomenti.
  public void InfoUtente()
  { }

  //Metodi di accesso alle proprieta'
  public String getUser()
  { return this.user;
  }

  public void setUser(String user)
  { this.user = user;
  }

  public int getPagineViste()
  { return this.pagineViste;
  }

  public void setPagineViste(int pagine)
  { this.pagineViste = pagine;
  }
}
```

Gli Enterprise JavaBean non sono altro che l'equivalente della tecnologia JavaBean in ambito enterprise ed implementano la logica di business dell' applicazione. L'introduzione di questo tipo di tecnologia nel lato server fu introdotta per risolvere i seguenti problemi:

- Persistenza dei Dati
- Gestione delle transazioni
- Controllo della concorrenza
- Standard di sicurezza
- Integrazione di componenti software, anche di natura diversa
- Invocazione di procedure remote (Remote Procedure Call)

- Fornire servizi sul Web

Vi sono 3 tipi di EJB:

- EJB di Sessione o Session EJB : la vita di questo tipo di oggetto è legata alla ri-chiesta di utilizzo da parte del client, poiché dopo l'utilizzo viene elaborata la sua eliminazione. Per questo motivo, un Session EJB non è un elemento condiviso con altri Client ed ha, generalmente, un ciclo di vita limitato. Il suo scopo principale è quello di fornire un'interfaccia semplice per tutti i client che richiedono di accedere a determinate informazioni;

- EJB di Entità o Entity EJB: forniscono la caratteristica di persistenza dei dati. Questi oggetti inglobano i dati sul lato server e possono essere condivisi da client diversi. Nel loro utilizzo più frequente, inglobano i dati di un database tramite oggetti Java, grazie ai quali è possibile manipolarli più agevolmente all'interno delle proprie applicazioni. Sia gli EJB di sessione sia di entità hanno un funzionamento sincrono, perciò il client che li invoca deve attendere la terminazione della loro elaborazione potendo, poi, continuare la propria esecuzione. Fino alla versione 2 dello standard EJB, la gestione della persistenza veniva suddivisa in due tipologie:

  1. persistenza gestita dall'EJB-Container: i dati relativi ad un oggetto vengono memorizzati nella base dati dall'EJB-Container che si occupa anche della loro estrazione ottimizzata;
  2. persistenza gestita da un bean: il meccanismo è delegato allo sviluppatore, che implementa nel bean il sistema di memorizzazione e recupero dei dati.

  Dalla versione 3 sono state introdotte nuove API per la gestione della persistenza, chiamate JavaPersistence API (JPA) che, mediante il servizio ORM, denotano il processo di mapping dei dati, fra gli oggetti e le tabelle di un database.

- EJB pilotati da messaggi o Message Driven EJB (MDB): sono gli unici EJB ad avere comportamento asincrono. Il client che intende invocare queste componenti, deve essere in grado di pubblicare o accodare messaggi su una coda, i quali vengono raccolti dall' application server e forniti ad un MDB. Una volta ricevuto il messaggio, l'MDB può invocare in modo sincrono altri tipi di EJB, attendere i risultati di ritorno ed accodare un messaggio di fine elaborazione al client ; tutto questo con la possibilità da parte del client di proseguire la sua esecuzione.

2.3 Pattern MVC e MVP

Un' importante fase nella progettazione di applicazioni software è quella della scelta di un'opportuna architettura, che definisce le linee guida allo sviluppo del progetto. Definire bene tale processo è utile sia per standardizzare il modello di sviluppo del progetto corrente, sia per le applicazioni future.

L'impiego di un adeguato pattern porta numerosi vantaggi tra cui:

1. Incrementa il riutilizzo del codice.
2. Facilita il lavoro in team e la pianificazione del progetto, dividendo quest'ultimo in componenti indipendenti delegabili a gruppi di lavoro differenti.
3. Aiuta la manutenzione del codice.
4. Aumenta la flessibilità delle applicazioni e incrementa della scalabilità.

Pattern MVC

Il modello più utilizzato in ambito delle applicazioni orientate al Web (ma non solo) è il pattern Model-View-Controller (MVC). Fu introdotto per la prima volta nella progettazione del linguaggio SmallTalk, nel 1992. Questo pattern permette una maggiore strutturazione del codice, con un aumento della manutenibilità software e una suddivisione dell' applicazione in sottosistemi. Per comprendere la filosofia del modello, in Figura 2.7, è riportata la schematizzazione delle tre componenti in cui si divide.

Le componenti sono:

- la vista (view) dove viene gestita la presentazione dei dati. E' rappresentata da tutto quel codice di presentazione che permette all'utente, ad esempio tramite un browser, di operare le richieste. All' interno di questo livello lavorano sia programmatori che grafici che curano la parte estetica della presentazione;
- il modello (model) che rappresenta e gestisce i dati, tipicamente persistenti su database;
- il controllore (controller) che dirige i flussi di interazione tra vista e modello. Nello specifico, intercetta le richieste HTTP del client e traduce ogni singola richiesta in una specifica operazione per il model; in seguito può eseguire lui stesso l' operazione oppure delegare il compito ad un'altra componente. Inoltre, seleziona la corretta vista da mostrare al client ed inserisce, eventualmente, i risultati ottenuti.

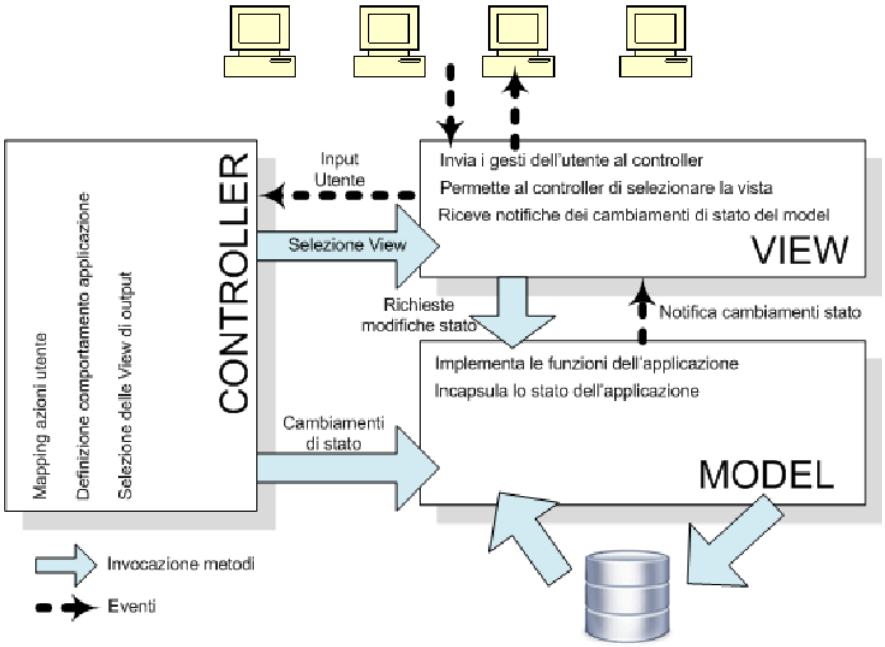

Figura 2.7: Pattern MVC

Pattern MVP

Il pattern MVP(Model View Presenter) è una variazione del pattern Model View Control (MVC) che, in maniera del tutto simile, separa la parte di gestione dei dati di un'applicazione dalla loro visualizzazione e manipolazione attraverso l'interfaccia utente.

Le prinicipali differenze tra MVP e MVC sono :

- La view nel MVP è aggiornata direttamente dal presenter ed è quindi meno legata al model ;
- MVP consente una migliore dello unit testing dell'applicazione visto che si accede alla view solo attraverso un'interfaccia ;
- tranne casi particolari con il MVP si ha una relazione uno-ad-uno tra view e presenter, il che semplifica di molto il flusso dei dati;
- con MVC il controller può essere condiviso da più view ed è il responsabile della selezione dei dati da visualizzare.

La prima implementazione del pattern MVP risale al 1979, il suo nome originario era Thing Model View Editor e durante gli anni si è evoluto fino ad arrivare alla versione attuale che comprende due implementazioni : Supervising Controller e Passive View.

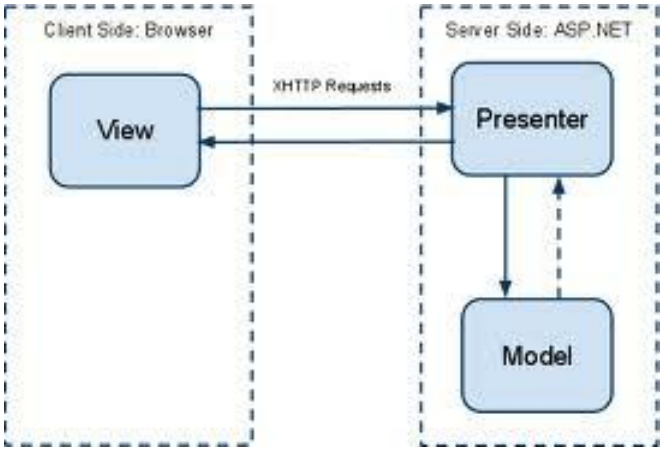

Figura 2.8: Pattern MVP

Nel primo tipo di implementazione la view conosce il modello e generalmente è connessa ad esso tramite un meccanismo di data binding dichiarativo che permette l'aggiornamento dei dati da visualizzare:

- L'utente interagisce in qualche modo con la view.
- La view notifica notifica al Presenter l'interazione.
- Il Presenter opera sul Model eventualmente modificandone lo stato rispetto all'interazione notificata dalla view.
- La view viene aggiornata in base ai nuovi dati esposti dal Model tramite un meccanismo di data binding.
- Il Presenter aggiorna i dati dellla view che non possono essere agganciati a quest'ultima tramite il binding dichiarativo.

Nel secondo tipo di implementazione la view non conosce il modello ed il presenter è l'unico responsabile dell'aggiornamento dei dati visualizzati:

- L'utente interagisce in qualche modo con la view.
- La view notifica notifica al Presenter l'interazione.
- Il Presenter opera sul Model eventualmente modificandone lo stato rispetto all'interazione notificata dalla view.
- Il presenter aggiorna la view in base ai nuovi dati esposti da model.

Le componenti sono :

- La vista (view) è un'interfaccia responsabile della visualizzazione dei dati e delle informazioni, raccoglie gli input dell'utente e mai la manipolazione dei dati avviene in maniera diretta ma sempre attraverso

un'interfaccia. Questo tipo di approccio consente di gestire facilmente eventuali modifiche alla GUI (Graphical User Interface) che non richiederanno mai l'aggiornamento del presenter.

- Il presentatore (presenter) è colui che, oltre ad aggiornare la vista, interagisce con il model,che può essere identificato sia come lo stato di un oggetto che come dati persistenti in un'applicazione, in base alle richieste ricevute dalla view.
- Il modello (model) è un'interfaccia che definisce i dati da visualizzare ed incapsula lo stato dell'applicazione.

Capitolo 3

GWT

La necessità di creare un'applicazione Web nasce dai molteplici vantaggi che le applicazioni RIA (Rich Internet Application) possiedono nei confronti delle tecnologie alternative. Infatti rispetto alle applicazioni desktop, non richiedono installazione, gli aggiornamenti sono automatici, sono indipendenti dalla piattaforma utilizzata, più sicure in quanto girano nel ristretto ambiente del Web browser e maggiormente scalabili perchè la maggior parte del lavoro computazionale viene eseguito dal server.

Con l'avvento della tecnologia Ajax, lo sviluppo di applicazioni Web si basa su uno scambio di dati in background fra Web browser e server, che consente l'aggiornamento dinamico di una pagina Web senza eplicito ricaricamento da parte dell'utente. Purtroppo, scrivere applicazioni Ajax è molto complicato e perciò particolarmente esposto ad errori e bug; questo perchè JavaScript è un linguaggio piuttosto differente da Java e richiede molta pratica per lo sviluppo; il tutto è peggiorato dal fatto che JavaScript tende ad avere differenze in base al browser Web utilizzato, concentrando gli sforzi ed il tempo degli sviluppatori più sulla parte grafica che sulla logica applicativa.

Google Web Toolkit (GWT) nasce proprio per risolvere questi problemi, fornendo un vero e proprio livello di astrazione che nasconde il codice JavaScript e provvede automaticamente ad uniformare le differenze tra i browser.

Rilasciato da Google nell'estate 2006 sotto licenza Apache, Google Web Toolkit è un set di tool open source che permette agli sviluppatori Web di creare e gestire complesse applicazioni fronted JavaScript scritte in Java. Il codice sorgente Java può essere compilato su qualsiasi piattaforma con i file Ant inclusi. I punti di forza di GWT sono la riusabilità del codice, la possibilità di realizzare pagine Web dinamiche mediante le chiamate asincrone di Ajax, la gestione delle modifiche, il bookmarking, l'internazionalizzazione e la portabilità fra i differenti browser.

Come mostrato in Figura 3.1, GWT è composto principalmente da 4 componenti: un compilatore Java-to-JavaScript, un Web browser hosted e 2

librerie Java:

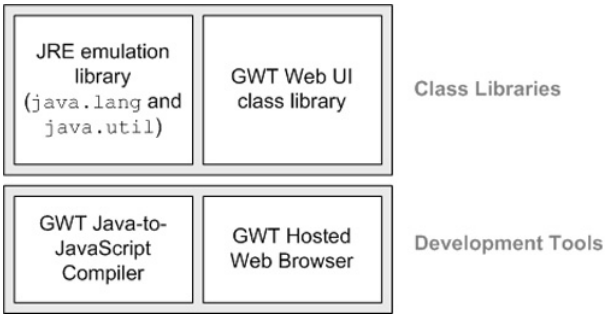

Figura 3.1: Architettura di GWT

Vediamo brevemente le singole componenti:

- Compilatore Java-to-JavaScript converte codice Java in JavaScript; deve essere utilizzato quando si vuole eseguire l'applicazione in modlità "Web mode".
- GWT Hosted Web Browser permette di eseguire le applicazioni in modalità "hosted mode" ossia di eseguire il codice Java nella JVM senza convertire il codice in linguaggi JavaScript/HTML.
- JRE emulation library contiene le implementazioni in linguagio Java-to-JavaScript delle librerie Java standard maggiormente utilizzate.
- GWT Wb UI class library è una libreria User Interface contenente un insieme di interfacce e classi che permettono di disegnare le pagine Web (ad es. bottoni, immagini, text boxess ecc.); questa è la libreria principale per crere applicazioni Web-based basate su GWT.

3.1 Il compilatore GWT

La necessità di scrivere codice in Java, invece di JavaScript, è radicata nella sempre crescente dimensione e complessità delle applicazioni RIA (Rich Internet Area). Infatti le applicazione di grandi dimensioni sono difficili da gestire e Java è stato programmato per renderle gestibili. GWT, oltre a riunire i benefici di Java, consente comunque di interagire con codice JavaScript esterno e con servizi server-side già esistenti.

Il cuore di GWT è un compilatore Java-to-JavaScript che produce codice in grado di essere eseguito su Internet Explorer, Mozilla Firefox, Safari, Opera e Chrome. Converte la sintassi Java in JavaScript, utilizzando le versioni JavaScript delle librerie Java più comunemente usate, come Vector, HashMap, e Date.

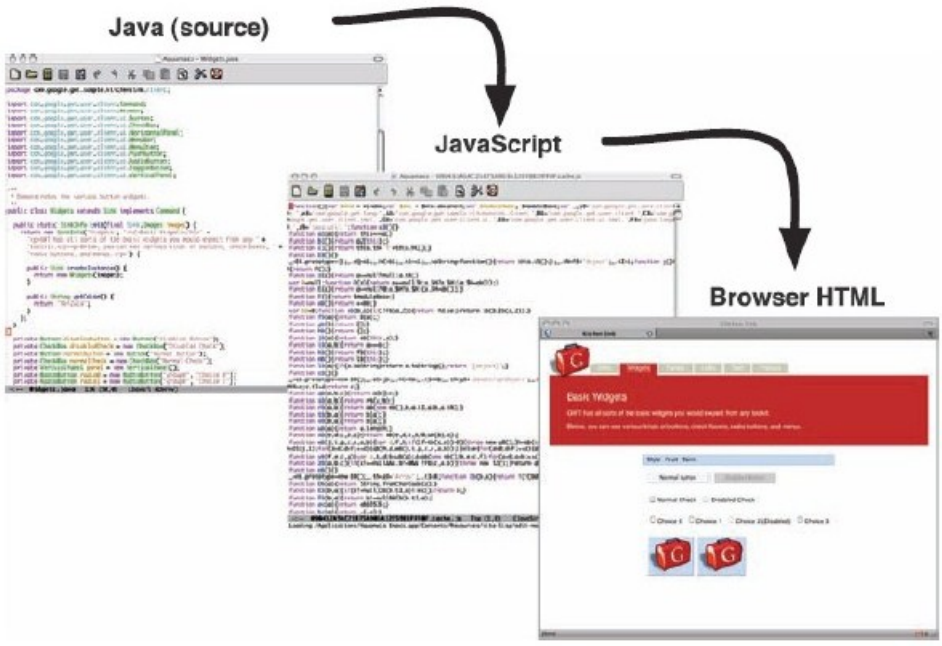

Figura 3.2: Funzionamento del compilatore GWT

Il compilatore è anch'esso un programma Java e, avviando la classe com.google.dwt.dev.GWTCompiler, traduce solo il codice che è effettivamente utilizzato all'interno del modulo, evitando così di trasferire codice JavaScript mai utilizzato al client. Questo procedimento richiede più tempo della normale compilazione Java, in quanto GWT crea un codice JavaScript altamente ottimizzato.

Il risultato della compilazione è una serie di file che, per un esempio "Hello World", risulta essere quelo di Figura 3.3.

La prima cosa che si nota è che non viene generato un solo file JS, ma uno per ogni piattaforma supportata. La classe <nomeprogetto>.nocache.js si prende cura del caricamento del corretto file per ogni specifica piattaforma. Il client e tutti gli eventuali server proxy intermedi possono tenere la cache dell'intera applicazione ottimamente perché il compilatore genera un file per piattaforma. Dal momento che il codice dell'applicazione non cambia tra una release e l'atra, l'applicazione può essere memorizzata nella cache per quanto tempo si desidera, riducendo sensibilmente il carico e la banda utilizzata per il server. Al fine di avviare e caricare l'applicazione, è sufficiente aprire il file index.html in un browser. Questo file contiene un riferimento al file <nome-progetto>.nocache.js.

Oltre al compilatore, GWT comprende anche una vasta libreria di widget

e pannelli, rendendo lo sforzo di costruire un' applicazione Web simile a quello per la realizzazione di un'applicazione desktop. La libreria di widget comprende i soliti oggetti come caselle di testo, menu a discesa, più widget complessi come barre di menu, finestre di dialogo, pannelli a schede e molti altri.

Figura 3.3: Esempio di compilazione

Quando si tratta di comunicazione con il server, GWT possiede uno strumento per ogni esigenza. Il primo è l'inserimento di alcuni involucri di varia complessità e capacità attorno all'oggetto JavaScript XMLHttpRequest, un oggetto spesso associato allo sviluppo Ajax. Un altro strumento fornito da GWT è un insieme di classi per sostenere il formato JavaScriptObject Notation (JSON) del messaggio. JSON è un formato di messaggio popolare, noto per la sua semplicità e la sua diffusa disponibilità. Come se non bastasse, GWT fornisce un ulteriore strumento che permette di inviare oggetti Java tra il browser e il server, senza la necessità di tradurli in un formato intermedio.

Questi strumenti di comunicazione consentono l'accesso ai servizi server-side, scritti in qualsiasi linguaggio, e rende possibile l'integrazione con framework come JSF, Spring, Struts, e EJB. Questa flessibilità offerta da GWT permette di continuare ad utilizzare gli stessi strumenti server-side in uso oggi.

3.2 GWT Shell e Hosted Mode Browser

Un'ulteriore ed importantissimo strumento per lo sviluppo offerto da GWT è la shell con l'hosted mode browser, che permette di testare la propria ap-

plicazione durante l'esecuzione del byte code Java nativo, fornendo la possibilità di utilizzare gli strumenti di ispezione preferiti. Oltretutto, l'hosted Web browser, fornito di una versione alleggerita del server Apache Tomcat, rende possibile testare il proprio JavaScript compilato con Junit.

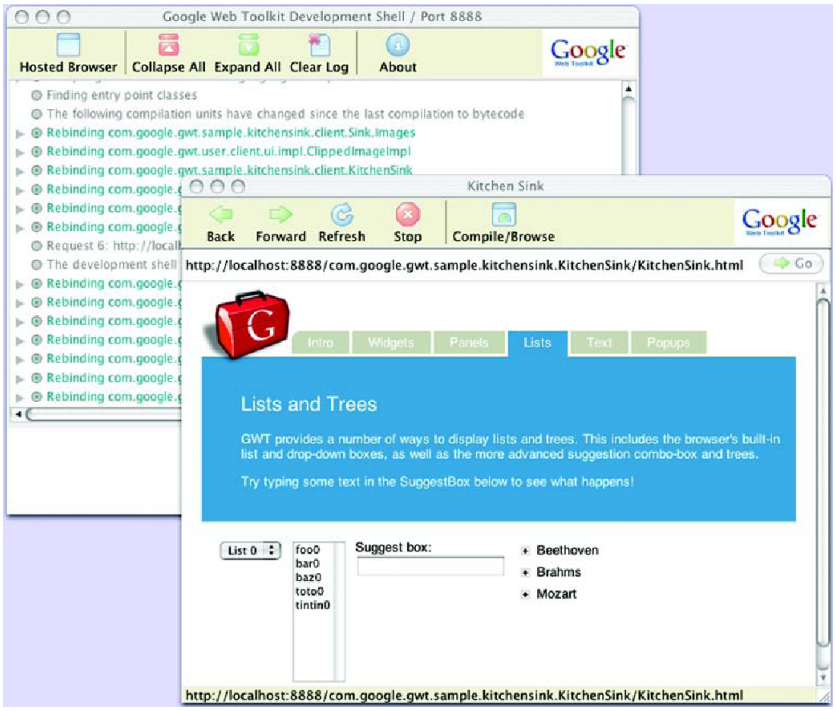

Figura 3.4: Hosted Mode Bowser

La console della shell fornisce un'ottima interfaccia per il log e per le GUI.

Il browser integrato è in grado di invocare le classi Java direttamente dagli eventi generati, ignorando così nella fase di debug la compilazione in JavaScript; in tal modo è possibile utilizzare un debugger Java standard per il codice Ajax, invece di affidarsi esclusivamente ai JavaScript compilati per i test e le interazioni.

Dalla versione 2.0, il browser Hosted Mode viene rimpiazzato dal Development Mode che, rispetto alle versioni precedenti, consente di poter effettuare il debugging dell'applicazioneWeb direttamente in un browser vero e proprio, invece di utilizzare il browser dell'SDK.

Altra particolarità del development mode è quella di poter scrivere il codice in un sistema operativo e testarlo su un altro.

3.3 Le API di GWT

In aggiunta ai vari componenti descritti fino a questo momento vengono fornite anche delle API (Application Programming Interface) per semplificare l'uso delle RIA.

Le API di GWT possono essere raggruppate in diverse categorie:

- Widgets e Panels consentono di disegnare le nostre interfacce utente (UI). La UI è composta da elementi (widget) e da contenitori (panel) che li contengono. Successivamente saranno descritti nel dettaglio i principali widget e panels presenti in una applicaione Web.
- I18I fornisce la localizzazione delle risorse di testo e l'internazionalizzazione alla nostra applicazione.
- Request Builder, JSON e XML Parser si occupano di facilitare la comunicazione con i server : costruiscono le richieste da inviare al server e processani i dati di risposta.
- RPC protocollo che consente la comunicazione con un server Java. Vedremo nel prossimo paragrafo come avviene la comunicazione con il server.
- History Management offrono uno strumento, compatibile con il browser, per definire le regole di historing e di navigazione all'interno di una pagina HTML la quale rappresenta l' applicazione Web.

Widgets e Panels

Il package com.google.gwt.user.client.ui contiene le classi che consentono di creare interfacce utenti dinamiche, di sviluppare le caratteristiche dell'interfaccia utente di un browser Web e forniscono componenti dinamici riutilizzabili che hanno lo stesso comportamento tra le diverse implementazioni e versioni supportate dal browser.

Queste componenti sono chiamate widgets e vanno dai più semplici bottoni e tabelle alle più complesse viste ad albero. Alcuni widget sono collegati direttamente agli elementi HTML che vengono usati nelle tradizionali pagine Web, altri sono composti da molti elementi HTML combinati con la gestione di particolari eventi come la pressione di un tasto della tastiera o un click del mouse.

Esistono differenti tipi di widget e possono essere raggruppati in base alla loro funzionalità :

- widgets statici ;
- form widgets ;

- wigdets complessi ;
- panels.

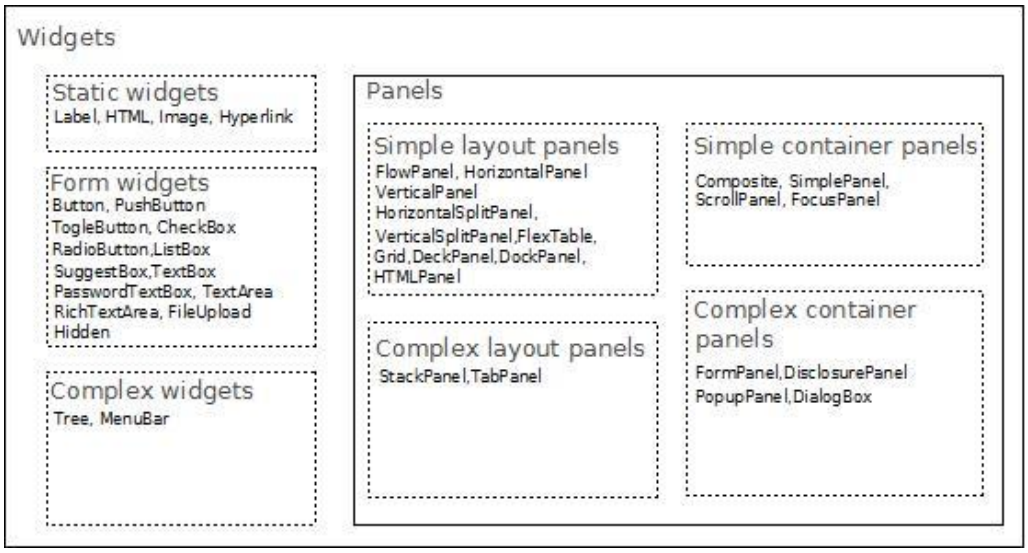

Figura 3.5: Widget divisi per funzionalità

Widgets statici

Questi semplici tipi di widget, come dice la stessa parola, non possono cambiare il loro stato ma possono comunque essere parte di un'interfaccia utente dinamica. Il codice che li descrive può modificare le loro proprietà (come per esempio il nome) ma non cambiare il risultato delle azione degli utenti. Alcuni esempi di widget statici possono essere Label, HTML, Immagine e Collegamento ipertestuale .
Di seguito è riportata un'immagine raffigurante i widget collegamento ipertestuale (contrassegnato dalla sottolineatura), immagine e HTML (permette di effettuare il collegamento).

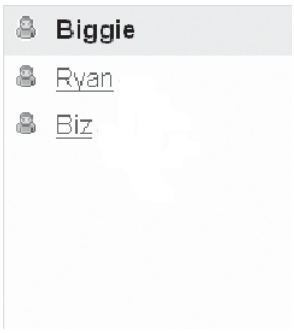

Figura 3.6: Esempio di widget statico

Form Widgets

I Form Widget rilasciati da GWT non devono essere utilizzati all'interno di moduli HTML, come avveniva nel modello Web tradizionale, e possono essere utilizzati in modo più flessibile come avviene nelle applicazioni desktop.
Tra i principali Form Widget troviamo: Button, CheckBox, RadioButton, ListBox, TextBox, PasswordTextBox, TextArea, FileUpload, e Hidden. Dalla versione GWT 1.4 sono stati introdotti ToggleButton, PushButton, SuggestBox, e RichTextArea.

Figura 3.7: Button

Figura 3.8: ListBox

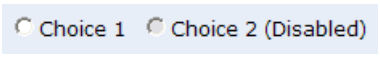

Figura 3.9: RadioButton

Figura 3.10: PasswordTextBox

Widgets complessi

Precedentemente abbiamo visto i widget dell'interfaccia utente nelle pagine Web che non hanno i tag HTML. I widget complessi sono stati creati dall'unione di tag HTML e di istruzioni per la gestione degli eventi, attraverso JavaScript, allo scopo di emulare i widget più sofisticati. Tra i widget complessi ci sono i Tree e MenuBar.

Figura 3.11: Tree

Figura 3.12: MenuBar

Panels

I panels (tradotti letteralmente in italiano come pannelli) sono pensati per disporre i widget in un' area della pagina HTML, con gli usuali layout verticali, orizzontali e flow (la disposizione HTML predefinita) .

Alcuni pannelli possono anche essere interattivi e sono in grado sia di fungere da contenitori di widget sia di rispondere a degli eventi di UI .

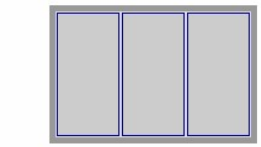

Figura 3.13: Un HorizontalPanel allinea i suoi contenuti in modo orizzontale

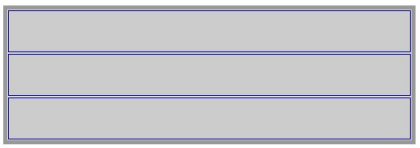

Figura 3.14: Un VerticalPanel allinea i suoi contenuti in modo verticale

Oltre a VerticalPanel e HorizontalPanel, che sono i due tipi più utilizzati nelle applicazioni Web, esistono numerose tipologie di pannelli[1].

3.4 Comunicare con il Server

Come già accennato in precedenza, una fondamentale differenza tra le applicazioni Ajax e le più tradizionali applicazioni Web HTML è che con le applicazioni Ajax non è necessario scaricare nuove ed intere pagine HTML quando vengono eseguite; questo perchè le pagine Ajax funzionano come vere e proprie applicazioni all'interno del browser, evitando così di richiedere nuovo HTML dal server per aggiornare l'interfaccia utente. Tuttavia, come tutte le applicazioni client-server, le applicazioni Ajax devono in ogni caso recuperare alcuni dati dal server per poter eseguire delle operazioni: per interagire con il server attraverso la rete, si effettua una chiamata di procedura remota (Remote Procedure Call). Il meccanismo di RPC , fornito da GWT, è basato su servlet Java e fornisce l'accesso alle risorse lato server. Questo meccanismo prevede la generazione di efficiente codice lato client e lato server per serializzare gli oggetti attraverso la rete utilizzado il deferred binding. Se usato correttamente, RPC da la possibilità di spostare tutta la logica UI sul client, conseguendo un notevole miglioramento delle prestazioni (si riduce la largezza di banda necessaria e il carico sul Web server). In alternativa, è possibile usufruire delle classi client HTTP di GWT per costruire ed inviare richieste HTTP personalizzate.

GWT fornsce un paio di modi diversi per la comunicazione con il server. E' possibile utilizzare il framework GWT RPC per rendere trasparenti le chiamate alle servlet Java e lasciare a GWT il compito di prendersi cura dei dettagli di basso livello come la serializzazione degli oggetti. GWT RPC facilita il passaggio di oggetti Java tra client e server (e anche viceversa) attraverso il protocollo HTTP.

[1] `http://gwt.google.com/samples/Showcase`

Utilizzando GWT RPC tutte le chiamate effettuate dalla pagina HTML al server sono asincrone. Questo significa che le chiamate non bloccano il client mentre attende una risposta dal server, ma viene eseguito il codice immediatamente successivo.
I vantaggi di effettuare chiamate asincrone rispetto alle più semplici (per gli sviluppatori) chiamate sincrone, si riscontrano in una migliore esperienza per gli utenti finali: innanzitutto, l'interfaccia utente è più reattiva; infatti, a causa del fatto che nei browser Web il motore JavaScript è generalmente di tipo single-thread, una chiamata sincrona al server genera un "blocco" fino alla conclusione della stessa, rovinando così l'esperienza dell'utente finale. Altro vantaggio delle chiamate asincrone è che risulta possibile eseguire altri lavori in attesa della risposta da parte del server; per esempio, è possibile costruire l'interfaccia utente e contemporaneamente recuperare i dati dal server per popolarla, riducendo così il tempo complessivo necessario all'utente per visualizzare i dati sulla pagina. Ultimo vantaggio , ma non meno importante, è che è possibile effettuare chiamate multiple al server nello stesso tempo; tuttavia questo parallelismo risulta fortemente limitato dal piccolo numero di connessioni che in genere i browser concedono alle singole applicazioni.

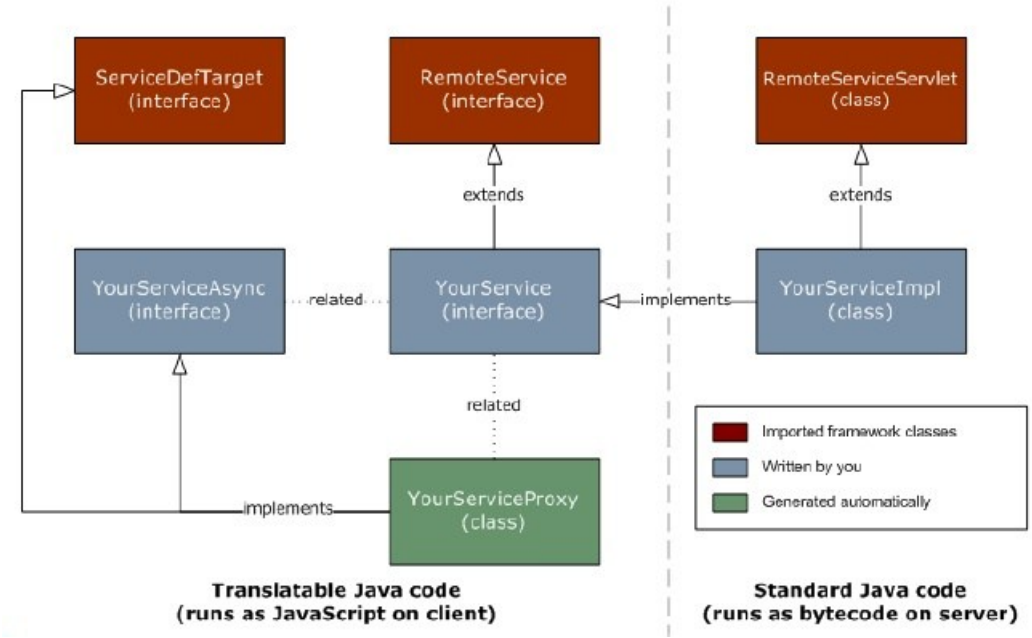

Figura 3.15: Implementazione di una RPC

Il codice server-side che viene invocato dal client viene spesso definito come un servizio, così l'azione di eseguire una chiamata di procedura remota a volte è indicata come invocare un servizio. Bisogna fare particolare attenzione all'uso del termine servizio che , in questo contesto, non ha lo stesso significato del più generale servizio Web.

Per creare un servizio di RPC è necessario:

1. Definire un'interfaccia per il tuo servizio che estenda RemoteService ed elenchi tutti i metodi RPC.
2. Definire una classe per implementare il codice server-side che estenda la classe RemoteServiceServlet ed implementi l'interfaccia creata al punto precedente.
3. Definire un'interfaccia asincrona che sarà chiamata dal codice client-side.

Per effettuare una chiamata RPC i passi da svolgere sono:

1. Creare un'istanza utilizzando l'interfaccia di servizio GWT.create().
2. Creare un oggetto Callback asincrono per essere informato quando la RPC è terminata.
3. Effettuare la chiamata.

3.5 Anatomia di una applicazione GWT

Per sviluppare un' applicazione Web con GWT è importante tenere conto della divisione in cartelle che Google ha pensato per il suo prodotto. Infatti non si possono suddividere i propri file in una gerarchia MVC tradizionale, ma bisogna tenere conto di alcuni aspetti fondamentali.

Innanzitutto troviamo il descrittore del modulo , in formato XML, che GWT utilizza per individuare la configurazione dell'applicazione; al suo interno occorre prestare molta attenzione a due tipi di voci: i moduli ereditati (inherited modules), paragonabili alle importazioni nelle normali classi Java, e il nome della classe principale a cui l'applicazione deve accedere all'inizio dell'esecuzione (Entry Point Class). Un Entry Point Class è una classe Java che deve implementare l'interfaccia EntryPoint, e di conseguenza deve fornire una implementazione del metodo onModuleLoad, invocato per primo quando si esegue l'applicazione. Solitamente in questo metodo vengono creati nuovi componenti grafici, vengono impostati gli handler per gli eventi (a chi ha avuto modo di utilizzare Swing tutto questo suonerà familiare) e viene modificato il DOM del browser.

Nell'esempio Figura 3.16 , il file si trova nella cartella principale "src/helloworld", con il nome "<nome-progetto>.gwt.xml".

Sempre nella cartella principale si trovano due sottocartelle chiamate "client" e "server". All' interno di "client" occorre inserire il codice Java client-side, come il pattern "model" ed il pattern "view" relativo al modello MVC, e

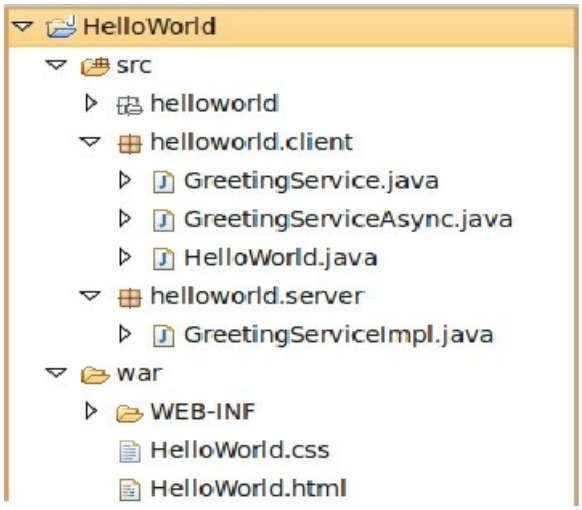

Figura 3.16: Struttura progetto GWT

verrà tradotto da GWT in JavaScript durante la compilazione. Purtroppo le classi importabili ed utilizzabili all'interno di questa cartella sono in numero piuttosto ristretto, limitando lo sviluppo ed aumentando gli sforzi dei programmatori. Fortunatamente, di release in release, questo numero aumenta, fornendo nuovi strumenti di lavoro.

Nella cartella "server", invece, troviamo il codice Java server-side, ossia il backend della nostra applicazione. Al suo interno si possono inserire i servizi (le servlet per le chiamate RPC) e i DAO (classi che rappresentano un'entità tabellare di un database ed effettuano su di esso operazioni CRUD).

Da notare nella figura 3.16 come per ogni servizio (o servlet) occorre creare un'interfaccia sincrona e una asincrona da inserire nella cartella client.

Come in tutte le applicazioni Web, si trova anche la cartella "war", cosiddetta pubblica, in quanto viene riportato tutto il codice (HTML e CSS) e gli oggetti (immagini e file) pubblici, oltre ad un file "Web.xml" contenente, tra le altre cose, l'elenco e la configurazione delle servlet del progetto .

3.6 Google App Engine

Google AppEngine può essere visto come il modo di creare applicazioni Web per essere ospitate sui Web server di Google. GAE è infatti un framework di tecnologie con le quali creare applicazioni senza preoccuparsi dell' infrastruttura sulle quali vengono ospitate.

L' infrastruttura è costituita da centinaia di server sparsi per il mondo che si distribuiscono le copie dell'applicazione in modo che qualunque server possa essere in grado di rispondere ad una richiesta, o perché è poco carico

in quel momento o perché si trova geograficamente vicino all'utente.

I cardini di Google AppEngine sono BigTable e GoogleFile System (GFP). Il primo è un database scalabile, lo stesso che supporta applicazioni come Google Maps o Google Search. È distribuito su molti server ed è in grado di servire gigabytes di dati ogni secondo. BigTable non è però un database relazionale, tuttavia AppEngine fornisce un interfaccia per gli sviluppatori (chiamata DataStore) con un approccio molto simile a quello di una struttura a tabelle. Il secondo è un file system distribuito altamente scalabile per applicazioni che manipolano grandi quantità di dati distribuiti.

Lo sviluppatore può controllare i parametri dell'applicazione attraverso il pannello di controllo di GAE: numero di richieste, banda in entrata e in uscita, tempo di CPU, richieste e traffico HTTPS, numero di query e spazio usato dal database, CPU usata dal database, numero di email e allegati inviati, numero e banda usata nel recuperare indirizzi (URL Fetch), numero di accessi e traffico prodotto verso Memcache, conteggio e traffico generato per manipolare immagini e il numero di deploy per l'applicazione.

Google App Engine supporta le applicazioni scritte in diversi linguaggi di programmazione come Java o Phyton. Con App Engine ambiente di runtime Java, è possibile creare la vostra applicazione utilizzando le tecnologie standard di Java, tra cui la JavaVirtualMachine (JVM), Java servlet, e il linguaggio di programmazione Java o qualsiasi altro linguaggio con un interprete o compilatore basato su JVM, come JavaScript o Ruby .

Tutte le applicazioni possono utilizzare fino a 1 GB di memoria con CPU e larghezza di banda sufficienti per supportare un'applicazione efficiente, il tutto assolutamente gratis. Quando si attiva la fatturazione per l'applicazione, vengono impostati alcuni livelli di utilizzo delle risorse; al di sopra di tali livelli si deve pagare in base alla quantità di risorse utilizzate.

Figura 3.17: Logo di Google App Engine

Capitolo 4

Progetto Rivenditori-Tipologie

In questo capitolo verranno descritte tutte le operazioni necessarie per la corretta installazione e configurazione del software utilizzato e verrà presentato il progetto di stage analizzandone le funzionalità tramite i file sorgenti creati e opportunamente commentati.

4.1 Configurazione ambiente di svilupo

Per installare il software necessario alla creazione dell'applicazione Web si è utilizzato il sistema operativo (OS) XP di Windows perchè già precedentemente installato nel terminale adottato dall' azienda. I programmi che dovranno essere installati sono:

- Java SDK;
- SpringSource Tool Suite (STS);
- Google Web Toolkit.

4.1.1 Java SDK

Java SDK (Software Development Kit) è il pacchetto Java utilizzato per la compilazione dei file sorgente utilizzati, poiché non è sufficiente la JRE (Java Runtime Environment, motore per l'esecuzione di programmi Java) per poter compiere questa funzione; per installarlo è sufficiente scaricarlo[1] (la versione scaricata è nel formato .exe cioè un file eseguibile) e con un doppio click partirà in automatico l'installazione.

Un passaggio molto importante dell'installazione riguarda il path o percorso nel quale Java SDK viene installato. Per eseguire il comando javac, necessario per la compilazione dei file sorgente, è necessario impostare le variabili d'ambiente:

1. Start -Pannello di Controllo - Sistema e Manutenzione - Sistema.

[1] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

2. Andare su Proprieta di sistema - Avanzate.

3. Fare click sul pulsante variabili d'ambiente. Apparirà una scheda contenente due aree: Variabili di sistema e Variabili dell'utente.

4. Nella sezione Variabili di sistema se ne trova una di nome PATH di cui si deve modificare il valore; digitare il path di installazione di Java seguito da \bin; .

5. Aggiungere una Variabile di sistema (funzionalità visibile nel riquadro inferiore) di nome JAVA_HOME ed assegnarle il valore della directory in cui è stato installato l'ambiente Java.

L'installazione di Java è completata.

Per verificare se tutto e andato bene aprire il terminale o prompt dei comandi e digitare javac. Se il terminale rispondera con un messaggio Usage javac... l'installazione e avvenuta correttamente ed il Sistema Operativo riconosce i comandi Java.

Se invece il terminale risponde: javac non e riconosciuto come comando... significa che le variabili di ambiente non sono state settate correttamente oppure durante l'installazione di Java sono insorti errori e ripercorrere i punti 1-5.

4.1.2 SpringSource Tool Suite

Un Integrated Development Enviroment (IDE), in italiano ambiente di sviluppo integrato è un software che aiuta i programmatori nello sviluppo del codice. Normalmente consiste in un Editor di codice sorgente, un compilatore e/o un interprete e (solitamente) un debugger.

SpringSource Tool Suite è l'IDE di riferimento per chi sviluppa con la famiglia di tecnologie Spring: è basato su Eclipse (IDE molto utilizzato nell'ambito lavorativo che supporta numerosi linguaggi di programmazione) e fornisce il miglior ambiente di sviluppo per la creazione di applicazioni aziendali.

Quando si effettua il download dell'IDE[2] si può scaricare direttamente il file eseguibile (.exe) o in alternativa il file compresso (.zip). Se si è scelto il file eseguibile, una volta terminato il download con un doppio click si avvierà l'installer che consentirà il processo di installazione. Se si è scaricato il file compresso usare 7-Zip e non il programma predefinito per l'estrazione dei file in quanto potrebbe causare degli errori.

[2] `http://www.springsource.org/downloads/sts`

Una volta che STS installer è stato lanciato i passi necessari per l'installazione sono :

1. Click "Next" sulla pagina principale.
2. Leggere ed accettare la licenza e cliccare "Next".
3. Selezionare il percorso di installazione e cliccare "Next".
4. Selezionare le componenti che si desidera installare e cliccare "Next".
5. Selezionare il path dove si è installata la JDK .
6. Creare un percorso breve (solo in Windows).
7. Click "Finish" per terminare l'installazione. Spuntare l'opzione "Launch SpringSource Tool Suite" per avviare il programma una volta usciti dall'installer.

4.1.3 Google Web Toolkit

Come già descritto nel Capitolo 3, GWT è un framework rivolto agli sviluppatori Java e adatto allo sviluppo di applicazioni Web. Per installare il plugin è necessario aver già installato STS. Una volta aperta la pagina iniziale di STS cliccare su "Extensions", spuntare "Google Plugin for Eclipse" e confermare l'installazione premendo il pulsante "Install" in basso a destra.

Se l'installazione è avvenuta correttamente il computer vi chiederà di riavviare STS. Quando si riaprirà il programma si noterà, nella barra dei pulsanti in alto , la presenza di un nuovo pulsante : GWT è installato e pronto all'uso.

Figura 4.1: Pulsante di GWT

4.2 Descrizione applicazione Web

4.2.1 Progettazione del modello

Questa applicazione segue il modello Application Manager. E' possibile utilizzare tale modello per gestire una grande quantità di dati. Esso fornisce tipicamente un interfaccia che rende semplice individuare i dati, caricarli dinamicamente dal server per poi procedere con le classiche operazioni di creazione, lettura, aggiornamento ed eliminazione (CRUD).

Il modello dell'applicazione è ripartito tra una applicazione Web ed un editor database poiché entrambi devono operare sugli stessi dati in un database condiviso. Come già detto nel paragrafo 1.2.3 , le entità del modello sono Rivenditore e Tipologia legate tra loro attraverso una relazione uno a molti: affinchè avvenga questa relazione è necessario che l'entità Rivenditore abbia un attributo che faccia riferimento alla chiave primaria dell'entità Tipologia. Tale attributo sarà da ora in poi chiamato posted-by.

Nel codice Java, la classe Rivenditore terrà un riferimento a un'istanza di Tipologia per l'attributo posted-by, e sarà in possesso di un elenco di tipologie.

La classe Rivenditore avrà la seguente implementazione :

```
@PersistenceCapable(identityType=IdentityType.APPLICATION)
public class Rivenditore extends BaseObject {
private static final long serialVersionUID=1L;
@PrimaryKey
@Persistent(valueStrategy=IdGeneratorStrategy.IDENTITY)
protected Long id;
@Persistent
private String codice;
@Persistent
private String descrizione;
@Persistent
private String tipologia_id;
  public String getDescrizione() {
  return descrizione;
  }
  public void setDescrizione(String description){
  this.descrizione = description;
  }
  public String getCodice() {
  return codice;
  }
  public void setCodice(String codec) {
  this.codice = codec;
  }
  public String getTipologia_id() {
  return tipologia_id;
  }
  public void setTipologia_id(String user_id) {
  this.tipologia_id = user_id;
  }
}
```

Google AppEngine include per il suo datastore il supporto di due diverse API standard: Java Data Object (JDO) e Java Persistence API (JPA). Queste interfacce sono fornite da DataNucleus Access Platform, un' implementazione open source di diversi standard Java di persistenza, con un adattatore per il datastore AppEngine. Per poter utilizzare JDO in AppEngine, occorre modificare gli oggetti del progetto, aggiungendo le annotazioni alle classi e relativi attributi:

- @PersistenceCapable prima della dichiarazione della classe;
- @PrimaryKey e @Persistent(valueStrategy ...) prima della dichiarazione dell'attributo di chiave primaria;
- @Persistent prima della dichiarazione di un attributo.

Si può notare che la classe Rivenditore estende la Classe BaseObject, la quale consente all'applicazione di utilizzare tutti gli oggetti del sistema, in modo generico. Il posted-by è rappresentato dall'attributo tipologia_id che ha la funzione di chiave esterna. Questo significa che ogni istanza di Rivenditore farà riferimento ad una precisa istanza di Tipologia, proprio come era stato specificato nella Figura 1.4.

Di seguito viene illustrata l'implementazione della classe BaseObject:

```
public class BaseObject implements Serializable {
  protected String id;
  public String getId() {
  if( id == null )
  return null;
  return id.toString();
  }
  public void setId(String id) {
  this.id = id;
  }
}
```

Si noti che questa classe implementa l'interfaccia Serializable. La serializzazione è il processo per mezzo del quale lo stato di un oggetto viene scritto in un flusso (serializzato) in modo tale che successivamente può essere riletto. Gli oggetti di una classe sono "serializzabili" se e solo se la classe implementa l'interfaccia Serializable.

Guardiamo il codice della classe Tipologia:

```
@PersistenceCapable(identityType=IdentityType.APPLICATION)
public class Tipologia extends BaseObject{
private static final long serialVersionUID=1L;
@PrimaryKey
@Persistent(valueStrategy=IdGeneratorStrategy.IDENTITY)
protected Long id;
@Persistent
private String codice;
  @Persistent
  private String descrizione;
  public String getDescrizione() {
  return descrizione;
  }
  public void setDescrizione(String email){
    this.descrizione = email;
  }
  public String getCodice() {
    return codice;
  }
  public void setCodice(String name){
  this.codice = name;
  }
}
```

Come la classe Rivenditore, anche la classe Tipologia estende la classe BaseObject e fornisce i metodi getter e setter per i suoi tre attributi. Queste classi Java saranno utilizzate nella nostra applicazione sia lato client sia lato server per l'applicazione Java servlet con GWT-RPC.

4.2.2 Utilizzo DAO

E' buona norma separare gli oggetti entità dal codice di accesso ai dati delle entità. Un modo per separare il codice di accesso dagli oggetti entità è la creazione di Data Access Objects (DAO). La creazione di questi oggetti ci permette di riutilizzare le entità senza l'accoppiamento del codice di accesso ai dati.

In tutte applicazioni Web che utilizzano un pattern architetturale MVC il recupero dei dati è affidato alla parte view tramite un'interfaccia.
Nel nostro progetto l'interfaccia ObjectDAO, che definiremo in seguito, ha il compito di operare le richieste per la manipolazione dei dati e di seguito verranno elencati ed illustrati i metodi che consentono di eseguire le cosiddette "funzioni amministrative" sui dati del server:

- GetById(String id, ObjectListener handler): chiede il server di restiuire un oggetto in base al suo ID. Il metodo accetta un ID come primo parametro e un ObjectListener come secondo parametro. Dal momento che queste chiamate di metodo al server sono sempre asincrone, il valore di ritorno viene restituito attraverso l'interfaccia ObjectListener.
- GetAll (CollectionListener handler): chiede al server di restituire tutti gli oggetti del tipo CollectionListener.
- GetAllFrom (BaseObject object, String member, CollectionListener handler): chiede al server di restituire tutti gli oggetti dato un campo specificato di un oggetto. Il valore restituito viene restituito al CollectionListener.
- Save(BaseObject object): invia un oggetto al server. Se l'oggetto non è ancora stato salvato, allora un nuovo oggetto viene creato, altrimenti, un oggetto esistente viene aggiornato.
- Delete (BaseObject object): elimina l'oggetto specificato dal parametro.
- AddTo (BaseObject object, String string, BaseObject objectToAdd): aggiunge un oggetto .

Il DAO è il punto di integrazione tra il resto dell'applicazione ed il lato server. La view fa riferimento a questi DAO attraverso un'interfaccia ObjectFactory. L'applicazione crea un ObjectFactory per ogni metodo di interazione con il server. La sua interfaccia utilizzata nella view dell'applicazione simile a questa :

```
public interface ObjectFactory {

  public interface ObjectDAO {
  void getById( String id, ObjectListener handler );
  void getAll( CollectionListener handler );
```

```
  void getAllFrom( BaseObject object, String member,
  CollectionListener handler );
  void save( BaseObject object );
  void delete( BaseObject object );
  void addTo( BaseObject object, String string, BaseObject
  objectToAdd);
  } // fine interfaccia ObjectDAO

  ObjectDAO getTipologiaDAO();
  ObjectDAO getRivenditoreDAO();
  void setListener( ObjectFactoryListener listener );
}
```

L'interfaccia fornisce un metodo per accedere a ciascuna classe di oggetti sul server, nonché un metodo per impostare ObjectFactoryListener così che gli eventi possono essere ricevuti:

```
public interface ObjectFactoryListener {
void onRefresh();
void onError(String error);
void onLoadingStart();
void onLoadingFinish();
}
```

Con la definizione di queste interfacce, possiamo costruire la parte view dell'applicazione, implementarle ed implementare il server.

4.2.3 Costruzione dell'interfaccia del database

La maggior parte delle applicazioni che utilizzano il modello Application Manager, hanno l'interfaccia a due riquadri: il riquadro di sinistra, in genere, presenta una visione gerarchica dei dati del sistema, ed il riquadro a destra ha una vista dettagliata della voce selezionata. La parte sinistra è composta da elenchi e cartelle, ed è dinamica: l'utente con un click sulla cartella apre l'elenco di tutti i dati presenti, i quali vengono scaricati dal server, e successivamente può selezionare quale oggetto desidera. Quando l'utente seleziona una voce, una visione dettagliata della voce appare nello spazio alla sua destra.

Uso dei widget Tree e SplitPanel

La principale vista dell'applicazione è definita nella classe DatabaseEditorView. Questa classe estende Composite widget di GWT e costruisce i due riquadri descritti precedentemente. E' la vista principale per l'applicazione e viene aggiunta alla pagina host HTML attraverso la classe RootPanel :

```
public class RivTipWebApp implements EntryPoint{
  public void onModuleLoad(){
  //create view
  DatabaseEditorView view = new DatabaseEditorView();
  RootPanel.get("databaseEditorView").add( view );
  }
}
```

DatabaseEditorView costruisce l'interfaccia dell'applicazione utilizzando i Widgets HorizontalSplitPanel e tree sulla parte sinistra. Si può vedere come

questa classe esegue questo, cercando nella dichiarazione e nel costruttore di classe :

```
public class DatabaseEditorView extends Composite implements
TreeListener,ObjectFactoryListener {

//widget
private Label loadingLabel = new Label("loading...");
private Tree treeList = new Tree();
private HorizontalSplitPanel mainPanel=new
HorizontalSplitPanel();
private LoadingPanel loading = new
LoadingPanel(loadingLabel);

//tree items
private TipologieTreeItem tipologiaItems = new
TipologieTreeItem(this);
private RivenditoriTreeItem rivenditoreItems = new
RivenditoriTreeItem(this);

//object factory
private ObjectFactory objectFactory;

//Costruttore
public DatabaseEditorView(){
initWidget( mainPanel );
setStyleName("databaseEditorView");
RoundedPanel rounded = new RoundedPanel( "#f0f4f8" );
rounded.setWidget(treeList);
rounded.setWidth("100%");
mainPanel.setLeftWidget( rounded );
mainPanel.setSplitPosition("250px");
treeList.addItem( rivenditoreItems );
treeList.addItem( tipologiaItems );
treeList.addTreeListener(this);
RootPanel.get().add( loading);
}

public void onTreeItemSelected(TreeItem item) {
if( item instanceof BaseTreeItem ){
BaseTreeItem eventsItem = (BaseTreeItem)item;
eventsItem.onTreeItemSelected();
}}

public void onTreeItemStateChanged(TreeItem item) {
if( item instanceof BaseTreeItem ){
BaseTreeItem eventsItem = (BaseTreeItem)item;
eventsItem.onTreeItemStateChanged();
}}

public ObjectFactory getObjectFactory() {
return objectFactory;
}

public void setObjectFactory(ObjectFactory objectFactory)
{
this.objectFactory = objectFactory;
objectFactory.setListener(this);}

public void setMainPane(Widget view) {
mainPanel.setRightWidget( view );
}

public void setMainPaneLoading() {
mainPanel.setRightWidget( loadingLabel );
}

public Tree getTree() {
return treeList;
}
```

```
public void onRefresh() {
BaseTreeItem item = (BaseTreeItem)treeList.getSelectedItem();
if( item == null ){
mainPanel.setRightWidget( null );
}
else{
item.onRefresh();
}}

public void onError( String error ){
PopupPanel popup = new PopupPanel(true);
popup.setStyleName( "error" );
popup.setWidget( new HTML(error) );
popup.show();
popup.center();
}

public void onLoadingFinish() {
loading.loadingEnd();
}

public void onLoadingStart() {
loading.loadingBegin();
}
} // fine classe
```

Questa classe inizia estendendo il widget Composite e implementa due interfacce. La prima interfaccia, TreeListener, consente a questa classe di rispondere agli eventi dei sui widget Tree (ne parleremo in seguito di questi eventi). La seconda interfaccia, ObjectFactoryListener, fornisce un feedback per l'utente su cosa sta accadendo sul server. In particolare, la classe utilizza il widget LoadingPanel per permettere all'utente di sapere che una richiesta sincrona è in attesa sul server. I widget elencati come attributi sono HorizontalSplitPanel e Tree. Nel costruttore la vista è costruita inizializzando il widget principale per questa classe composito al HorizontalSplitPanel, quindi aggiungendo il widget tree sul lato sinistro del pannello condiviso.
Noterete che un RoundedPanel viene utilizzato come contenitore dei widget Tree. Il widget RoundedPanel non viene fornito da GWT ma aiuta visivamente a separare i nodi padri dai nodi figli e aggiunge un aspetto amichevole all'applicazione.
La classe crea inizialmente due elementi: uno è un'istanza della classe TipologieTreeItem e l'altro è un'istanza della classe RivenditoriTreeItem. Esse rappresentano i punti di partenza della struttura ad albero che l'utente vede ed è in grado di interagire all'avvio dell'applicazione. In generale quando si crea un oggetto di tipo Rivenditore o Tipologia, esso viene inserito sotto il relativo nodo di appartenenza ed il tutto appare come un widget Tree.

Ampliamento e Caricamento dinamico degli elementi Tree

L'estensione della classe TreeItem dà all'applicazione prestazioni leggermente migliori, rende il codice più organizzato e facile da leggere, consente di eseguire operazioni per ogni elemento della struttura ad albero e permette di fornire funzionalità aggiuntive. L'applicazione utilizza due classi astratte, BaseTreeItem e DynamicTreeItem, per fornire funzionalità condivise da

tutti gli elementi della struttura tree.

La classe BaseTreeItem contiene un riferimento alla vista principale dell'applicazione, in modo che ogni elemento del widget tree può ottenere informazioni, e tre metodi, tutti senza implementazione, chiamati dal DatabaseEditorView in risposta ad una particolare azione eseguita sugli elementi della struttura ad albero.

```
public class BaseTreeItem extends TreeItem{
  private final DatabaseEditorView view;
  //Costruttore
  public BaseTreeItem(String html,DatabaseEditorView view){
  super(html);
  this.view = view;}
  public DatabaseEditorView getView() {
  return view;
  }
  public boolean isSelected(){
  return getTree().getSelectedItem() == this;
  }
  public void onTreeItemSelected(){}
  public void onTreeItemStateChanged(){}
  public void onRefresh(){}
} // fine classe
```

I tre metodi non hanno un'implementazione perchè si lascia al programmatore la libertà di poter gestire le azioni dell'utente. Una possibile implementazione dei metodi onTreeItemStateChanged e onTreeItemSelected può essere:

```
private void onTreeItemStateChanged () {
  onTreeItemSelected();
  }

private void onTreeItemSelected() {
  getView().getObjectFactory().getTipologiaDAO().
  getAll(this);
}
```

Con questa implementazione quando l'utente seleziona la voce Tipologie della struttura ad albero, viene chiamato il metodo onTreeItemSelected della classe TipologieTreeItem che recupera tutti gli oggetti di tipo tipologia attraverso l'oggetto DAO.

La classe TipologieTreeItem implementa l'evento onTreeItemStateChanged e l'evento onTreeItemSelected. Questi eventi non sono uguali: l'evento onTreeItemStateChanged si verifica quando l'utente apre o chiude un elemento dell'albero, l' evento onTreeItemSelected si verifica quando l'utente seleziona una voce del widget tree utilizzando la tastiera o il mouse.
Si noti inoltre che l'implementazione di onTreeItemSelected passa un riferimento a se stesso tramite il metodo getAll sull'oggetto DAO di tipo Tipologia. Questo riferimento è l'interfaccia CollectionListener. Ciò significa che la classe TipologieTreeItem implementa l'interfaccia CollectionListener (e di conseguenza il metodo onCollection) che permette di ricevere la lista

delle tipologie quando viene recuperata dal server.

La Gestione degli eventi sugli elementi del widget tree , richiedono, in modo asincrono all'ObjectFactory di fornire all'utente una vista che l'applicazione carica in modo dinamico. Quando si avvia l'applicazione per prima volta, ogni nodo padre non presenta nodi figli, e quindi non è presente alcuna indicazione visiva che indichi la presenza di nodi figli; infatti in termini di indizi visivi, non avranno il simbolo "+" accanto a loro nome per indicare che può essere espanso. Per risolvere questo problema abbiamo bisogno di costruire la classe DynamicTreeItem. Quando il nodo è inizialmente chiuso, avrà il segno + per indicare che può essere espanso. Una volta espanso, l'evento onTreeItemSelected caricherà gli elementi figlio in modo asincrono. Mentre l'utente attende che siano caricati viene visualizzato il widget label con la scritta loading .... Al termine del caricamento il widget viene rimosso.

Di seguito vengono illustrate le implementazioni delle classi DynamicTreeItem e TipologieTreeItem. La classe RivenditoriTreeItem non verrà illustrata in quanto presenta lo stesso codice della classe TipologieTreeItem con la differenza che la parola Tipologia è sostituita da Rivenditore.

```
public class DynamicTreeItem extends BaseTreeItem {
final int STATE_EMPTY=0;
final int STATE_LOADING=1;
final int STATE_LOADED=2;
private int state;

public DynamicTreeItem(String html,DatabaseEditorView view)
{
super(html, view);
setEmpty();
}

public void addItem( TreeItem item ){
if( !isLoaded() ){
state = STATE_LOADED;
removeItems();
}
super.addItem(item);
}

public boolean isLoaded()
{ return state == STATE_LOADED;}
public boolean isLoading()
{ return state == STATE_LOADING;}
public boolean isEmpty()
{ return state == STATE_EMPTY;}
public void setToLoading()
{ state = STATE_LOADING;}
public void setEmpty(){
removeItems();
addItem("loading...");
state = STATE_EMPTY;
}
}// fine classe DynamicTreeItem

public class TipologieTreeItem extends DynamicTreeItem
implements CollectionListener {
//costruttore
public TipologieTreeItem(DatabaseEditorView view) {
super("<img␣src='item-tipologie.png'hspace='3'>
```

```
Tipologie",view);
}

public void onTreeItemSelected() { onFocused(); }

public void onTreeItemStateChanged() { onFocused(); }

public void onRefresh() {
setEmpty();
onFocused();}

private void onFocused() {
getView().getObjectFactory().getTipologiaDAO().getAll(this);
}

//metodo onCollection presente in CollectionListener

public void onCollection( List list ){
if( isEmpty() ){
removeItems();
for( Iterator it = list.iterator(); it.hasNext(); )
{
addItem( newTipologiaTreeItem( (Tipologia)it.next(),
getView() ) );
}
}
if( isSelected() ){
getView().setMainPane( newTipologiaListView( list,
getView() ) );
}

}// fine classe TipologieTeeItem
```

Creazione della vista dell'area di lavoro

La parte destra dell'interfaccia del database detta vista dell'area di lavoro rappresenta una vista più dettagliata dell'elemento selezionato nella struttura ad albero presente nella parte sinistra dell' interfaccia. La vista dell'area di lavoro viene visualizzata ogni volta che l'utente seleziona un nuovo elemento e sostituisce la vista precedentemente visualizzata.

Queste viste possono essere suddivise in due categorie: una list view per gli elementi del widget tree che rappresentano una raccolta di dati , e una object view per gli elementi che rappresentano un singolo oggetto. Ogni vista nella list view estende la classe ListView per condividerne le funzionalità e ogni vista della categoria object view estende la classe ObjectView. La ListView segue il layout mostrato nella Figura 4.2 .

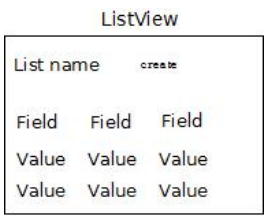

Figura 4.2: Layout di ListView

In questo layout è possibile vedere il comando create, accanto al nome della lista, che permette di aggiungere un nuovo oggetto alla lista.

La classe RivenditoreListView estende la visualizzazione elenco (ListView) per elencare tutti i rivenditori, mentre la classe TipologiaListView estende ListView allo scopo di elencare tutte le tipologie. Guardiamo alla realizzazione della classe RivenditoreListView per vedere come la classe ListView viene utilizzata nella applicazione:

```
public class RivenditoreListView extends ListView{

  private class GetAuthorListener implements ObjectListener{
  private final int row;
  public GetAuthorListener( int row ){
    this.row = row;
  }
  public void onObject(BaseObject object) {
  Tipologia user = (Tipologia)object;
  if( user != null )
  addField(row,user.getCodice());
  }
  }// fine classe GetAuthorListener

// costruttore
public RivenditoreListView( List rivenditori,
DatabaseEditorView view){
super("<img␣src='item-rivenditori.png'hspace='3'>
Rivenditori",view );
addColumn( "id" );
addColumn( "codice" );
addColumn( "descrizione" );
addColumn( "lista␣tipologie" );
int row = 0;
for( Iterator it = rivenditori.iterator(); it.hasNext();){
Rivenditore rivenditore = (Rivenditore)it.next();
addField(row,rivenditore.getId());
addField(row,rivenditore.getCodice());
addField(row,rivenditore.getDescrizione());
if( rivenditore.getTipologia_id().length()>0)
{
getView().getObjectFactory().getTipologiaDAO().getById(
rivenditore.getTipologia_id(), new GetAuthorListener(row) );
row++;}}

} // fine costruttore

  //nuovo RivenditoreDialog quando l'utente clicca create
  protected void onCreate()
  {
  new RivenditoreDialog( getView() );
  }

} // fine classe RivenditoreListView
```

In tale classe il layout è gestito dal costruttore. Sempre nel costruttore la classe RivenditoreListView utilizza i metodi addColumn e addField di ListView per costruire la tabella di valori: il metodo addColumn aggiunge una colonna e prende il valore di intestazione di tale colonna come parametro, mentre addField aggiunge un valore del campo nella riga specificata.

Per l'attributo lista tipologie che ogni oggetto Rivenditore ha con un oggetto tipologia, la classe richiede l'oggetto TipologiaDAO. Poiché la richiesta

è asincrona, essa ha bisogno di attendere una risposta mediante l'attuazione di un'istanza dell'interfaccia ObjectListener. La classe gestisce l'evento onObject che imposta il valore del campo in base al valore dell'attributo codice che è stato recuperato. Il responsabile per soddisfare questa richiesta è ObjectFactory. Potrebbe esserci una risposta immediata l'oggetto è memorizzato nella cache, oppure potrebbe eseguire una richiesta al server, nel qual caso l'applicazione visualizza la scritta loading per indicare che i dati vengono caricati.

Alla fine della definizione della classe è presente il metodo onCreate. La superclasse ListView chiama questo metodo quando il comando create viene cliccato. La classe RivenditoreListView implementa questo metodo e crea una finestra di dialogo chiamata RivenditoreDialog. Questa finestra, descritta più avanti, fornisce all'utente i campi necessari per costruire un nuovo oggetto Rivenditore.
Come RivenditoreListView, anche la classe TipologieListView estende ListView. Ecco qui la sua implementazione:

```
public class TipologiaListView extends ListView{

// costruttore
public TipologiaListView(List tipologie,DatabaseEditorView
view){
  super("<img␣src='item-tipologie.png'hspace='3'>Tipologie",
  view );
  addColumn( "id" );
  addColumn( "codice" );
  addColumn( "descrizione" );
  int row = 0;
  for( Iterator it = tipologie.iterator(); it.hasNext();){
    Tipologia user = (Tipologia)it.next();
    addField(row,user.getId());
    addField(row,user.getCodice());
    addField(row,user.getDescrizione());
    row++;
  }
}

//nuova Tipologia quando l'utente clicca create
protected void onCreate(){
new TipologiaDialog( getView() );
}
}// fine classe
```

L'implementazione della classe ListView utilizza un widget TitleCommandBar per il titolo e un widget FlexTable per la tabella degli oggetti. Il seguente codice implementa la classe ListView:

```
public class ListView extends Composite {

  private final DatabaseEditorView view;
  private final TitleCommandBar title;
  private final FlexTable rows = new FlexTable();

  //costruttore
  public ListView( String titleValue, DatabaseEditorView  view ){
  this( titleValue, view, "create" );
  }
```

```
  //costruttore
  public ListView(String titleValue,DatabaseEditorView view,
  String createLabel){
  this.view = view;
  VerticalPanel mainPanel = new VerticalPanel();
  initWidget( mainPanel );
  mainPanel.setWidth("100%");
  rows.setWidth("100%");
  title = new TitleCommandBar(titleValue);
  mainPanel.add( title );
  mainPanel.add(rows);
  rows.getRowFormatter().setStyleName(0,"gwtapps-ListHeaderRow");
  title.addCommand(createLabel, new ClickListener(){
  public void onClick( Widget sender ) {
  onCreate();
  }});
  } //fine costruttore

  protected void addColumn( String name ){
  int column = rows.getRowCount()>0?rows.getCellCount(0):0;
  rows.setWidget(0, column, new Label(name));
  }

  protected void addField( int row, String value ){
  row = row + 1;
  int column = 0;
  if( rows.getRowCount()==row ){
  if( row%2 == 0)
  rows.getRowFormatter().setStyleName(row,"gwtapps-Even");}
  else{
  column = rows.getCellCount(row);
  }
  Label fieldValue = new Label(value);
  fieldValue.setStyleName("gwtapps-FieldValue");
  rows.setWidget(row, column, fieldValue);
  }

  protected void onCreate(){
  }
} // fine classe
```

Il layout della classe ObjectView è leggermente diverso da quello di ListView e ha il compito di visualizzare un oggetto e supportare i comandi edit e un delete. Gli attributi (o campi) dell'oggetto vengono visualizzati in orizzontale anziché in verticale perchè l'utilizzo dello spazio orizzontale è più efficiente e facilita la lettura dei campi. L'ObjectView segue il layout mostrato nella Figura 4.3 .

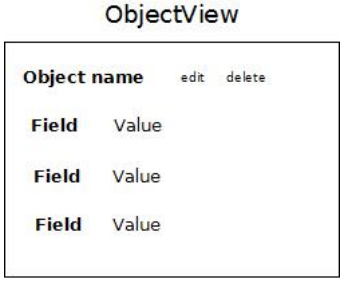

Figura 4.3: Layout di ObjectView

L'implementazione della classe ObjectView:

```
public class ObjectView extends Composite{

  private FlexTable fields = new FlexTable();
  private TitleCommandBar title;
  private final DatabaseEditorView view;

  //costruttore
  public ObjectView(final String titleValue,DatabaseEditorView
  view ){
  this.view = view;
  VerticalPanel mainPanel = new VerticalPanel();
  initWidget( mainPanel );
  mainPanel.setWidth("100%");
  fields.setWidth("100%");
  title = new TitleCommandBar(titleValue);
  mainPanel.add( title );
  mainPanel.add(fields);
  getTitleCommandBar().addCommand("edit", new ClickListener(){
  public void onClick( Widget sender ) {
  onEdit();
  }});

  getTitleCommandBar().addCommand("delete",new ClickListener()
  {
  public void onClick( Widget sender ) {
  if(Window.confirm("Sei␣sicuro␣di␣voler␣eliminare␣l'oggetto?"))
  {
  onDelete();
  getView().getTree().getSelectedItem().remove();
  }}});

  }// fine costruttore

  protected void onDelete() {}

  protected void onEdit() {}

  protected void addField(String name, String value ) {
  int row = fields.getRowCount();
  Label fieldName = new Label( name );
  Label fieldValue = new Label( value );
  fieldName.setStyleName("gwtapps-FieldName");
  fieldValue.setStyleName("gwtapps-FieldValue");
  fields.getCellFormatter().setWidth(row,1,"100%");
  fields.setWidget(row, 0, fieldName);
  fields.setWidget(row, 1, fieldValue);
  if( row%2 == 0)
  fieldValue.addStyleName("Even");
  }

  public TitleCommandBar getTitleCommandBar() {
  return title;
  }

  public DatabaseEditorView getView() {
  return view;
  }
}// fine classe
```

Da notare che al comando di rimozione è stato associato il metodo window.confirm di GWT che visualizza una finestra di dialogo per l'utente allo scopo di aiutare a prevenire la cancellazione accidentale dei dati.
La classe ObjectView è molto importante in quanto permette ad un oggetto di tipo rivenditore o tipologia di essere visualizzato e successivamente modificato e cancellato. Affinchè un qualsiasi oggetto abbia queste ultime

proprietà, le classi RivenditoreView e TipologiaView, che permettono la visualizzazione di un singolo oggetto, devono estendere la classe ObjectView per ereditare i metodi onDelete e onEdit.

```
public class RivenditoreView extends ObjectView{
 private class GetAuthorListener implements ObjectListener
 {
  public void onObject(BaseObject object) {
  Tipologia tipologia = (Tipologia)object;
  if( tipologia != null )
  addField("lista␣tipologie",tipologia.getCodice());
  }
 } // fine classe GetAuthorListener

  private final Rivenditore rivenditore;

  public RivenditoreView(Rivenditore story1,DatabaseEditorView
  view)
  {
  super( "<img␣src='item-story.png'hspace='3'>"+ story1.getCodice(), view);
  this.rivenditore = story1;
  addField( "id", rivenditore.getId() );
  addField( "codice", rivenditore.getCodice() );
  addField( "descrizione", rivenditore.getDescrizione() );
  if( rivenditore.getTipologia_id() != null &&
  rivenditore.getTipologia_id().length()>0)
  getView().getObjectFactory().getTipologiaDAO().getById
  (rivenditore.getTipologia_id(),new GetAuthorListener() );
  }

  protected void onDelete() {
  getView().getObjectFactory().getRivenditoreDAO().
  delete(rivenditore);
  }

  protected void onEdit() {
  new RivenditoreDialog( rivenditore, getView() );
  }
} // fine classe RivenditoreView

public class TipologiaView extends ObjectView{
  private final Tipologia user;
  public TipologiaView(Tipologia user1,DatabaseEditorView view)
  {
  super( "<img␣src='item-user.png'hspace='3'>"
  +user1.getCodice(),view );
  this.user = user1;
  addField( "id", user.getId() );
  addField( "name", user.getCodice() );
  addField( "email", user.getDescrizione() );
  }

  protected void onDelete() {
  getView().getObjectFactory().getTipologiaDAO().
  delete(user);
  }
  protected void onEdit() {
  new TipologiaDialog( user, getView() );
  }
}// fine classe TipologiaView
```

Finestre di dialogo per la creazione e la modifica di oggetti

Gli oggetti di tipo Tipologia e Rivenditore utilizzano le finestre di dialogo per la modifica e la creazione di oggetti, chiamando rispettivamente le classi TipologiaDialog e RivenditoreDialog. La finestra TipologiaDialog viene

visualizzata quando gli utenti fanno clic sul comando Edit nella vista TipologiaView e il comando Create nella vista TipologiaListView, mentre la finestra RivenditoreDialog viene visualizzata quando gli utenti fanno clic sul comando Edit nella vista RivenditoreView e sul comando Create nella vista RivendiroreListView. Queste finestre di dialogo sono simili in quanto entrambe hanno bisogno di fornire gli stessi campi e quando sono visualizzate gli utenti devono fare clic sul pulsante OK o Cancella per tornare alla applicazione.

Ad esempio, la finestra di dialogo Rivenditore mostrata nella Figura 4.4 viene visualizzata quando gli utenti fanno clic sul comando create.

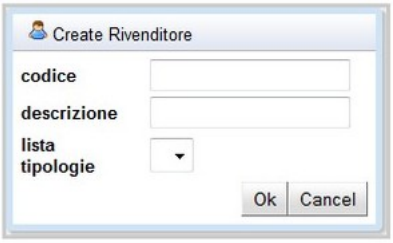

Figura 4.4: Finestra di dialogo per la creazione dell'oggetto Rivenditore

Si noti che ogni campo di questa finestra di dialogo ha una etichetta sulla sinistra e una casella di testo a fianco ad eccezione del campo lista tipologie che utilizza una casella di riepilogo per consentire agli utenti di selezionare dalla lista degli oggetti disponibili le tipologie. Dopo che l'utente compila i campi e fa clic sul pulsante OK, RivenditoreDialog chiama il metodo DAO per il salvataggio. Lo stesso processo viene eseguito per la modifica, e l'applicazione chiama anche il metodo di salvataggio.

Il codice della classe RivenditoreDialog :

```
public class RivenditoreDialog extends ObjectDialogBox{

  Rivenditore rivenditore = new Rivenditore();
  ListBox tipologiaList = new ListBox();

  public RivenditoreDialog(DatabaseEditorView view)
  {
  super("<img␣src='item-rivenditore.png'hspace='3'>CreateRivenditore",view);
  rivenditore = new Rivenditore();
  init();
  }

  public RivenditoreDialog(Rivenditore rivenditore,
  DatabaseEditorView view)
  {
  super("<img␣src='item-story.png'hspace='3'>Edit␣Rivenditore",view );
  this.rivenditore = rivenditore;
  init();
  }

  private void init() {
  addField( "codice", rivenditore.getCodice() );
```

```
  addField( "descrizione", rivenditore.getDescrizione() );
  addField( "lista tipologie", tipologiaList );
  addButtons();
  final String postedby_id = rivenditore.getTipologia_id();
  getView().getObjectFactory().getTipologiaDAO().getAll(new
  CollectionListener(){
  public void onCollection(List list) {
  for( Iterator it = list.iterator(); it.hasNext(); ){
  Tipologia tipologia = (Tipologia)it.next();
  tipologiaList.addItem( tipologia.getCodice(),
  tipologia.getId() );
  if( postedby_id != null &&
  postedby_id.compareTo(tipologia.getId())==0)
  tipologiaList.setSelectedIndex(tipologiaList.getItemCount()-1);
  } // fine ciclo for
  } // fine metodo onCollection
  });

  } // fine metodo init

  public void onSubmit(){
  rivenditore.setCodice( getField(0) );
  rivenditore.setDescrizione( getField(1) );
  rivenditore.setTipologia_id( tipologiaList.getValue
  ( tipologiaList.getSelectedIndex()));
  getView().getObjectFactory().getRivenditoreDAO().
  save( rivenditore );
  }
}// fine classe
```

La classe RivenditoreDialog non estende direttamente la classe DialogBox di GWT, ma estende la classe ObjectDialogBox che aggiunge alcuni metodi di supporto, come ad esempio addField, che possono essere condivisi anche dalla finestra TipologiaDialog. Nella classe RivenditoreDialog sono presenti due costruttori: il primo viene utilizzato quando la finestra di dialogo crea un nuovo oggetto, il secondo accetta un riferimento ad un oggetto Rivenditore e viene utilizzato per modificare l'oggetto. Entrambi i costruttori chiamano il metodo init per inizializzare i campi della finestra di dialogo. Per il campo lista tipologie, viene inviata una richiesta per la lista delle tipologie. Viene creata una classe interna anonima in grado di ricevere la risposta da questa richiesta asincrona. Quando la risposta viene ricevuta, la classe interna anonima popola la casella di riepilogo con i nomi delle tipologie. L'ultimo metodo della classe, onSubmit, viene chiamato quando l'utente fa clic sul pulsante Ok. Una volta che tutti i valori sono stati copiati dal widget all'oggetto, il metodo onSubmit chiama il metodo save sul DAO di rivenditore per il salvataggio.

La classe ObjectDialogBox che estende DialogBox di GWT, costruisce la disposizione dei widget all'interno della finestra di dialogo e fornisce le classi derivate con il metodo addField per personalizzare i campi modificabili. Il codice seguente mostra l'implementazione della classe.

```
public class ObjectDialogBox extends DialogBox{

  private FlexTable fields = new FlexTable();
  private final DatabaseEditorView view;

  //costruttore
```

```
  public ObjectDialogBox(String string,
  DatabaseEditorView view){
  this.view = view;
  setHTML( string );
  setWidget( fields );
  show();
  center();
  }

  public void addField(String name, String value){
  TextBox fieldValue = new TextBox();
  fieldValue.setText(value);
  addField( name, fieldValue );
  }

  public void addField(String name,Widget fieldValue){
  int row = fields.getRowCount();
  Label fieldName = new Label( name );
  fieldName.setStyleName("gwtapps-FieldName");
  fieldValue.setStyleName("gwtapps-FieldValue");
  fields.getCellFormatter().setWidth(row,1,"100%");
  fields.setWidget(row, 0, fieldName);
  fields.setWidget(row, 1, fieldValue);
  }

  public void addButtons(){
  int row = fields.getRowCount();
  HorizontalPanel buttons = new HorizontalPanel();
  fields.setWidget(row, 1, buttons );
  buttons.add( new Button( "Ok", new ClickListener(){
  public void onClick( Widget sender ){
  onSubmit();
  hide();
  }}));

  buttons.add( new Button("Cancel", new ClickListener(){
  public void onClick( Widget sender ){
  hide();
  }}));

  fields.getCellFormatter().setHorizontalAlignment(row, 1,
  HasHorizontalAlignment.ALIGN_RIGHT );

  } // fine metodo addButtons

  public String getField(int row){
  TextBox field = (TextBox)fields.getWidget(row, 1);
  return field.getText();
  }

  public void onSubmit(){}

  public DatabaseEditorView getView() {
  return view;
  }

}// fine classe ObjectDialogBox
```

Un metodo importante da notare in questa implementazione è center() che viene chiamato nel costruttore: questo metodo centra la finestra di dialogo nel mezzo del browser. Se non viene utilizzato, si avrebbe bisogno di impostare la posizione della finestra di dialogo utilizzando il metodo setPosition; in alternativa la finestra viene aggiunta alla fine dell'applicazione e l'utente dovrebbe scorrere tutta la pagina per vederla.

4.2.4 Integrazione con la servlet GWT-RPC

Scrivere il servizio RPC

Utilizzando GWT-RPC, i nostri oggetti del modello vengono automaticamente serializzati quando vengono utilizzati come parametri per una chiamata RPC. Le uniche restrizioni sono date dall'obbligatorietà di implementare l'interfaccia Serializable e dalla presenza di un costruttore senza argomenti. Sul server si possono riutilizzare alcune classi già viste precedentemente, e possiamo addirittura mapparle direttamente al database usando Hibernate, un object-relational mapping (ORM) per Java. Il primo passo per l'attuazione del servizio RPC è quello di dichiarare l'interfaccia di servizio che estenda l'interfaccia RemoteService. Per capire meglio guardare la Figura 3.15

```
public interface RPCObjectFactoryService extends RemoteService
{
/**
* @gwt.typeArgs <com.gwtapps.databaseeditor.client.model.
* BaseObject>
*/
List<BaseObject>getAll( String type );
/**
* @gwt.typeArgs <com.gwtapps.databaseeditor.client.model.
* BaseObject>
*/
List<BaseObject>getAllFrom(String type,String Id,String member);

BaseObject getById( String type, String Id );

void save( BaseObject object );

void delete( String type, String id );

void addTo(String type, String Id, String member,
BaseObject objectToAdd);

}
```

Questa interfaccia ha sei metodi ed ognuno ha il primo parametro che indica il tipo di oggetto che deve essere utilizzato: questo potrà essere Rivenditore o Tipologia.
Si noti l'annotazione gwt.typeArgs che è stata aggiunta. Questa dice al compilatore GWT che gli oggetti della lista restituita devono essere di tipo BaseObject. La classe Tipologia, estendendo BaseObject, permette agli oggetti di essere trasportati in questa lista. Questa annotazione è necessaria per ridurre la quantità di codice generato dal generatore di codice RPC. Se questa non viene specificata, si dovrebbe generare il codice di serializzazione per ogni oggetto che è presente nella lista.

Successivamente, la versione asincrona dell'interfaccia ha bisogno di essere implementata per l'applicazione client poiché ogni metodo chiamato deve essere asincrono:

```
public interface RPCObjectFactoryServiceAsync {

void getAll(String type,AsyncCallback callback);
void getAllFrom( String type, String Id, String member,
AsyncCallback callback );
void getById(String type,String Id, AsyncCallback callback);
void save(BaseObject object,AsyncCallback callback);
void delete(String type,String id,AsyncCallback callback);
void addTo(String type,String Id,String member, BaseObject
objectToAdd, AsyncCallback callback);
}
```

Questa è quasi la stessa interfaccia precedente ad eccezione che qualunque valore ritornato è impostato su void ed il parametro AsyncCallback è aggiunto a ciascun metodo.
Per implementare il servizio sul server, abbiamo bisogno di implementare l'interfaccia RPCObjectFactoryService che estenda il RemoteServiceServlet di GWT:

```
public class RPCObjectFactoryServiceImpl
extends RemoteServiceServlet
implements RPCObjectFactoryService {

  public void addTo(String type, String Id, String member,
  BaseObject objectToAdd) {
  if( type.equals("Rivenditore") ){
  PersistenceManager pm = PMF.get().getPersistenceManager();
  pm.setDetachAllOnCommit(true);
  try {
    Rivenditore story = (Rivenditore)pm.getObjectById(
    Rivenditore.class,Long.parseLong(Id) );
    long userId = Long.parseLong(objectToAdd.getId());
    Tipologia user = (Tipologia)pm.getObjectById(
    Tipologia.class,userId );
    pm.makePersistent(story);
  } finally {
  pm.close();
  }
  }}

  public List<BaseObject> getAll(String type){
  List result = null;
  PersistenceManager pm = PMF.get().getPersistenceManager();
  pm.setDetachAllOnCommit(true);
  try {
  Query query = null;
  if( type.equals("Tipologia"))
  query = pm.newQuery(Tipologia.class);
  else
  query = pm.newQuery(Rivenditore.class);
  result = new ArrayList<BaseObject>((List<BaseObject>)
  query.execute());
  } finally {
  pm.close();
  }
  return result;}

  public List<BaseObject> getAllFrom(String type,String Id,
  String member) {
  List result = new ArrayList();
  if( type.equals("Rivenditore") ){
  PersistenceManager pm = PMF.get().getPersistenceManager();
  pm.setDetachAllOnCommit(true);
  try {
  Rivenditore story = (Rivenditore)pm.getObjectById(
  Rivenditore.class,Long.parseLong(Id) );
  result = new ArrayList<BaseObject>();
```

```
  } finally {
  pm.close();
  }}
  return result;}

  public BaseObject getById(String type, String Id) {
  BaseObject result = null;
  PersistenceManager pm = PMF.get().getPersistenceManager();
  pm.setDetachAllOnCommit(true);
  try {
  if( type.equals("Tipologia"))
  result = pm.getObjectById(Tipologia.class,
  Long.parseLong(Id) );
  else
  result = pm.getObjectById(Rivenditore.class,
  Long.parseLong(Id) );
  } finally {
  pm.close();
  }
  return result;}

  public void save(BaseObject object) {
  PersistenceManager pm = PMF.get().getPersistenceManager();
  try {
  pm.makePersistent(object);
  } finally {
  pm.close();
  }}

  public void delete(String type, String id) {
  PersistenceManager pm = PMF.get().getPersistenceManager();
  try {
  if( type.equals("Tipologia"))
  pm.deletePersistent( pm.getObjectById(Tipologia.class,
  Long.parseLong(id) ));
  else
  pm.deletePersistent(pm.getObjectById(Rivenditore.class,
  Long.parseLong(id) ));
  } finally {
  pm.close();
  }}

}// fine classe
```

Ora che abbiamo un servizio RPC, abbiamo bisogno di collegarlo all'implementazione DAO in modo che possa essere utilizzato dalla vista dell'applicazione. Fortunatamente l'interfaccia ObjectDAO ha una stretta corrispondenza con l'interfaccia del servizio, e gli oggetti del modello possono essere utilizzati automaticamente con il servizio, quindi questo lavoro è abbastanza semplice.
Il seguente codice implementa l'interfaccia ObjectDAO per RPC:

```
protected class RPCObjectDAO implements ObjectDAO{

  private final String type;
  public RPCObjectDAO( String type ){
  this.type = type;
  }

  public void getAll(CollectionListener handler){
  service.getAll(type, new CollectionCallback( handler ) );
  }

  public void getAllFrom( BaseObject object, String member,
  CollectionListener handler){
```

```
  service.getAllFrom(type, object.getId(), member,
  new CollectionCallback( handler ) );
  }

  public void getById(String id, ObjectListener handler){
  service.getById(type, id, new ObjectCallback( handler ) );
  }

  public void save(BaseObject object){
  service.save( object , new RefreshCallback() );
  }

  public void delete(BaseObject object) {
  service.delete(type,object.getId(),new RefreshCallback() );
  }

  public void addTo(BaseObject object,String member,BaseObject
  objectToAdd) {
  service.addTo( type,object.getId(),member,objectToAdd,
  new RefreshCallback() );
  }
} // fine classe
```

Il DAO accetta una stringa come parametro, che per questa applicazione è Tipologia o Rivenditore e utilizza il parametro in ogni chiamata al servizio. Il servizio è una variabile membro della classe esterna RPCObjectFactory :

```
public class RPCObjectFactory implements ObjectFactory{

  private ObjectFactoryListener listener;

  protected class RPCObjectDAO implements ObjectDAO{
  // codice scritto precedentemente
  }

  protected class CollectionCallback
  implements AsyncCallback{
  private CollectionListener handler;
  public CollectionCallback(CollectionListener handler){
  this.handler = handler;}

  public void onFailure(Throwable exception){
     GWT.log("error",exception);listener.onLoadingFinish();
   }

  public void onSuccess(Object result) {
    handler.onCollection((List)result);
    listener.onLoadingFinish();
    }
  }

  protected class ObjectCallback implements AsyncCallback{
    private ObjectListener handler;
    public ObjectCallback(ObjectListener handler) {
    this.handler = handler;
    }

  public void onFailure(Throwable exception){
    GWT.log( "error", exception );
    listener.onLoadingFinish(); }

  public void onSuccess(Object result) {
    handler.onObject((BaseObject)result);
    listener.onLoadingFinish();
    }
  }

  protected class RefreshCallback implements AsyncCallback{
```

```
  public void onFailure(Throwable exception){
     GWT.log( "error", exception );
    listener.onLoadingFinish();
    }

  public void onSuccess(Object result) {
    listener.onRefresh();
    listener.onLoadingFinish();
    }
  }
  private RPCObjectDAO rivenditoreDAO = new
  RPCObjectDAO("Rivenditore");

  private RPCObjectDAO tipologiaDAO = new
  RPCObjectDAO("Tipologia");

  private RPCObjectFactoryServiceAsync service;

  public RPCObjectFactory(String baseUrl) {
    service = (RPCObjectFactoryServiceAsync)
    GWT.create( RPCObjectFactoryService.class );
  ServiceDefTarget endpoint = (ServiceDefTarget) service;
    endpoint.setServiceEntryPoint( baseUrl );
  }

  public ObjectDAO getRivenditoreDAO() {
  return rivenditoreDAO;
  }

  public ObjectDAO getTipologiaDAO() {
  return tipologiaDAO;
  }

  public void setListener(ObjectFactoryListener listener){
  this.listener = listener;
  }
} // fine classe
```

Per gestire i callbacks dalle chiamate RPC, la classe RPCObjectFactory implementa tre classi interne che estendono interfaccia AsyncCallback di GWT. La classe CollectionCallback viene utilizzata per le chiamate RPC che si aspettano un elenco di oggetti come valore di ritorno. La classe ObjectCallback viene utilizzata per le chiamate RPC che si aspettano un singolo oggetto come valore di ritorno. La classe RefreshCallback viene utilizzata quando l'elemento attualmente visualizzato nell'interfaccia dovrà essere aggiornato. In questa applicazione i metodi save e delete utilizzano questa callback.

Con la servlet impostata e la RPCObjectFactory collegata al livello DAO, siamo in grado di eseguire l'applicazione su RPC aggiungendo il RPCObjectFactory all'Entry Point dell'applicazione in questo modo:

```
public class RivTipWebApp implements EntryPoint{

public void onModuleLoad(){

  //creazione della vista
  DatabaseEditorView view = new DatabaseEditorView();
  RootPanel.get("databaseEditorView").add( view );
  RPCObjectFactory objectFactory = new RPCObjectFactory
  (GWT.getModuleBaseURL()+"objectFactory" );
```

```
  //passa alla lista l'object factory
  view.setObjectFactory( objectFactory );
  }
}
```

A questo punto, però, non abbiamo collegato la servlet con il database. Questo viene gestito nella prossima sezione utilizzando Hibernate.

Utilizzo di Hibernate per memorizzare il modello

Hibernate è uno strumento object-relational mapping per le applicazioni Java e consente di esegue il mapping object-oriented da classi Java a relazioni di un database relazionale. Questa applicazione utilizza Hibernate per mappare i campi dagli oggetti Rivenditore e Tipologia alle tabelle del database.

Due elementi importanti per questa applicazione sono gli elementi di mappatura. Ognuno punta a un file di mapping XML che definisce come una classe deve essere mappata nel database. C'è un file di mapping per la classe Tipologia chiamato Tipologia.hbm.xml e un file di mappatura per la classe Rivenditore chiamato Rivenditore.hbm.xml.

Il file Tipologia.hbm.xml è il seguente:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate
/Hibernate␣Mapping␣DTD␣3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.gwtapps.rivtipWebapp.client.model.Tipologia"
table="Tipologie">
<id column="id" name="id" type="java.lang.String">
<generator class="org.hibernate.id.UUIDHexGenerator">
</generator>
</id>
<property name="codice"/>
<property name="descrizione"/>
</class>
</hibernate-mapping>
```

Il primo elemento della mappatura indica il nome della classe (compreso il path) che viene mappata ed il nome della tabella. All'interno del primo elemento c'è la mappatura ID, che imposta il tipo ed il nome dell'identificatore; successivamente viene definito generator class per generare automaticamente un nuovo ID utente quando nuovi oggetti vengono salvati. I restanti tre elementi indicano i campi che devono essere mappati.

Il file Rivenditore.hbm.xml è implementato così:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate
/Hibernate␣Mapping␣DTD␣3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.gwtapps.rivtipWebapp.client.model.Rivenditore"
```

```
table="Rivenditori">
<id column="id" name="id" type="java.lang.String">
<generator class="org.hibernate.id.UUIDHexGenerator">
</generator>
</id>
<property name="codice"/>
<property name="descrizione"/>
<one-to-many
  class="com.gwtapps.rivtipWebapp.client.model.Tipologia"
column="tipologia_id"/>
</list>
<property name="tipologia_id"/>
</class>
</hibernate-mapping
```

Il file Rivenditore.hbm.xml ha un layout simile al file Tipologia.hbm.xml, in cui la classe e il nome della tabella sono definiti insieme con l'ID generato automaticamente e tre i campi. Inoltre, è definita la relazione molti a uno con la classe Tipologia attraverso l'attributo tipologia_id che rappresenta la chiave primaria della relazione.

4.3 Avvio applicazione e risultati ottenuti

Ora che abbiamo scritto la nostra applicazione in linguaggio Java proviamo ad eseguirla. Per prima cosa è necessario effettuare la compilazione di tutti i file tramite GWT. Per compilare il progetto occorre:

1. Selezionare il progetto.
2. Andare sull'icona di GWT e selezionare GWT Compile Project. Apparirà una finestra nella quale è possibile scegliere il nome del progetto e l'Entry Point. Una volta impostati cliccare il pulsante Compile.
3. Nella console appare il risultato della compilazione :

```
Compiling module com.gwtapps.rivtipWebapp.RivTipWebApp
Compiling 6 permutations
Compiling permutation 0...
Compiling permutation 1...
Compiling permutation 2...
Compiling permutation 3...
Compiling permutation 4...
Compiling permutation 5...
Compile of permutations succeeded
Compilation succeeded -- 37,227s
```

Quando la compilazione è avvenuta il progetto è pronto per essere eseguito. Per eseguirlo :

1. Selezionare il progetto.
2. Tasto destro, Run As, Web Application.
3. Compare una finestra nella quale è possibile scegliere il file HTML da eseguire. Una volta scelto cliccare il pulsante OK. Il file e il progetto hanno sempre lo stesso nome.

4. Apparirà nella finestra Development Mode un indirizzo del tipo : `http://127.0.0.1:8888/RivTipWebApp.html?gwt.codesvr=127.0.0.1:9997`.

5. Copiare ed incollare l'indirizzo nella barra degli indirizzi del Web browser. Se utilizziamo il browser Google Chrome apparirà in Figura 4.5.

Figura 4.5: GWT Plugin per il Web browser

   Effettuare il download e riavviare il browser. Se non compare in automatico la pagina HTML con l'indirizzo precedentemente inserito reinserirlo.

6. La pagina HTML viene caricata e l'applicazione è pronta per essere eseguita.

Ecco alcune immagini dell'applicativo Web ottenuto :

Figura 4.6: Schermata iniziale dell'applicazione

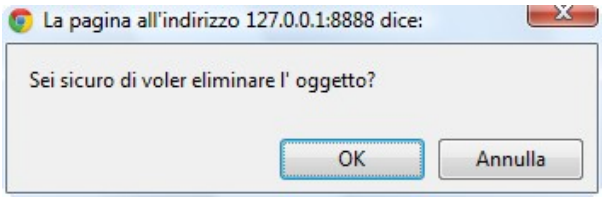

Figura 4.7: Messaggio di conferma per l'eliminazione di un qualsiasi oggetto dell'applicazione

Figura 4.8: ListView di Tipologie

Figura 4.9: ListView di Rivenditori

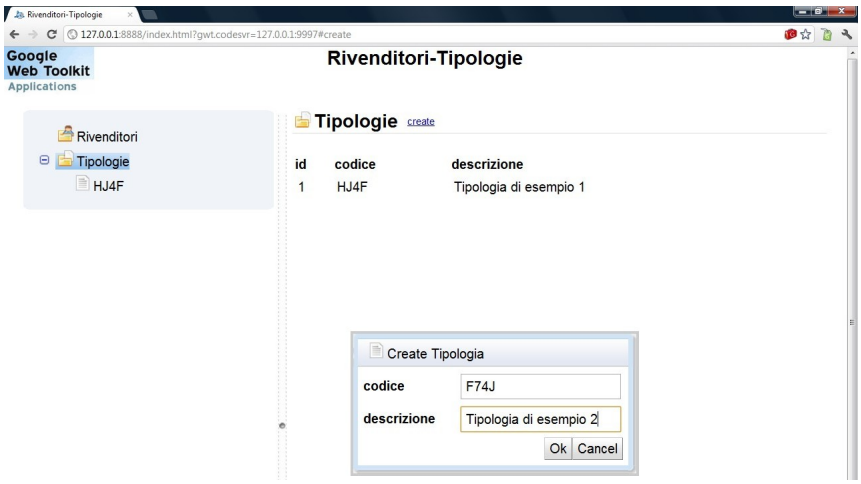

Figura 4.10: Creazione dell'oggetto Tipologia

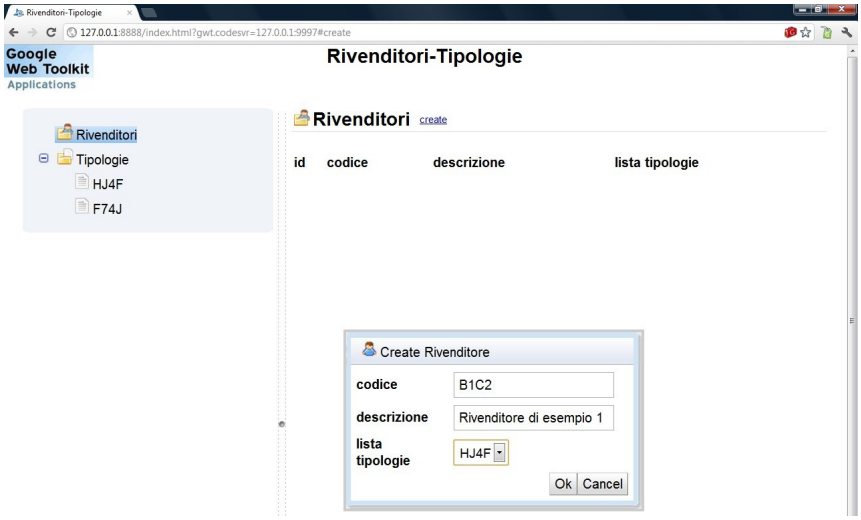

Figura 4.11: Creazione dell'oggetto Rivenditore

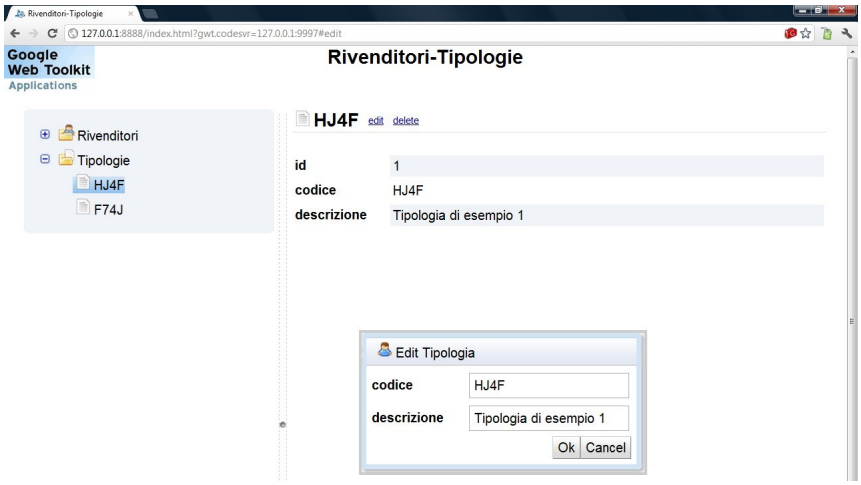

Figura 4.12: Modifica dell'oggetto Tipologia

Figura 4.13: ObjectView di Tipologia

Conclusioni

Posso sicuramente ritenermi soddisfatto di come si è svolta l'intera attività di tirocinio e dei risultati ottenuti. Innanzitutto, ho potuto affrontare problemi pratici che riguardano la produzione di software, sia dal lato progettuale sia da quello implementativo. Ho potuto costatare che la scelta della tecnologia Java spesso è associata a realtà aziendali e la sua complessità, purtroppo, ne scoraggia l'utilizzo agli sviluppatori di applicazioni (web e non), i quali preferiscono soluzioni più semplici come PHP, ASP e altri linguaggi di scripting. Inoltre, ho potuto rafforzare l'importanza della standardizzazione del processo di produzione di software: l'adozione di framework di sviluppo permette di avere una struttura adattabile e flessibile e di risolvere una certa gamma di problematiche.

Tramite questa relazione si vuole creare e fornire un valido manuale consultabile per comprendere le applicazioni web e il framework Java GWT. All'interno di questa tesi sono state elencate e descritte tutte le azioni necessarie per garantire la corretta creazione dell'applicazione rendendo così disponibile, in un'unica guida, sia i concetti teorici che quelli pratici utilizzati durante il periodo di tirocinio.

Visto l'uso sempre più diffuso di dati e di database nella vita aziendale di tutti i giorni, tale applicazione sarà la base di successivi sviluppi di applicazioni web, destinate alla gestione dei dati, in quanto permette di collegare due concetti diversi (in questo caso Rivenditori e Tipologie) qualsiasi sia l'ambito a cui si riferiscono. I concetti possono essere ampliati e sviluppati nel dettaglio, aumentando così la realtà di interesse decisa dall'azienda, la qualità e la personalizzazione dell'applicativo web.

Bibliografia

[1] Paolo Atzeri, Stefano Ceri, Stefano Paraboschi, Riccardo Torlone,"Basi di dati - Modelli e linguaggi di interrogazione" Terza Edizione, McGraw-Hill.

[2] Cay Horstmann, "Concetti di informatica e fondamenti di Java", Apogeo.

[3] Alure D., Crupi J., Malks D., Core J2EE Patterns, Sun Microsystem.

[4] Robert Hanson, Adam Tacy, "GWT in Action: Easy Ajax with the Google Web Toolkit", Manning Publication Co. .

[5] Robert Cooper Robert, Charles Collins. "GWT in Practice", Manning Publications Co. .

[6] Ryan Dewsbury, "Google Web Toolkit Application", Prentice Hall.

[7] Robin Roos, "Java Data Objects", Pearson Education Inc.

Sitografia

[w1] Pagina ufficiale di GWT (http://developers.google.com/Web-toolkit).

[w2] DataAccessObject (http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html).

[w3] SpringSource, Spring Framework (http://www.springsource.org).

[w4] Pagina wiki .it su AJAX(http://it.wikipedia.org/wiki/AJAX).

[w5] Sito ufficiale di AppEngine (http://code.google.com/intl/it-IT/appengine).

[w6] JBoss Community, Hibernate(http://www.hibernate.org).

[w7] Sito ufficiale di AppEngine (http://code.google.com/intl/it-IT/appengine).

[w8] Api di GWT (http://java.html.it/articoli).

[w9] JEE tutorial (http://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html) .

[w10] Java Persistence Api (http://www.giuseppesicari.it/articoli/jpa-java-persistence-api) .

[w11] Giorgio Natili, Introduzione al design pattern e Model View Presenter[3].

[w12] Marco Siniscalco, Pattern MVP (http://marcosiniscalco.wordpress.com/2011/12/02/patternmvp-ed-asp-net-1).

[3] (http://actionscript.it/it/index.cfm/tutorials/design-patterns-e-model-view-presenter)

Elenco delle figure

Elenco delle tabelle