Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali

Corso di Laurea in Scienze di Internet

Sistema per la gestione di basi di dati relazionali da XML

Tesi di Laurea in Programmazione di Internet

Relatore:
Chiar.mo Prof.
ANTONIO MESSINA

Presentata da:
RAFFAELE BACOLINI

Sessione Prima
Anno Accademico 2009/2010

Dedico questo momento
a mio nonno Raffaele
e a mio nonno Mario

Indice

Elenco delle figure

Introduzione

Nello sviluppo di sistemi informatici si sono affermate numerose tecnologie, che vanno utilizzate in modo combinato e, possibilmente sinergico.

Da una parte, i sistemi di gestione di basi di dati relazionali consentono una gestione efficiente ed efficace di dati persistenti, condivisi e transazionali. Dall'altra, gli strumenti e i metodi orientati agli oggetti (linguaggi di programmazione, ma anche metodologie di analisi e progettazione) consentono uno sviluppo efficace della logica applicativa delle applicazioni

E' utile in questo contesto spiegare che cosa s'intende per sistema informativo e sistema informatico.

Sistema informativo: L'insieme di persone, risorse tecnologiche, procedure aziendali il cui compito è quello di produrre e conservare le informazioni che servono per operare nell'impresa e gestirla.

Sistema informatico: L'insieme degli strumenti informatici utilizzati per il trattamento automatico delle informazioni, al fine di agevolare le funzioni del sistema informativo. Ovvero, il sistema informatico raccoglie, elabora, archivia, scambia informazione mediante l'uso delle tecnologie proprie dell'Informazione e della Comunicazione (ICT): calcolatori, periferiche, mezzi di comunicazione, programmi. Il sistema informatico è quindi un componente del sistema informativo.

Le informazioni ottenute dall'elaborazione dei dati devono essere salvate da qualche parte, in modo tale da durare nel tempo dopo l'elaborazione. Per realizzare questo scopo

viene in aiuto l'informatica.
I dati sono materiale informativo grezzo, non (ancora) elaborato da chi lo riceve, e possono essere scoperti, ricercati, raccolti e prodotti. Sono la materia prima che abbiamo a disposizione o produciamo per costruire i nostri processi comunicativi. L'insieme dei dati è il tesoro di un'azienda e ne rappresenta la storia evolutiva.

All'inizio di questa introduzione è stato accennato che nello sviluppo dei sistemi informatici si sono affermate diverse tecnologie e che, in particolare, l'uso di sistemi di gestione di basi di dati relazionali comporta una gestione efficace ed efficiente di dati persistenti.

Per persistenza di dati in informatica si intende la caratteristica dei dati di sopravvivere all'esecuzione del programma che li ha creati. Se non fosse cosi, i dati verrebbero salvati solo in memoria RAM e sarebbero persi allo spegnimento del computer.

Nella programmazione informatica, per persistenza si intende la possibilità di far sopravvivere strutture dati all'esecuzione di un programma singolo. Occorre il salvataggio in un dispositivo di memorizzazione non volatile, come per esempio su un file system o su un database.

In questa tesi si è sviluppato un sistema che è in grado di gestire una base di dati gerarchica o relazionale consentendo l'importazione di dati descritti da una grammatica DTD. Nel capitolo 1 si vedranno più in dettaglio cosa di intende per Sistema Informativo, modello client-server e sicurezza dei dati. Nel capitolo 2 parleremo del linguaggio di programmazione Java, dei database e dei file XML. Nel capitolo 3 descriveremo un linguaggio di analisi e modellazione UML con esplicito riferimento al progetto sviluppato. Nel capitolo 4 descriveremo il progetto che è stato implementato e le tecnologie e tools utilizzati.

Capitolo 1

Scambio di dati

1.1 Il sistema informativo

Nel contesto aziendale di oggi, il sistema informativo riveste un ruolo di fondamentale importanza, perché una buona implementazione di questo può rappresentare un potenziale vantaggio rispetto alla concorrenza.
Il sistema informativo può essere definito come "l'insieme di persone, apparecchiature, procedure aziendali il cui compito è quello di produrre e conservare le informazioni che servono per operare nell'impresa e gestirla".
Il sistema informativo quindi interagisce con tutti gli altri processi aziendali, ed ha come oggetto di trattazione l'informazione.
L'informazione è la principale risorsa scambiata, selezionata ed elaborata nelle attività gestionali di coordinamento e controllo.
Essa è una risorsa immateriale e non tangibile, ma per esistere nel mondo fisico, deve essere rappresentata in modo fisico. La rappresentazione dell'informazione in termini fisici è costituita dai dati.

I Dati, che possono essere di natura digitale ma anche di diversa natura, come ad esempio quella cartacea, vengono intesi come i "fatti grezzi" a partire dai quali si possono

dedurre informazioni utili alle significative decisioni aziendali. Per essere utili i dati devono essere organizzati, archiviati e classificati in modo da essere facilmente reperibili. Lo scopo principale di una base di dati (in inglese database) è proprio questo. Un database è definito come un insieme strutturato di dati che, in un momento, fornisce una rappresentazione semplificata di una realtà in evoluzione.

Un database consente la gestione dei dati stessi (l'inserimento, la ricerca, la cancellazione ed il loro aggiornamento) da parte di applicazioni software.

Una base di dati, oltre ai dati veri e propri, deve contenere anche le informazioni sulle loro rappresentazioni e sulle relazioni che li legano.

Spesso la parola "database" è, impropriamente, usata come abbreviazione dell'espressione Database Management System (DBMS), i due concetti sono differenti.

In un sistema informatico, la base di dati può essere manipolata direttamente da programmi applicativi (applicazioni), interfacciandosi direttamente con il sistema operativo, strategia adottata universalmente fino agli anni settanta, e tuttora impiegata quando i dati hanno una struttura molto semplice, o quando sono elaborati da un solo programma applicativo, oppure attraverso appositi sistemi software, detti Sistemi per la gestione di basi di dati (i DBMS appunto).

L'introduzione dei Database Management System, a partire dalla fine degli anni Sessanta, fu motivata dal fatto che la gestione di basi di dati complesse condivise da più applicazioni richiedeva l'utilizzo di un unico sistema software in grado di accedere in maniera efficace a grandi quantità di dati, e di gestire l'accesso garantendo l'integrità. L'utilizzo dei database e la relativa gestione attraverso i DBMS rispetto alle applicazioni software possiede vantaggi sotto molti punti di vista; Sotto il punto di vista della facilità di progettazione, l'utilizzo dei database permette:

- di creare diverse "visibilità" dei dati adatte a ciascuna applicazione;

- di gestire in maniera automatica e controllata i dati, come ad esempio in caso di cancellazione di un record da cui ne dipendono logicamente degli altri, porta anche

la cancellazione di quelli dipendenti

- di estendere i linguaggi esistenti o linguaggi ad alto livello, orientati a facilitare la manipolazione dei dati;

Sotto il punto di vista della facilità di gestione, il database permette:

- di modificare e arricchire le strutture di dati nel tempo senza la modifica delle applicazioni esistenti;

- di aggiungere nuovi campi all'interno di tracciati record già esistenti senza modificare le applicazioni che già li utilizzavano;

- di rendere esplicite le relazioni logiche tra dati, che altrimenti restano nascoste all'interno delle elaborazioni delle varie applicazioni;

Sotto il punto di vista dell'efficacia delle implementazioni il database permette:

- di ridurre la ridondanza dei dati, a differenza dell'utilizzo delle applicazioni in cui uno stesso dato può essere memorizzato su più di una di esse, e quindi una conseguente riduzione dell'occupazione fisica di memoria;

- di garantire una congruenza dei dati.

Sotto il punto di vista dell'utente il database garantisce:

- la disponibilità di un linguaggio per definire le caratteristiche dei dati e le strutture logiche che li legano;

- la disponibilità di estensioni dei linguaggi di programmazione che semplificano le operazioni sui dati;

- la disponibilità di accedere direttamente ai dati;

A causa di questi vantaggi, oggi, l'utilizzo dei DBMS per la gestione delle basi di dati, è molto più diffuso rispetto all'utilizzo di applicazioni specifiche, soprattutto in molti campi come la contabilità, le risorse umane, la finanza, la gestione di rete o la telefonia. Un DBMS garantisce anche la sicurezza e l'integrità del database. Ogni transazione, infatti, ossia ogni sequenza di operazioni che hanno un effetto globale sul database gode delle cosiddette proprietà **ACID**:

- **"A"** sta per Atomicity (Atomicità), ossia se tutte le operazioni della sequenza terminano con successo si ha il commit e la transazione è conclusa con successo, ma se anche solo una di queste operazioni fallisce, l'intera transazione é abortita;
- **"C"** sta per Consistency (Consistenza), ossia ogni transazione, una volta terminata, deve lasciare il database in uno stato consistente;
- **"I"** sta per Isolation (Isolamento), ossia nel caso di esecuzione di transazioni concorrenti, non ci deve essere influenzamento dell'una sulle altre;
- **"D"** sta per Durability (Curabilità), ossia gli effetti sul database prodotti da una transazione terminata con successo sono permanenti, quindi non possono essere compromessi da eventuali malfunzionamenti.

1.2 Comunicazione tra Client e Server

Prima di trattare gli argomenti principali di questa tesi è utile dare una definizione formale del modello client server. Il modello client server, oggi il modello di riferimento per il networking e le applicazioni di database, si contrappone come filosofia e come struttura al Mainframe. Nel modello di computing del Mainframe vi è un server centrale, detto appunto Mainframe, che esegue tutte le operazioni sui dati e usa terminali "stupidi" (senza capacità propria di elaborazione) come unità di visualizzazione e di inserimento dei dati.

Figura 1.1: Modello client-server

Il sistema client server, invece, è un tipo di modello implementato da applicazioni di rete in cui una macchina client, con capacità di elaborazione propria, istanzia l'interfaccia utente (e non solo, come vedremo) dell'applicazione e si connette con una macchina server o ad un sistema di database per richiedere risorse in esso contenute. La macchina server permette ad un certo numero di client di condividere le risorse, lasciando che sia il server a gestire gli accessi alle risorse per evitare conflitti tipici dei primi sistemi informatici.

Ad esempio il browser che richiede una determinata pagina web attraverso internet ad un server dove tale pagina è memorizzata, rappresenta un client che richiede una risorsa ad un server. Per introdurci nell'analisi del modello client server è utile dare una classificazione delle funzioni fondamentali che un generico applicativo svolge. Queste possono essere suddivise in tre blocchi principali:

- **Presentazione**, ossia la visualizzazione tramite interfaccia grafica ed interazione con l'utente;
- **Applicazione**, ossia elaborazione dell'informazione;
- **Data Management**, ossia acquisizione e mantenimento dell'informazione;

Secondo il tipo di distribuzione di queste funzioni si possono individuare diversi livelli:

- Il primo livello che si può individuare è quello in cui tutta la parte computazionale è spostata sul server, mentre il client rappresenta solamente un display.

- Il secondo livello individuato è molto simile al primo, ma in questa situazione il server non implementa in alcun modo la funzione di presentazione; un esempio può essere un browser che interpreta il codice HTML presente sul server: è implicata quindi l'esistenza di una rete tra client e server.

- Il terzo livello individuabile, prevede lo spostamento di una parte di applicazione anche sul client, che quindi non potrà più avere un hardware troppo "thin", ma dovrà essere dotato di adeguate capacità di elaborazione e di caching dell'informazione.

- Il quarto livello rappresenta il sistema client server "completo" . La particolarità di questa configurazione, venendo lasciata sul server solo la funzione di Data Management, è di produrre il completo spostamento dell'informazione da elaborare sul client. Una soluzione di questo tipo è stata resa possibile grazie ai progressi fatti a livello di Data Management (ODBC e driver nativi di acceso al database, linguaggio SQL, Visual Basic).

- Il quinto ed ultimo livello che si può individuare è il cosiddetto "fat client" che rappresenta un vero e proprio database distribuito. Parte dei dati, infatti, risiedono sul client che è capace di gestirli, modificarli e memorizzarli sia localmente che in remoto.

Secondo il tipo di applicazione di rete che si vuole sviluppare è opportuno scegliere il livello più appropriato. Quando un client si connette direttamente ad un server o ad un database, si dice **2-tier architecture** (architettura a 2 livelli). Recentemente però sono state sviluppate nuove architetture in cui il client, che implementa solamente la funzione di presentazione (thin client), si connette ad un server, che implementa la funzione di

applicazione, il quale comunica transitivamente (ossia successivamente) con un database server, che implementa la funzione di Data Management, memorizzando i dati utilizzati dall'applicazione. Tale architettura è chiamata **3-tier architecture**.

Il collegamento tra client e server è implementato per mezzo di una rete di comunicazione utilizzando un determinato protocollo. Tale protocollo può essere in chiaro, e quindi i dati in transito possono essere intercettati, oppure in certi casi crittografato, proteggendo i dati in transito dalle intercettazioni.

Nella comunicazione orientata alla connessione il client e il server attuano il cosiddetto *handshaking*, ossia una serie di procedure in cui le due macchine si scambiano pacchetti di controllo prima di spedire i dati per prepararsi alla comunicazione. La comunicazione orientata alla connessione è utilizzata, tra gli altri dal protocollo di rete TCP (Transmission Control Protocol).

La comunicazione senza connessione, invece, non prevede nessun *handshaking*, e i dati vengono direttamente inviati sulla rete. Questo tipo di comunicazione è utilizzato, tra gli altri, dal protocollo di rete UDP (User Datagram Protocol).

La comunicazione orientata alla connessione è più lenta rispetto a quella senza connessione, a causa dei pacchetti di *handshaking*, ma risulta più affidabile in caso di perdita nella rete di dati o errori. In definitiva si può dire che la quasi totalità delle applicazioni distribuite oggi è implementata seguendo il modello client server.

1.3 Cosa intendiamo per sicurezza

La sicurezza informatica è la tutela del sistema informativo dalla violazione da parte di persone non autorizzate al fine di garantire confidenzialità, autenticità, integrità e disponibilità dei dati. Sebbene questa definizione sia chiara e concisa, in realtà, la sicurezza non è una caratteristica assoluta ma una misura che si tenta si massimizzare valutando attentamente l'equilibrio fra costo del rischio e costo della difesa. Bisogna

rendersi conto che, come insegna l'ingegneria del software ormai da qualche anno, un programma 'zero risk' è, oltre che irrealizzabile, inutile, in quanto implica un costo di progettazione, sviluppo e testing decisamente maggiore del costo, inteso come perdita di profitto o di immagine, che si avrebbe nel caso di intrusione o manipolazione del sistema. La protezione dagli attacchi informatici viene ottenuta agendo su più livelli: innanzitutto a livello fisico e materiale, ponendo i server in luoghi il più possibile sicuri, dotati di sorveglianza e/o di controllo degli accessi; anche se questo accorgimento fa parte della sicurezza normale e non della sicurezza informatica. Il secondo livello è normalmente quello logico che prevede l'autenticazione e l'autorizzazione di un'entità che rappresenta l'utente nel sistema.

Molti ex-hacker/cracker sono oggi dirigenti di società di sicurezza informatica o responsabili di questa in grandi multinazionali. Ciò dimostra quello che molti dicono e scrivono: per capire le strategie migliori di sicurezza informatica è necessario prima entrare nella mentalità dell'attaccante per poterne prevedere ed ostacolare le mosse. Proprio sulla base di queste osservazioni, quando si parla di sicurezza informatica spesso si distinguono i concetti di sicurezza passiva e di sicurezza attiva.

Sicurezza passiva

Per sicurezza passiva normalmente si intendono le tecniche e gli strumenti di tipo difensivo, ossia quel complesso di soluzioni il cui obiettivo è quello di impedire che utenti non autorizzati possano accedere a risorse, sistemi, impianti, informazioni e dati di natura riservata. Il concetto di sicurezza passiva pertanto è molto generale: ad esempio, per l'accesso a locali protetti, l'utilizzo di porte di accesso blindate, congiuntamente all'impiego di sistemi di identificazione personale, sono da considerarsi componenti di sicurezza passiva.

Sicurezza attiva

Per sicurezza attiva si intendono, invece, le tecniche e gli strumenti mediante i quali le informazioni ed i dati di natura riservata sono resi intrinsecamente sicuri, proteggendo gli stessi sia dalla possibilità che un utente non autorizzato possa accedervi (confidenzialità), sia dalla possibilità che un utente non autorizzato possa modificarli (integrità).

È evidente che la sicurezza passiva e quella attiva sono tra loro complementari ed entrambe indispensabili per raggiungere il desiderato livello di sicurezza di un sistema. Le possibili tecniche di attacco sono molteplici, perciò è necessario usare contemporaneamente diverse tecniche difensive per proteggere un sistema informatico, realizzando più barriere fra l'attaccante e l'obiettivo.

Spesso l'obiettivo dell'attaccante non è rappresentato dai sistemi informatici in sé, quanto piuttosto dai dati in essi contenuti, quindi la sicurezza informatica deve preoccuparsi di impedire l'accesso ad utenti non autorizzati, ma anche a soggetti con autorizzazione limitata a certe operazioni, per evitare che i dati appartenenti al sistema informatico vengano copiati, modificati o cancellati.

Le violazioni possono essere molteplici: vi possono essere tentativi non autorizzati di accesso a zone riservate, furto di identità digitale o di file riservati, utilizzo di risorse che l'utente non dovrebbe potere utilizzare ecc. La sicurezza informatica si occupa anche di prevenire eventuali Denial of service (DoS). I DoS sono attacchi sferrati al sistema con l'obiettivo di rendere non utilizzabili alcune risorse in modo da danneggiare gli utenti del sistema. Per prevenire le violazioni si utilizzano strumenti hardware e software.

Caratteristiche di sicurezza

Due caratteristiche fondamentali esplicano il concetto di sicurezza:

- **Safety** (sicurezza): una serie di accorgimenti atti ad eliminare la produzione di danni irreparabili all'interno del sistema;

- **Reliability** (affidabilità): prevenzione da eventi che possono produrre danni di qualsiasi gravità al sistema.

Capitolo 2

Tecnologie Utilizzate

2.1 Java

Il linguaggio java

Java [17] è un linguaggio orientato agli oggetti e, da questo punto di vista, l'impostazione data risulta piuttosto rigorosa, dato che non è possibile fare ricorso a organizzazioni miste del codice (in parte a oggetti, in parte strutturate) come avviene in altri linguaggi, primo fra tutti il C++. La struttura sintattica obbliga inoltre ad adottare un formalismo ben preciso, che porta nella maggior parte dei casi a un codice elegante e ben strutturato . La filosofia Object Oriented di Java ha come obiettivo la creazione di un linguaggio estremamente semplice, facile da apprendere volto a eliminare quei costrutti pericolosi che portano in certi casi a situazioni non facili da gestire o prevedere. Ad esempio, oltre alla ben nota mancanza dei puntatori ad aree di memoria, in Java sono stati eliminati costrutti come l'ereditarietà multipla, i template e qualche altro elemento di minore importanza. La programmazione orientata agli oggetti (OOP, Object Oriented Programming) [20] è un paradigma di programmazione, che prevede di raggruppare in un'unica entità (la classe) sia le strutture dati che le procedure che operano su di esse, creando per l'appunto un oggetto software dotato di proprietà (dati) e metodi (procedure) che

operano sui dati dell'oggetto stesso. La programmazione orientata agli oggetti può anche essere vista come una modellazione software degli oggetti del mondo reale o del modello astratto da riprodurre .

Principali Caratteristiche

Analizzando Java dal punto di vista del linguaggio si possono individuare alcune caratteristiche fondamentali: la prima e più importante è che il codice compilato (bytecode), senza necessità di ricompilazioni, è eseguibile su ogni tipo di piattaforma hardware-software che metta a disposizione una macchina virtuale Java. Dato che attualmente le virtual machines sono disponibili per ogni tipo di sistema operativo e per un numero molto elevato di hardware differenti, il bytecode è di fatto il primo esempio di portabilità reale e totale. Java è un linguaggio a oggetti puro, dato che non è possibile programmare in modo non object oriented in parte o del tutto, come invece accade ad esempio con il C++. La gestione della memoria, per motivi di sicurezza e di semplicità di programmazione, viene gestita dalla VM per mezzo di un efficiente Garbage Collector. Questo è forse uno degli aspetti più delicati di tutto il mondo Java e proprio le continue ricerche ed i progressi ottenuti hanno permesso alla piattaforma Java di diventare sempre più stabile e performante. Una delle caratteristiche fondamentali di Java è che esso mette a disposizione un meccanismo di multithreading, col quale è possibile, all'interno della stessa applicazione, eseguire contemporaneamente più task. La VM inoltre implementa un sistema automatico di loading delle classi in grado di caricarle in memoria, leggendole da disco o nel caso di applicazioni Internet scaricandole dalla rete, solo al momento dell'effettiva necessità, in maniera molto simile a quanto avviene con le DLL del sistema Windows. Il linguaggio Java infine è stato progettato con il preciso obiettivo di offrire un elevato livello di sicurezza e semplicità d'implementazione; anche il debug di una applicazione richiede mediamente sforzi minori rispetto all'utilizzo di altre tecnologie.

2.2 Database

Cos'è un database

Una base di dati (database) può essere definita come un insieme di dati strettamente correlati, memorizzati su un supporto di memoria di massa, costituenti un tutt'uno, che possono essere manipolati, da più programmi applicativi; oppure possiamo dire che è un sistema di gestione di dati integrati, ricompilati e immagazzinati secondo precisi criteri, necessari all'attività che si deve svolgere. Un database è quindi una sorta di versione elettronica di un archivio, un luogo dedicato all'archiviazione, all'organizzazione e alla consultazione delle informazioni. I database archiviano le informazioni in campi, record e tabelle. Un campo è una singola unità di informazione, un record è una serie completa di campi, una tabella è invece una raccolta di record. Per usare le informazioni contenute in un database, occorre un Database Management System (DBMS). Si tratta di una serie di programmi software che consentono di immettere, selezionare e organizzare le informazioni archiviate nel database. I DBMS più diffusi sono i Relational Database Management System (RDBMS) [18] . Un RDBMS archivia le informazioni in tabelle di righe e colonne. Chi utilizza regolarmente i fogli di calcolo ha già una certa familiarità con l'archiviazione dei dati in tabelle. Ciascuna colonna di una tabella di database contiene un tipo differente di attributo, mentre ciascuna riga corrisponde a un singolo record. Per esempio, in una tabella dedicata ai clienti, le colonne potrebbero includere nome, indirizzo, numero di telefono e informazioni sull'account. Ciascuna riga rappresenta invece un cliente diverso.

Tipi di database

La forma più semplice di database è costituita da un file contenente tante righe quante sono le informazioni che devono essere memorizzate nel database. Questo tipo di archiviazione dei dati ha notevoli limitazioni: tempi di ricerca elevati, problemi di ridon-

danza delle informazioni memorizzate, problemi nella manutenzione delle informazioni. Il vantaggio è che si basa su una struttura semplicissima e relativamente facile da gestire. Questo tipo di database viene utilizzato generalmente quando la quantità di informazioni da memorizzare è molto piccola. Da questo tipo di database si sono sviluppati i database gerarchici, che utilizzano più file per memorizzare le informazioni, introducendo una relazione gerarchica di tipo padre-figlio (ad albero) fra gli stessi. Un database di questo tipo è IMS (Information Management System) di IBM (spesso è riferito con il nome del linguaggio da questo utilizzato, DL/I - Data Language I). Oggigiorno questo tipo di database non viene più utilizzato. I database reticolari (network) espandono il concetto di relazione gerarchica, rendendo possibile il raggiungimento di un nodo da più percorsi. Tale tipologia di database viene anche riferita con il nome CODASYL DBTG (Conference on Data System Languages, Data Base Task Group). Nonostante il fatto che sia possibile definire relazioni multiple tra le varie informazioni, la gestione di tali tipi di database diviene sempre più crescente all'aumentare del numero di relazioni, tant'è che oggigiorno questo tipo di database non viene più utilizzato. Dal modello a rete è nato il database relazionale o RDB (Relational DataBase). In questo tipo di database, la struttura logica è indipendente da quella fisica e fornisce sia elevate prestazioni che flessibilità di gestione, sia dei dati che della struttura del database stesso. Il database a oggetti (object-oriented) risolve alcune limitazioni del modello relazionale, in particolare la scarsa capacità di gestione dei BLOB (Binary Large OBject) cioè dei tipi di dati complessi come le immagini, documenti. Questo è dovuto al fatto che i database sono nati per gestire dati (sequenze di 0 e di 1) con dimensioni relativamente piccole. I BLOB sono oggetti che possono avere dimensioni decisamente più grosse rispetto a quelle per le quali sono nati i database. I BLOB vengono generalmente memorizzati all'esterno del DB e sul DB viene memorizzato soltanto un riferimento ad essi (ad esempio il loro path sul filesystem). I database ad oggetti gestiscono nativamente i BLOB, ma ogni database utilizza un proprio modo di gestione dei BLOB. Le informazioni vengono memorizzate

in oggetti che hanno relazioni di tipo gerarchico. Questi tipi di database non sono molto diffusi: vengono utilizzati soltanto in ambienti CAD ed engineering. Oggi i principali database in circolazione sono di tipo relazionale, ciò perché praticamente tutti gli insiemi di dati che corrispondono a entità complesse organizzate come imprese, scuole, associazioni varie, etc, implicano collegamenti tra i vari dati ad esempio: ai fornitori sono collegate le merci, agli alunni i corsi, e così via. La norma fondamentale per stabilire relazioni tra tabelle, cioè tra contenitori di dati correlabili, è che il campo di collegamento non deve avere ripetizioni, ossia ogni record deve potere essere identificato in maniera univoca. Il campo che permette l'identificazione di ogni record è detto chiave primaria e deve essere comune alle tabelle che si intende correlare.
Si possono stabilire tre tipi di relazione:

- uno a uno: si tratta di relazioni tra elementi che hanno una corrispondenza univoca: ad un elemento di una tabella ne corrisponde uno soltanto in un'altra e viceversa;
- uno a molti: sono relazioni che si stabiliscono tra un record di una tabella e più records di un'altra tabella, ma non il contrario;
- molti a molti: un record può essere relazionato a più di un record di un'altra tabella e viceversa; questo tipo di relazione è normalmente definita tramite una terza tabella che costituisce un ponte tra le due da relazionare.

Modello a oggetti vs modello relazionale

I modelli a oggetti costituiscono una promettente evoluzione delle basi di dati. I sistemi a oggetti integrano la tecnologia della base di dati con il paradigma ad oggetti sviluppato nell'ambito dei linguaggi di programmazione. Nelle basi di dati a oggetti, ogni entità del mondo reale è rappresentata da un oggetto. Per esempio: dati multimediali, cartine geografiche, ecc. Da notare che è difficile pensare ad una struttura relazionale per queste entità. I programmatori che utilizzano il paradigma OOP e hanno necessità di

salvare i propri oggetti, possono scegliere di salvarli su un database relazionale; in questo caso però bisogna convertire gli oggetti in tabelle con righe e colonne, nel fare questo bisogna mantenere anche la descrizione e le relazioni delle varie classi nonchè lo schema relazionale e il mapping delle classi nel db relazionale. Se al contrario scegliamo una base di dati a oggetti, i programmatori possono salvare gli oggetti cosi come sono e dovranno solamente preoccuparsi di modellare una base di dati a oggetti che descriva correttamente la realtà fisica che deve immagazzinare nel database. La differenza tra questi due modelli sta nella mancanza di interoperabilità. Ciò significa che si può accedere ad un ODBMS solo per mezzo del DBMS che lo gestisce. Per esempio, ad un database Goods si può accedere solo per mezzo di un programma scritto per Goods e non qualcos'altro.

2.3 MySQL

Database MySQL

MySql [16] è il database relazionale open source più diffuso al mondo. Originariamente sviluppato dalla società svedese TcX per uso interno, MySql costituisce oggi la soluzione ottimale (soprattutto in ambiente Linux) per chi è in cerca di un database veloce, flessibile, affidabile e soprattutto gratuito. MySQL è un RDBMS, ossia un sistema di gestione per database relazionali. MySQL si occupa della strutturazione e della gestione a basso livello dei dati stessi, in modo da velocizzarne l'accesso, la modifica e l'inserimento di nuovi elementi. L'acronimo RDBMS significa Relational DataBase Management System e sta ad indicare che MySQL offre la possibilità di conservare i dati non in un enorme storeroom ma in diverse tabelle, in modo di velocizzarne l'accesso. L'acronimo SQL [30] significa Structured Query Language ed indica il linguaggio standard di interrogazione dei DataBase.

MySQL è diventato uno dei sistemi per la gestione di database più importanti al mondo. Dai piccoli progetti di sviluppo ad alcuni tra i siti più famosi e più prestigiosi

sul Web, MySQL si è dimostrato una soluzione solida e veloce per tutti i tipi di necessità di archiviazione di dati. MySQL è in grado di gestire database in diversi formati. I principali sono:

- **myISAM**, implementazione dei B-alberi, è il formato nativo ed include molti tool di ottimizzazione e backup;
- **innoDB**, supporta le transazioni e le foreign keys;
- **BerkeleyDB**, con supporto per le transazioni ma poco flessibili e con basse prestazioni.

2.4 XML

Il formato xml

XML *(eXtensible Markup Language)* [19] è un "linguaggio a marcatori" *(tag language)* estremamente flessibile, derivato da SGML *(Standard Generalized Markup Language)* e nato per gestire la pubblicazione su larga scala di documenti elettronici tramite il Web. In seguito, XML si è rivelato adatto a rappresentare dati in contesti estremamente eterogenei.

Nel 1996 è stato proposto XML, un linguaggio che consentiva di realizzare browser completamente estensibili grazie alla possibilità di definire attraverso il linguaggio e "dentro" i documenti la tipologia dei tag ammessi e la struttura del linguaggio di marcatura. Lo standard di riferimento era SGML, ma occorreva considerare anche esigenze nate col Web e con le sue tecnologie. Uno degli obiettivi era quello di poter includere nel documento una semplice specifica della sua struttura e del suo significato, in modo da renderlo "autocontenuto" dal punto di vista della possibilità di interpretarlo. Per fare ciò si è introdotto il DTD (Document Type Definition), una specifica sintetica della struttura di una classe di documenti. Un documento XML è intrinsecamente caratterizzato da

una struttura gerarchica. Esso è composto da componenti denominati elementi. Ciascun elemento rappresenta un componente logico del documento e può contenere altri elementi (sottoelementi) o del testo. Gli elementi possono avere associate altre informazioni che ne descrivono le proprietà. Queste informazioni sono chiamate **attributi**.

L'organizzazione degli elementi segue un ordine gerarchico o arboreo che prevede un elemento principale, chiamato **root element** o semplicemente root o radice. La radice contiene l'insieme degli altri elementi del documento. Possiamo rappresentare graficamente la struttura di un documento XML tramite un albero, generalmente noto come document tree. La struttura logica di un documento XML viene tradotta in una corrispondente struttura fisica composta di elementi sintattici chiamati **tag**. Questa struttura fisica viene implementata tramite un file di testo creato con un qualsiasi editor. I tag sono distinti dal testo libero (o PCDATA, acronimo di Parsable Character DATA) per il fatto di iniziare e finire con le parentesi angolari '$\langle\ e\ \rangle$' .
Un esempio potrebbe essere rappresentato da una lettera che ha un mittente, una data, un destinatario, un oggetto, una forma cortese di saluto, un corpo, una chiusura ed una firma.
Il corpo della lettera ha almeno un paragrafo.

```
<?xml version="1.0" encoding='utf-8'?>
<!DOCTYPE lettera SYSTEM "lettera.dtd">
<lettera>
        <contatto_da> ... </contatto_da>
        <data> .... </data>
        <contatto_a> ... </contatto_a>
        <oggetto> ... </oggetto>
        <saluto> ... </saluto>
        <corpo>
                <paragrafo> ... </paragrafo>
                <paragrafo> ... </paragrafo>
                <paragrafo> ... </paragrafo>
        </corpo>
        <chiusura> ... </chiusura>
        <firma> ... </firma>
</lettera>
```

Figura 2.1: eXtensible Markup Language di una lettera

DTD

Lo scopo di un Document Type Definition (definizione del tipo di documento) [21] è quello di definire le componenti ammesse nella costruzione di un documento XML. Il termine non è utilizzato soltanto per i documenti XML ma anche per tutti i documenti derivati dall'SGML (di cui peraltro XML vuole essere una semplificazione che ne mantiene la potenza riducendone la complessità) tra cui famosissimo è l'HTML. In SGML, un DTD è necessario per la validazione del documento. Anche in XML, un documento è valido se presenta un DTD ed è possibile validarlo usando il DTD.
Tuttavia XML permette anche documenti ben formati, ovvero documenti che, pur essendo privi di DTD, presentano una struttura sufficientemente regolare e comprensibile da poter essere controllata. Una caratteristica fondamentale dell'XML è l'estensibilità. L'autore di un documento XML può creare nuovi tag per descrivere i contenuti semantici dei propri dati, semplificando il loro scambio fra i gruppi di persone interessate allo stesso

settore. Ciò ha portato alla necessità di definire delle regole grammaticali, o vincoli, alle quali gli elementi devono attenersi.
Queste regole grammaticali sono definite nelle specifiche XML e sono codificate nel DTD. Le regole grammaticali o vincoli specificano:

- qual è l'insieme degli elementi e degli attributi che si possono usare nel documento XML;
- quali sono le relazioni gerarchiche fra gli elementi;
- qual è l'ordine in cui gli elementi appariranno nel documento XML;
- quali elementi ed attributi sono opzionali;

Quando un documento XML è ben formato e rispetta le regole del DTD a cui si riferisce si dice che è un documento XML valido. In un documento XML si può specificare il DTD in modo esplicito (DTD interno) o con un riferimento ad un documento distinto (DTD esterno).

Un esempio di documento DTD relativo all'XML sopra descritto è il seguente:

```
<?xml version="1.0" encoding='utf-8'?>
<!ELEMENT lettera(contatto_da, data, contatto_a, oggetto, saluto, corpo,
closing, firma)>
<!ELEMENT contatto_da (#PCDATA) >
<!ELEMENT data (#PCDATA) >
<!ELEMENT contatto_a (#PCDATA) >
<!ELEMENT oggetto (#PCDATA) >
<!ELEMENT saluto (#PCDATA) >
<!ELEMENT corpo (paragrafo+) >
<!ELEMENT paragraph (#PCDATA) >
<!ELEMENT chiusura (#PCDATA) >
<!ELEMENT firma (#PCDATA) >
```

Figura 2.2: Document Type Definition di una lettera

Capitolo 3

Analisi UML

In ingegneria del software, UML (Unified Modeling Language, linguaggio di modellazione unificato) [22] è un linguaggio di modellazione e di specifica basato sul paradigma object-oriented. Il nucleo del linguaggio fu definito nel 1996 da Grady Booch, Jim Rumbaugh e Ivar Jacobson (detti i tre amigos) sotto l'egida dello OMG, che tuttora gestisce lo standard di UML. Il linguaggio nacque con l'intento di unificare approcci precedenti (dovuti ai tre padri di UML e altri), raccogliendo le best practices nel settore e definendo così uno standard industriale unificato. UML svolge un'importantissima funzione di lingua franca nella comunità della progettazione e programmazione a oggetti. Gran parte della letteratura di settore usa UML per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile a un vasto pubblico. Di per sé, UML è solo un linguaggio di modellazione, e non definisce alcuna specifica metodologia per la creazione di modelli (o alcun processo software). UML può quindi essere utilizzato nel contesto di diversi approcci metodologici. La OMG gestisce uno standard metodologico, correlato a UML ma proposto come specifica indipendente, detto RUP. UML consente di costruire modelli object-oriented per rappresentare domini di diverso genere. Nel contesto dell'ingegneria del software, viene usato soprattutto per descrivere il dominio applicativo di un sistema software e/o il comportamento e la struttura del sistema stesso. Il modello

è strutturato secondo un insieme di viste che rappresentano diversi aspetti della cosa modellata (funzionamento, struttura, comportamento, e così via), sia a scopo di analisi che di progetto, mantenendo la tracciabilità dei concetti impiegati nelle diverse viste. Oltre che per la modellazione di sistemi software, UML viene non di rado impiegato per descrivere domini di altri tipi, come sistemi hardware, strutture organizzative aziendali, processi di business. Lo standard UML, gestito da OMG, definisce una sintassi e delle regole di interpretazione; non si tratta quindi di una metodologia di progettazione e per questo motivo può essere adottato con diverse metodologie o in ambiti diversi da quello informatico.

3.1 Use Case Diagram

I diagrammi di caso d'uso [23] descrivono le relazioni e le dipendenze tra un gruppo di Casi d'uso e gli attori partecipanti al processo. È importante notare che i diagrammi di caso d'uso non sono adatti a rappresentare la progettazione, e non possono descrivere le parti interne di un sistema. I diagrammi di caso d'uso servono a facilitare la comunicazione con gli utenti futuri del sistema e con il cliente, e sono particolarmente utili a determinare le funzionalità necessarie che il sistema deve avere. I diagrammi di caso d'uso dicono cosa deve fare il sistema, ma non specificano come si deve fare.

Caso d'uso

Un Caso d'uso descrive, dal punto di vista degli attori, un gruppo di attività in un sistema che produce un risultato concreto e tangibile. I casi d'uso sono descrizioni delle interazioni tipiche tra gli utenti di un sistema e il sistema stesso. Rappresentano l'interfaccia esterna del sistema, e specificano un modulo di requisiti di cosa il sistema deve fare (ricorda, solo cosa, non come). Lavorando con i casi d'uso, è importante ricordare alcune semplici regole:

- Ogni caso d'uso è relativo ad almeno un attore;
- Ogni caso d'uso ha un iniziatore (cioè un attore);
- Ogni caso d'uso porta a un risultato rilevante (un risultato con valore di business).

I casi d'uso possono anche avere relazioni con altri casi d'uso. I tre tipi più comuni di relazioni tra casi d'uso sono:

- "inclusione" che specifica che un caso d'uso avviene all'interno di un altro caso d'uso;
- "estensione" che specifica che in certe situazioni, o a un certo punto (chiamato punto di estensione) un caso d'uso sarà esteso da un altro;
- la generalizzazione specifica che un caso d'uso eredita le caratteristiche del sovra-caso d'uso, e può sostituirne alcune o aggiungerne nuove in un modo simile all'ereditarietà tra classi.

Attore

Un attore è un'entità esterna (fuori dal sistema) che interagisce con il sistema partecipando a (e spesso iniziando) un caso d'uso. Gli attori possono essere persone reali (per esempio utenti del sistema), altri sistemi o eventi esterni. Gli attori non rappresentano le persone fisiche o i sistemi, ma il loro ruolo. Ciò significa che quando una persona interagisce con il sistema in modi diversi (assumendo ruoli diversi) sarà rappresentata da diversi attori. Per esempio una persona che fornisce supporto per telefono e immette ordini dal cliente nel sistema sarebbe rappresentato da un attore ≪Squadra di supporto≫ e un attore ≪Rappresentante di vendita≫ .
Di seguito vi è riportato il diagramma dei casi d'uso che evidenzia le funzionalità del progetto realizzato

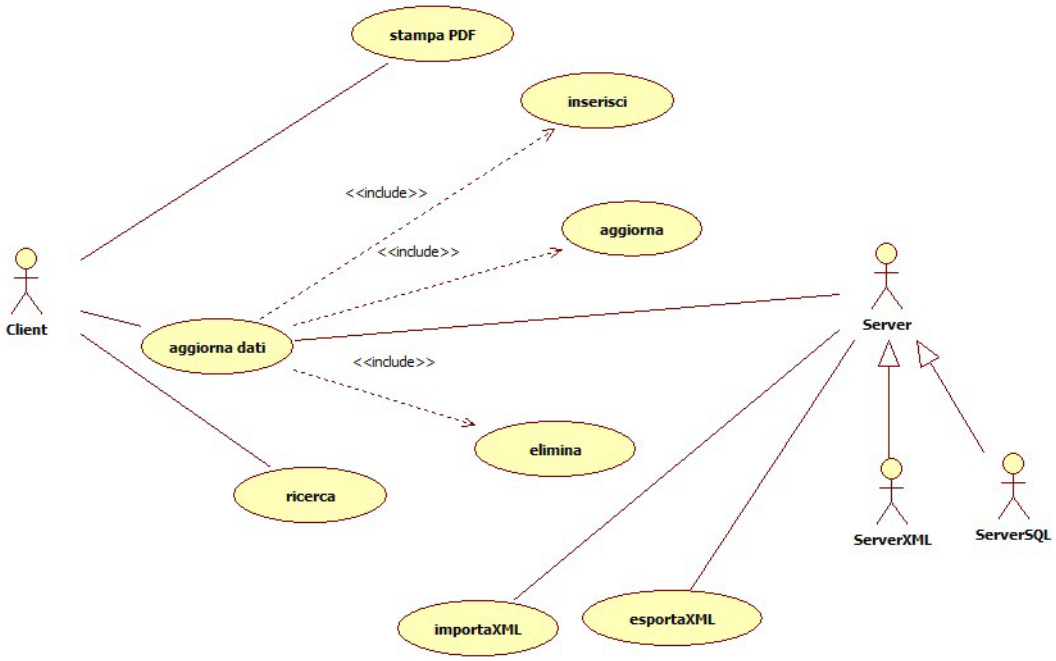

Figura 3.1: Diagramma dei casi d'uso

3.2 Class Diagram

I diagrammi delle classi (class diagram) [24] sono uno dei tipi di diagrammi che possono comparire in un modello UML. In termini generali, consentono di descrivere tipi di entità, con le loro caratteristiche, e le eventuali relazioni fra questi tipi. Gli strumenti concettuali utilizzati sono il concetto di classe del paradigma object-oriented e altri correlati.

Classe

L'elemento di modello principale dei diagrammi delle classi è la classe. Una classe rappresenta una categoria di entità (istanze), nel caso particolare dette oggetti; il nome della classe indica la categoria di entità descritta dalla classe. Ogni classe è corredata da un insieme di attributi (che descrivono le caratteristiche degli oggetti della classe) e operazioni (che descrivono il comportamento della classe). Il simbolo grafico che rappresenta le classi UML è un rettangolo suddiviso in tre scomparti, rispettivamente dedicati al nome della classe, agli attributi e alle operazioni.

Associazione

Due classi possono essere legate da associazioni che rappresentano i legami (link) che possono sussistere fra gli oggetti delle classi associate. Ogni categoria di associazione (aggregazione, composizione, dipendenza, generalizzazione, ecc.) viene rappresentata mediante una particolare freccia che connette le due classi coinvolte. Tali associazioni possono essere corredate da un insieme di informazioni aggiuntive, per esempio relative alla molteplicità (il numero di oggetti delle due classi associate che possono essere coinvolti in un link).

Dipendenza

Due classi possono essere legate da una relazione di dipendenza, che indica che la definizione di una delle due fa riferimento alla definizione dell'altra.

Generalizzazione

Due classi possono essere legate da una relazione di generalizzazione, che indica che una delle due classi (detta superclasse) si può considerare una generalizzazione dell'altra (detta sottoclasse).

Di seguito vi è riportato il diagramma delle classi che mostra la struttura del progetto realizzato.

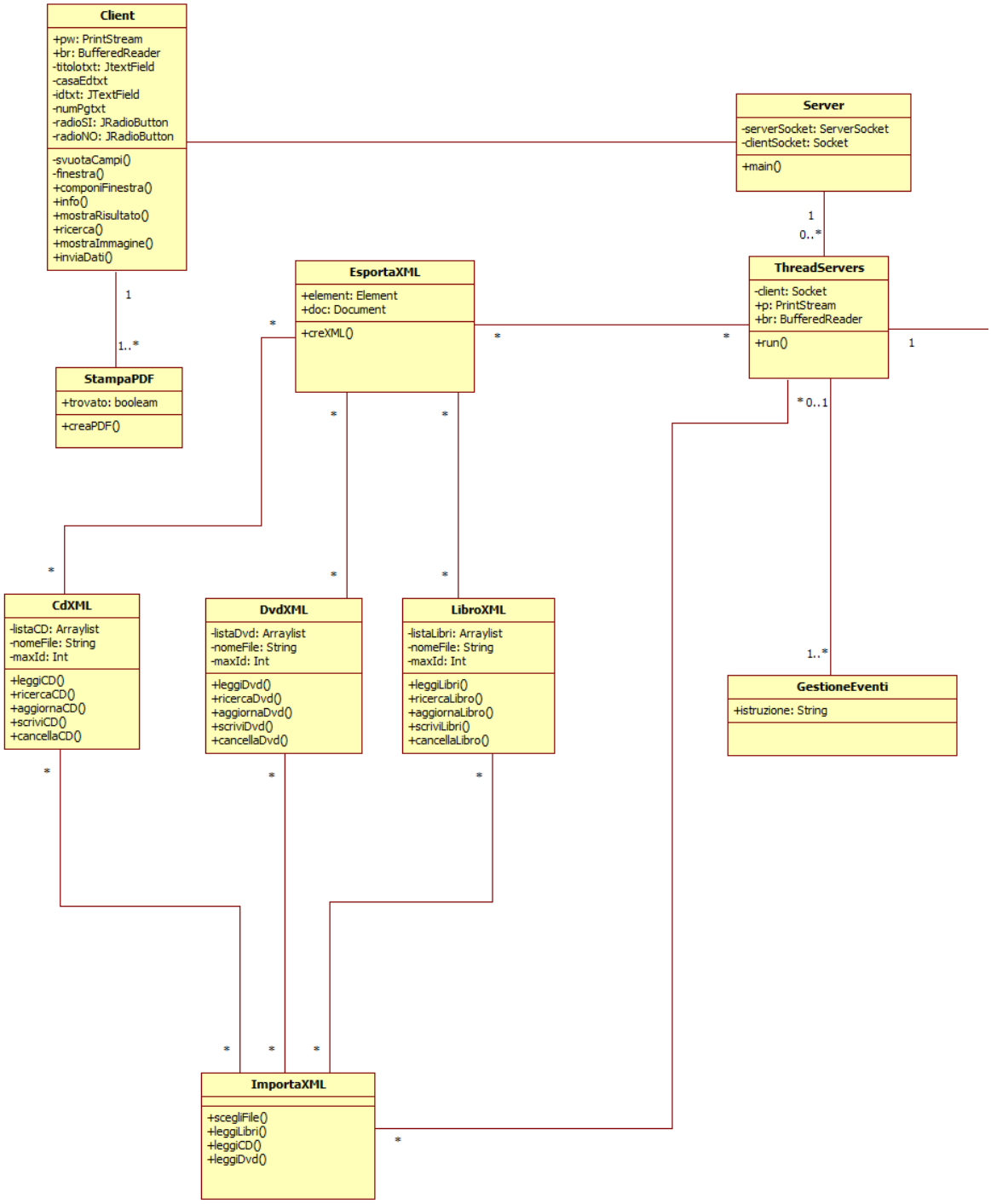

Figura 3.2: I parte: diagramma delle classi

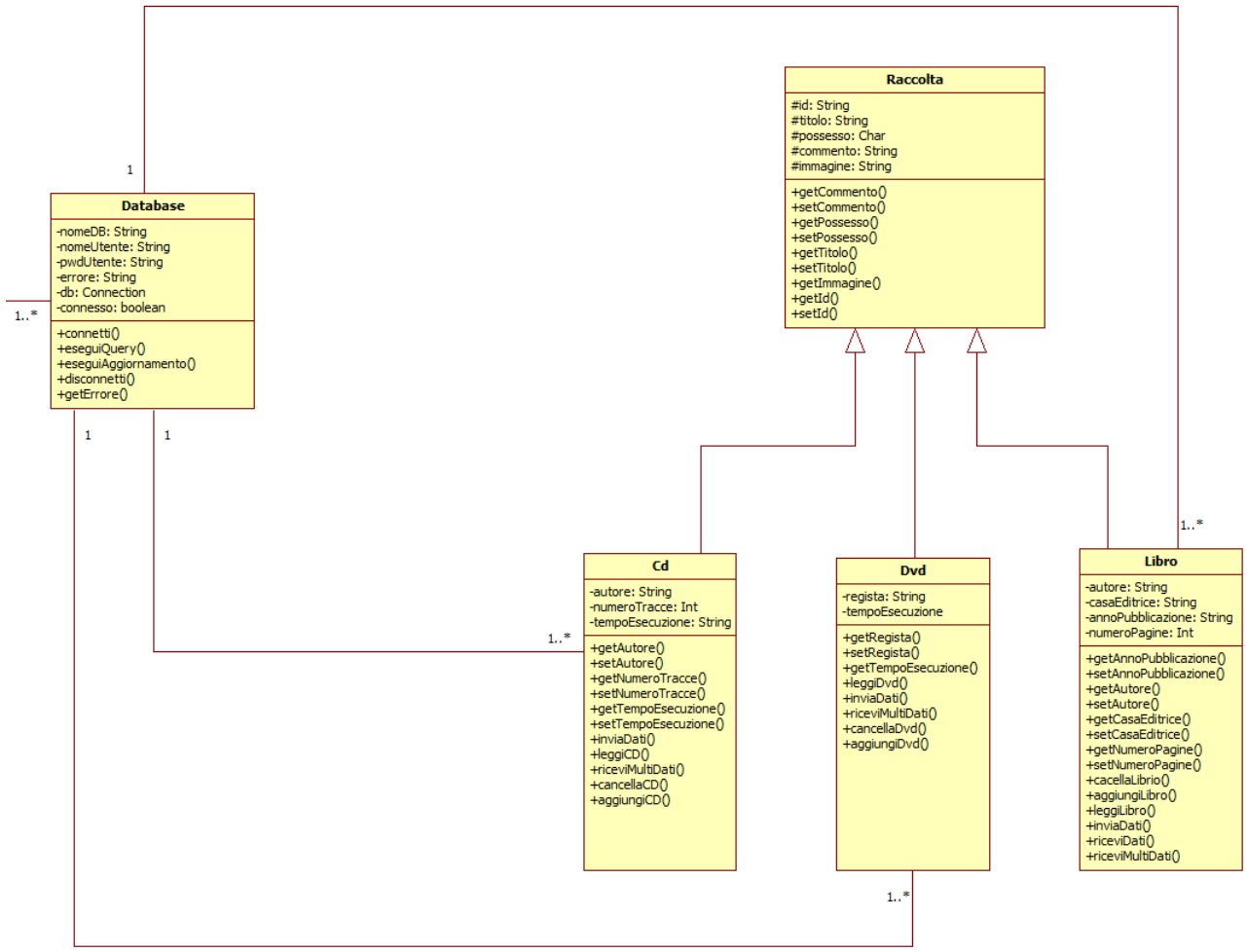

Figura 3.3: II parte: diagramma delle classi

3.3 Object Diagram

I diagrammi degli oggetti [25] rappresentano una variante dei diagrammi delle classi, tanto che anche la notazione utilizzata è pressoché equivalente con le sole differenze che i nomi degli oggetti vengono sottolineati e le relazioni vengono dettagliate.
Questa tipologia di diagramma si occupa della proiezione statica del sistema e mostra un ipotetico esempio di un diagramma delle classi. Si tratta della famosa diapositiva "scattata" a un istante di tempo preciso, riportante un ipotetico stato di esecuzione evidenziato dagli oggetti presenti in memoria e dal relativo stato. Pertanto mentre un diagramma delle classi è sempre valido, un diagramma degli oggetti rappresenta una possibile istantanea del sistema valida in un istante di tempo ben preciso.

Di seguito vi è mostrato il diagramma degli oggetti relativo ad una determinata situazione del progetto.

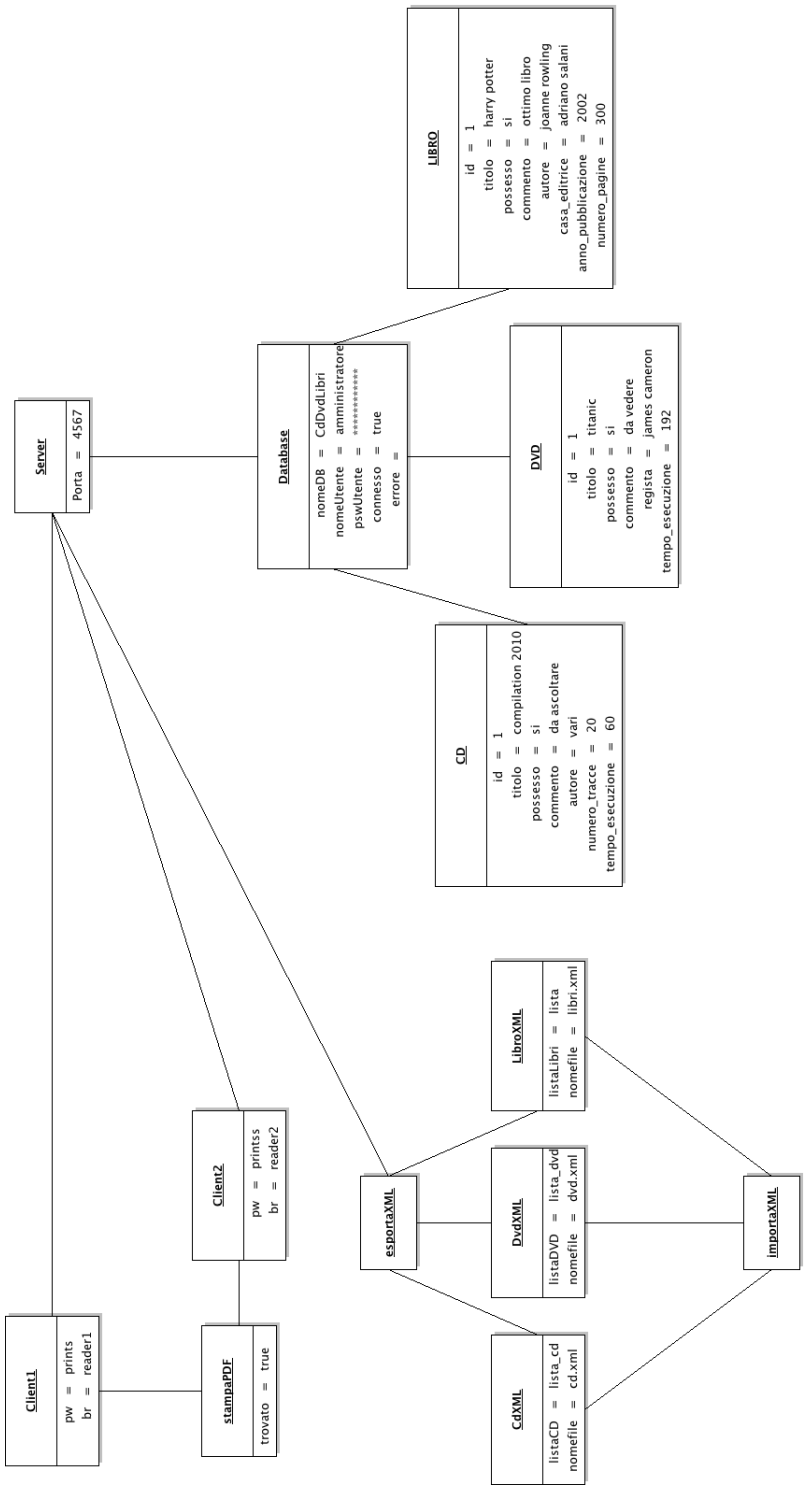

Figura 3.4: Diagramma degli oggetti

3.4 StateChart Diagram

Lo Statechart Diagram [26] è un diagramma previsto dall'UML per descrivere il comportamento di entità o di classi in termini di stato (macchina a stati).
Il diagramma mostra gli stati che sono assunti dall'entità o dalla classe in risposta ad eventi esterni. Il concetto di stato è spesso posto in relazione a ciclo di vita; l'insieme completo di stati che un'entità o una classe può assumere, dallo stato iniziale a quello finale, ne rappresenta il ciclo di vita. Gli elementi rappresentati da uno State Chart Diagram sono lo stato - distinguendo tra iniziale, intermedi e stato finale - l'evento, l'azione e la guardia.

Stato

Lo stato descrive una qualità dell'entità o classe che si sta rappresentando (pratica aperta, in lavorazione, sospesa, chiusa); l'evento è la descrizione dell'azione che comporta il cambiamento di stato, l'azione è l'evento che ne consegue, la guardia è l'eventuale condizione che si deve verificare perché si possa compiere l'azione.

Evento

Un evento è l'occorrenza di uno stimolo che può innescare una transizione fra stati La condizione o guardia è una espressione logica da valutare. Se da uno stato escono più transizioni, le loro condizioni devono essere mutuamente esclusive.
Di seguito vi è riportato il diagramma di stato che mostra i possibili stati che può assumere un articolo.

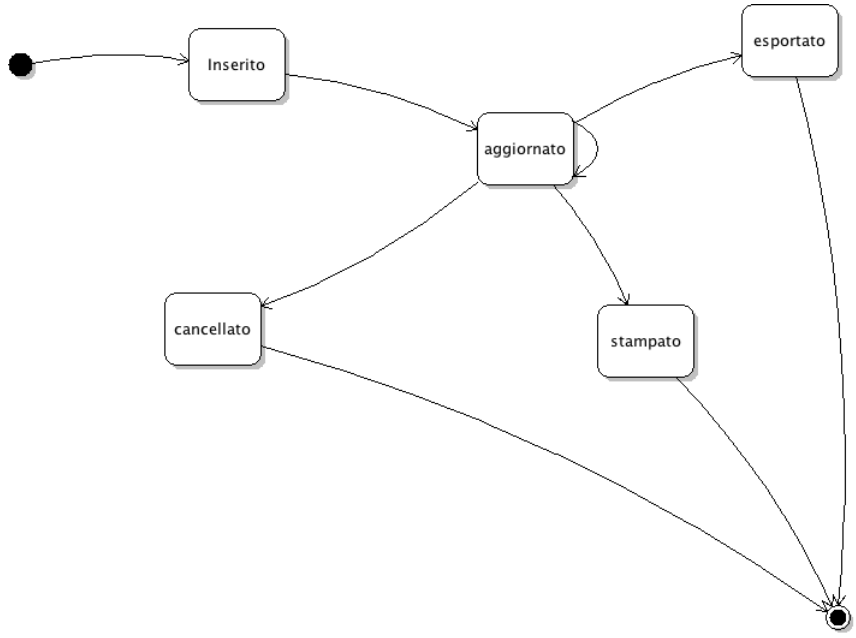

Figura 3.5: Diagramma di stato

3.5 Activity Diagram

L'Activity Diagram [27] è un diagramma definito all'interno dello Unified Modeling Language (UML) e definisce le attività da svolgere per realizzare una data funzionalità. Può essere utilizzato anche in fase di design per dettagliare un determinato algoritmo. In teoria ad ogni Use Case Diagram dovrebbe corrispondere un Activity Diagram. Più in dettaglio, un Activity Diagram definisce una serie di attività o flusso, anche in termini di relazioni tra di esse, chi è responsabile per la singola attività ed i punti di decisione.[27]

Activity

L'Activity rappresenta una specifica attività che deve essere svolta all'interno della funzione. È rappresentata da un rettangolo smussato con una descrizione dell'attività.

Flusso

Il flusso è rappresentato tramite delle frecce orientate, che indicano la sequenza temporale con cui devono essere effettuate le diverse attività. È previsto un simbolo per indicare l'inizio del flusso ed un altro per indicarne il termine. Le attività possono essere anche rese in parallelo, in questo caso il punto di divisione (fork) è rappresentato da frecce divergenti rispetto al segmento.
Nel caso le attività siano alternative, cioè svolte o meno rispetto ad una scelta, il punto di decisione è rappresentato da dei rombi da cui partono i flussi alternativi. Il punto di ricongiungimento (join) è reso tramite un segmento su cui le frecce si ricongiungono.

Di seguito vi è riportato il diagramma delle attività della ricerca di un articolo.

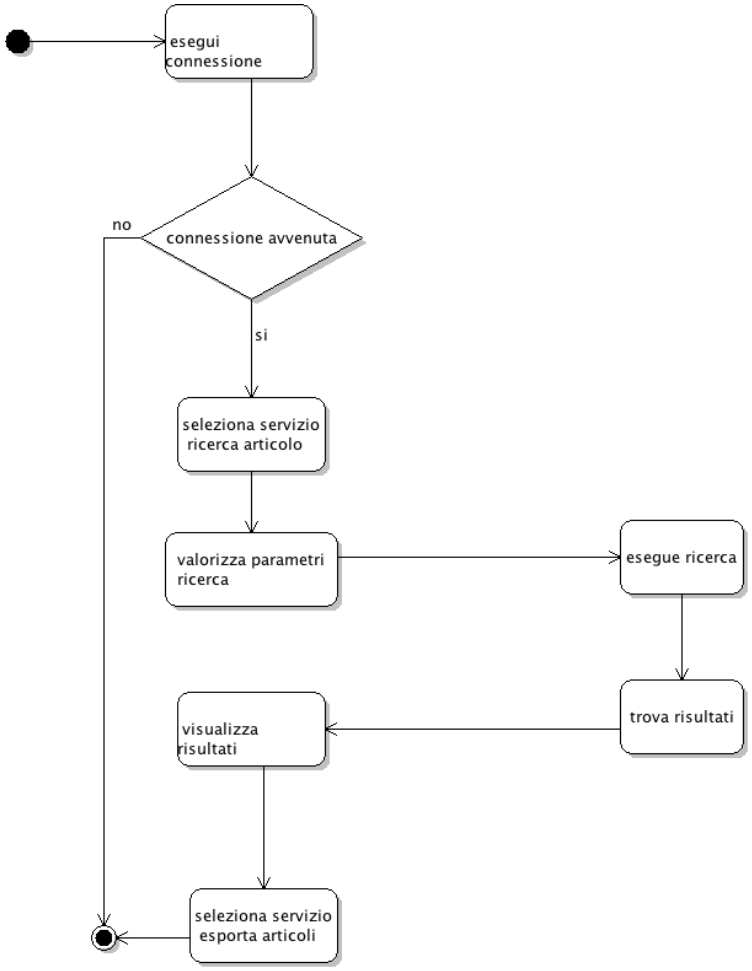

Figura 3.6: Diagramma di attivita'

3.6 Sequence Diagram

Un Sequence Diagram [28] è un diagramma previsto dall'UML utilizzato per descrivere uno scenario. Uno scenario è una determinata sequenza di azioni in cui tutte le scelte sono state già effettuate; in pratica nel diagramma non compaiono scelte, né flussi alternativi. Il Sequence Diagram descrive le relazioni che intercorrono, in termini di messaggi, tra Attori, Oggetti di business, Oggetti od Entità del sistema che si sta rappresentando.

Messaggio

Un messaggio è un'informazione che viene scambiata tra due entità. Solitamente chi invia il messaggio, la parte attiva, è l'attore. Il messaggio è sincrono, se l'emittente rimane in attesa di una risposta, o asincrono, nel caso l'emittente non aspetti la risposta e questa può arrivare in un secondo momento. Il messaggio che viene generato in risposta ad un precedente messaggio, al quale si riferisce anche come contenuto informativo, è detto messaggio di risposta. Un messaggio, in cui il ricevente è nello stesso tempo l'emittente, è detto ricorsivo.

Di seguito vi è riportato il diagramma di sequenza della ricerca di un articolo.

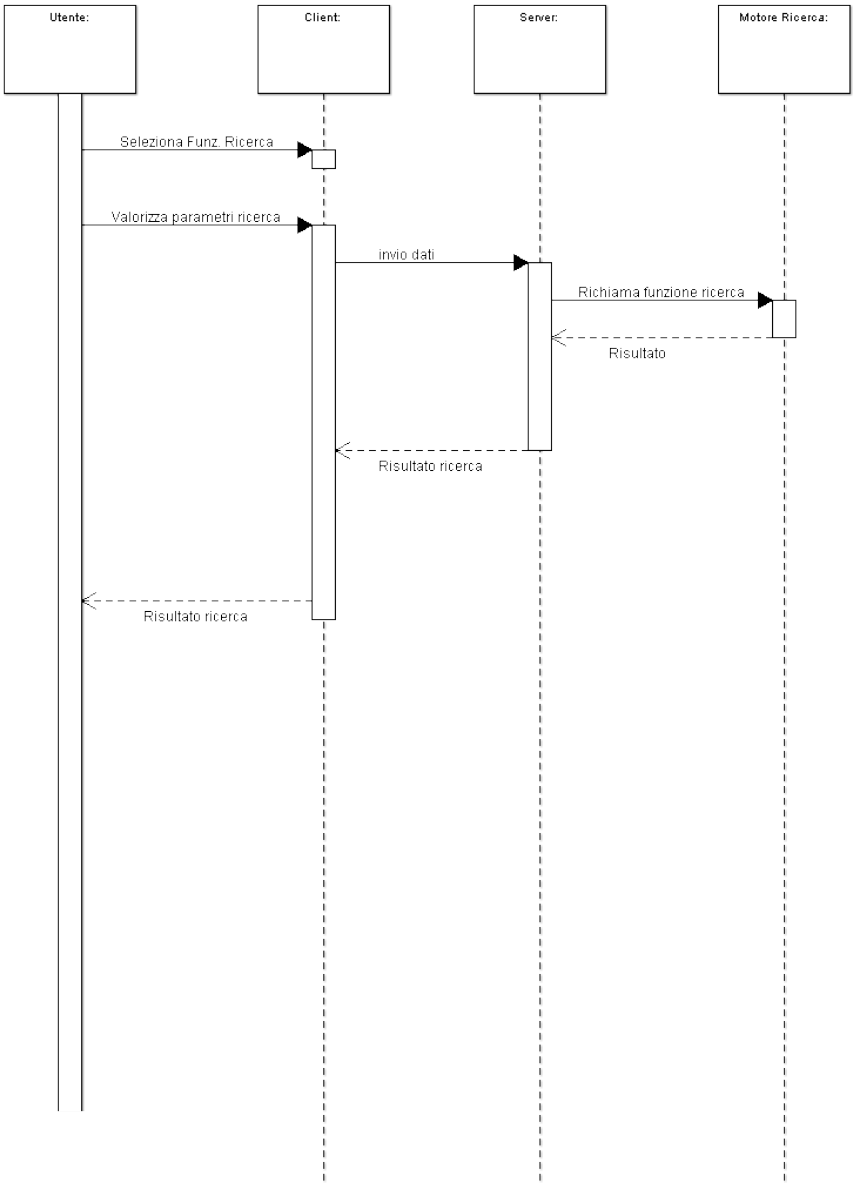

Figura 3.7: Diagramma di sequenza

3.7 Deployment Diagram

Nessun sistema software potrebbe mai funzionare correttamente se non si basasse su una solida struttura hardware. Per tale motivo, il Deployment Diagram [29] modella proprio questo aspetto del sistema. Il Deployment Diagram (diagramma di dispiegamento) è un diagramma di tipo statico previsto dal linguaggio di modellazione object-oriented UML per descrivere un sistema in termini di risorse hardware detti nodi, e di relazioni fra di esse. Spesso si utilizza un diagramma che mostra come le componenti software sono distribuite rispetto alle risorse hardware disponibili sul sistema. Gli elementi grafici UML del deployment diagram sono:

- Un cubo che rappresenta un nodo. Un nodo ha un suo nome ed è possibile anche usare uno stereotipo per indicare il tipo di risorsa che esso rappresenta. Se un nodo fa parte di un package, allora il nome del package deve precedere il nome del nodo.
- Una linea che unisce due cubi rappresenta una connessione tra i due nodi. E' possibile usare anche uno stereotipo anche per fornire informazioni sulla connessione.

Di seguito vi è riportato il diagramma di deployment dell'applicativo.

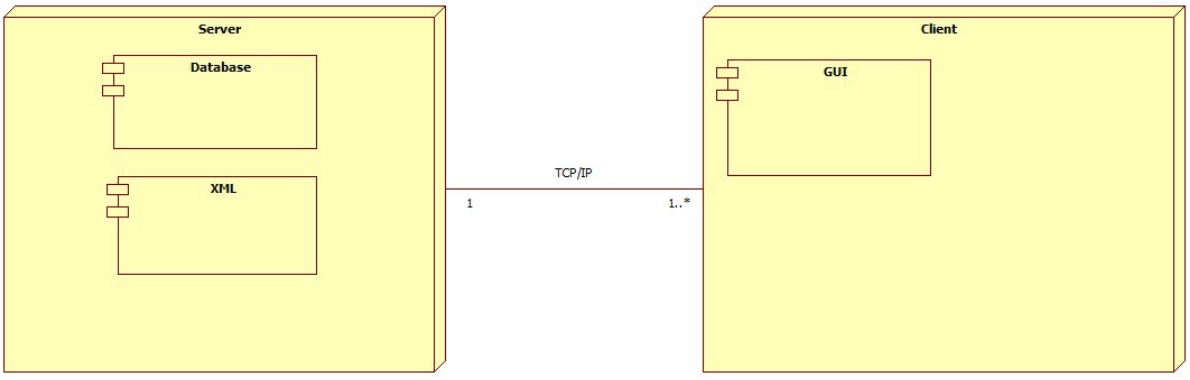

Figura 3.8: Diagramma di deployment

Capitolo 4

Implementazione dell'applicazione e tecnologie usate

Il sistema è nato per gestire un archivio dati di tipo relazionale che gestisce una banca dati (CD-DVD-LIBRI) attraverso l'inserimento/catalogazione delle informazioni presenti nei documenti da archiviare, es. titolo, autori, immagine copertina, numero tracce (CD-DVD), casa editrice, numero pagine, anno e commento. Dopo aver immesso i dati nel database è possibile ricercare il record attraverso l'inserimento di keyword nei campi noti nel record (pannello di ricerca).

All'avvio del Server sarà possibile scegliere le due modalità con cui si potrà avviare (XML o SQL)

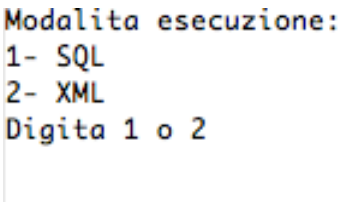

Figura 4.1: Scelta modalita' esecuzione

Se si desidera avviare il database in modalità SQL basta premere 1, altrimenti se si

desidera avviare il database in modalità XML allora basterà digitare il tasto 2 da tastiera. Una volta avviato il database nella modalità che si desidera, possono essere avviati uno o più client.

All'avvio il client avrà il seguente aspetto, le funzionalità disponibili saranno:

- Importa dati
- Inserimento
- Ricerca
- Info

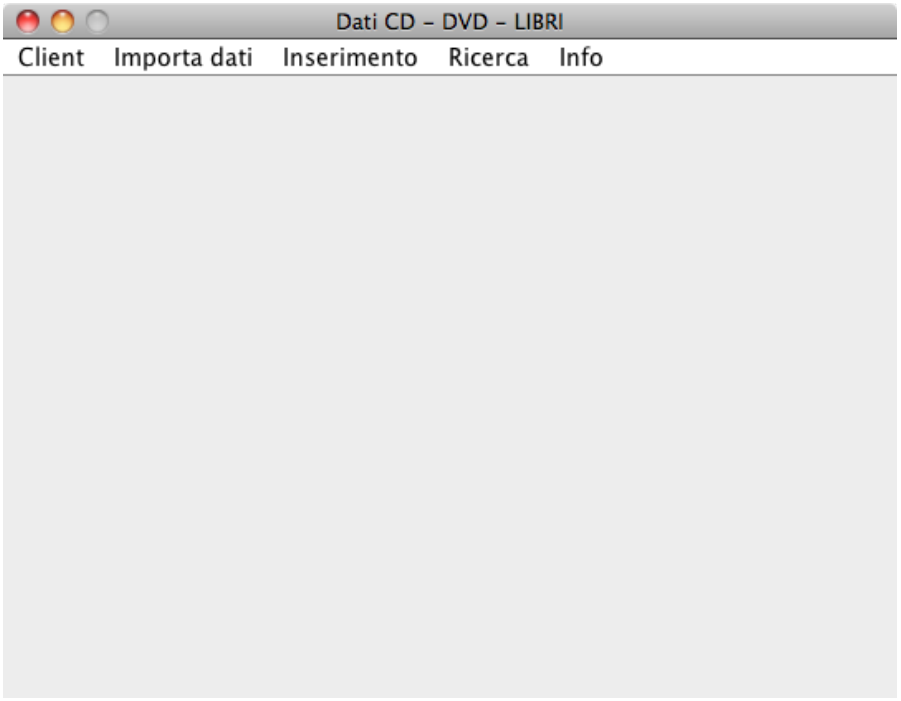

Figura 4.2: Schermata principale dell'applicativo

Utilizzando la voce di menu Importa dati e scegliendo i tipi di dati che si vogliono importare (CD, DVD, LIBRI) verrà aperta una finestra che consente di selezionare il file da importare ed il suo contenuto presente su file XML verrà caricato sulla relativa tabella del database.

Figura 4.3: Finestra che mostra il contenuto del database prima dell'importazione del file XML

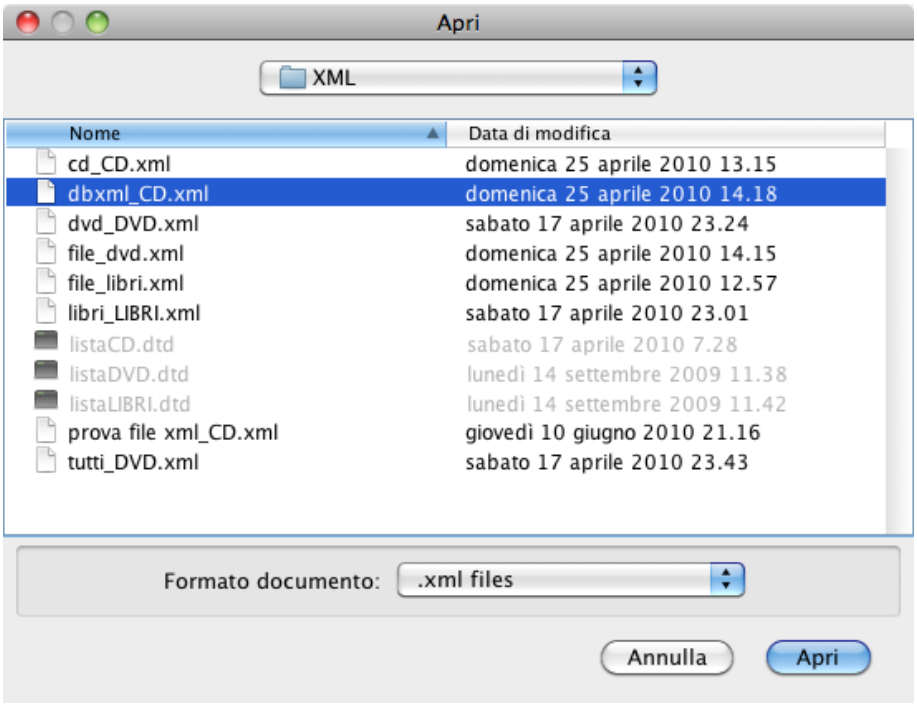

Figura 4.4: Finestra che mostra la scelta del file da importare

Figura 4.5: Finestra messaggio aggiornamento avvenuto

Figura 4.6: Finestra che mostra il contenuto del database dopo l'importazione del file XML

Utilizzando il menu Inserimento e selezionando la voce CD verranno mostrati i dati (caratteristiche) relativi al prodotto CD. Premendo il tasto Aggiorna, dopo aver inserito tutti i dati obbligatori (indicati con *) verrà eseguito inserimento/aggiornamento sulle tabelle del DB o sui file XML

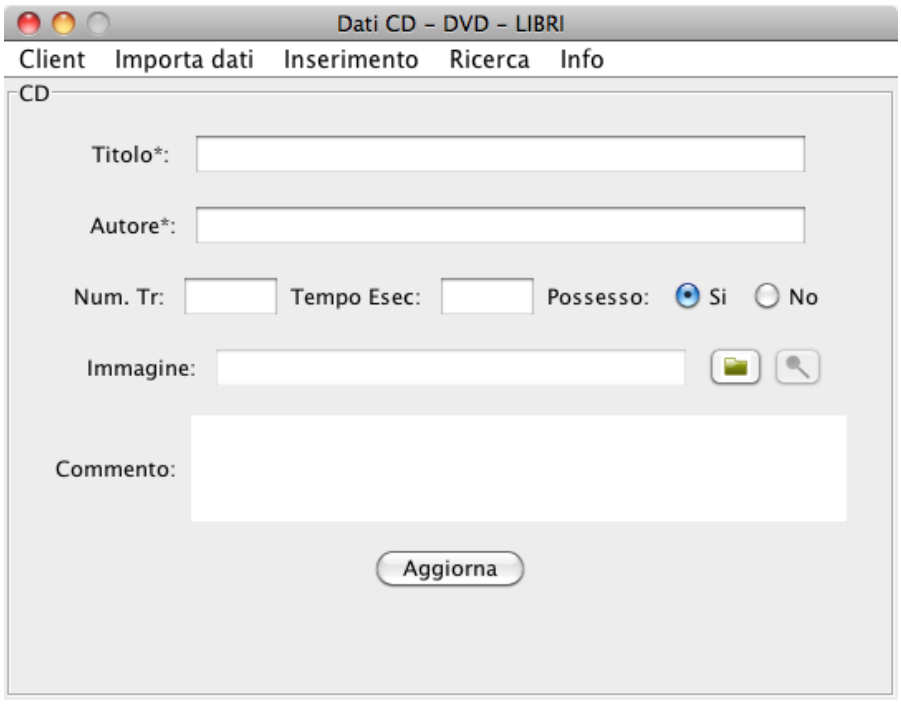

Figura 4.7: Finestra inserimento CD

Utilizzando il menu ricerca sarà possibile trovare i vari articoli presenti su file XML o sulle tabelle del database relativamente all'articolo selezionato.

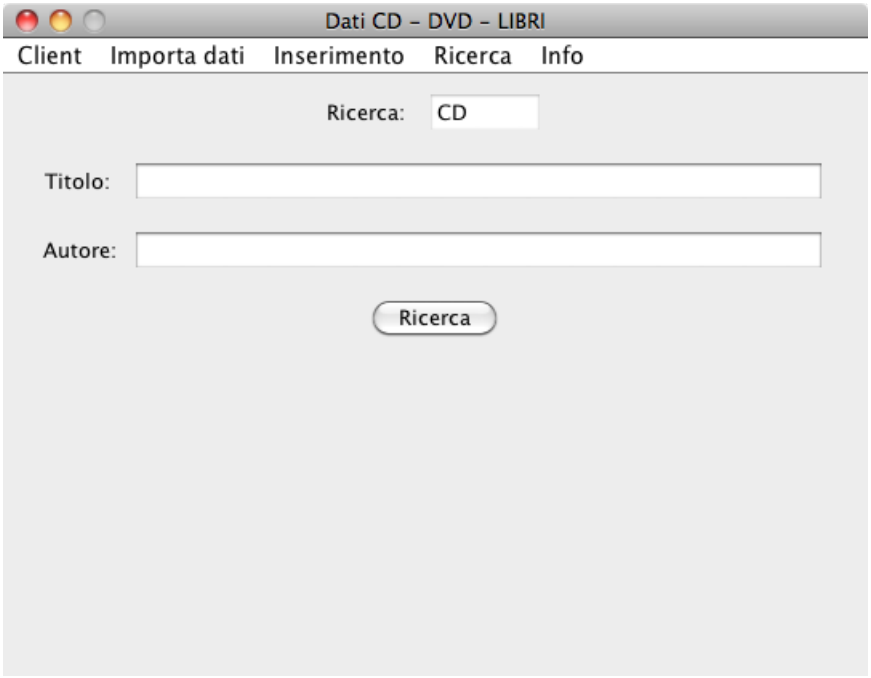

Figura 4.8: Finestra ricerca CD

Se non valorizzato né il titolo né l'autore verranno estratti tutti gli articoli relativi a quel prodotto. I risultati della ricerca sono visualizzati su di una tabella, sarà possibile selezionarne uno o più di uno.

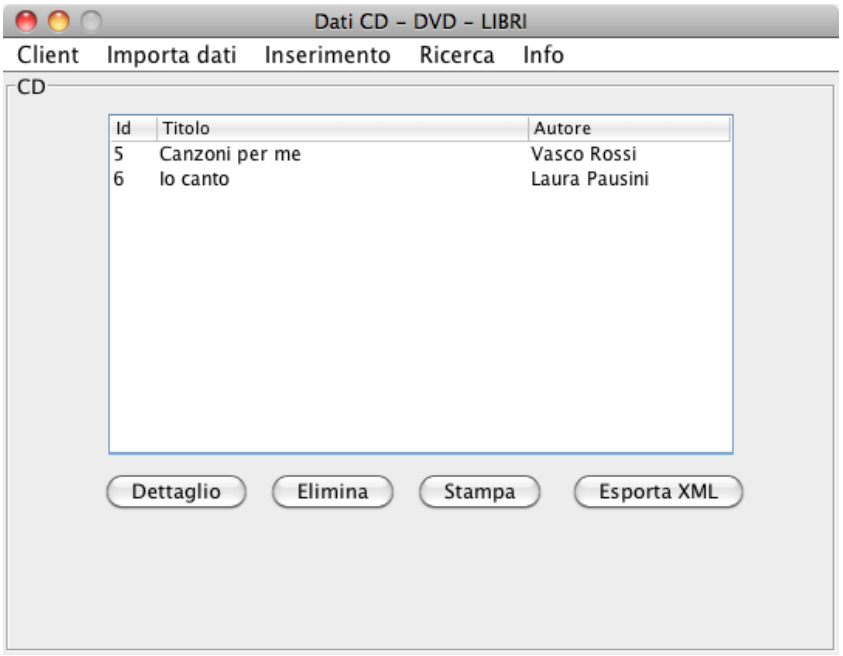

Figura 4.9: Finestra risultato ricerca CD

Sotto la tabella sono elencate le funzionalità previste ovvero:

Dettaglio: consente di aprire la pagina di dettaglio già compilata con i dati del prodotto selezionato

Figura 4.10: Finestra dettaglio di un CD

La finestra mostrerà tutti i dati del prodotto modificabili.

Elimina: consente di eliminare il prodotto selezionato

Figura 4.11: Messaggio di avviso avvenuta eliminazione

dopo l'eliminazione un messaggio avviserà l'utente dell'esito della cancellazione.

Stampa: crea un file pdf contenente i dati selezionati, chiede di inserire il nome del

file

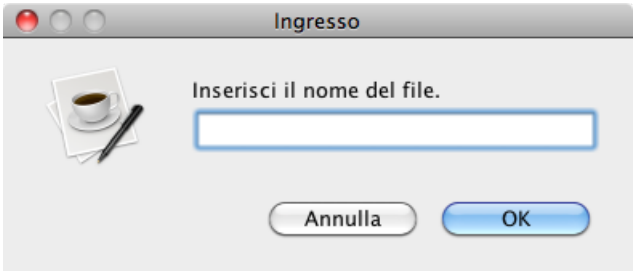

Figura 4.12: Finestra inserimento nome del file PDF da creare

anche in questo caso un messaggio avviserà l'utente dell'esito positivo della creazione del file PDF.

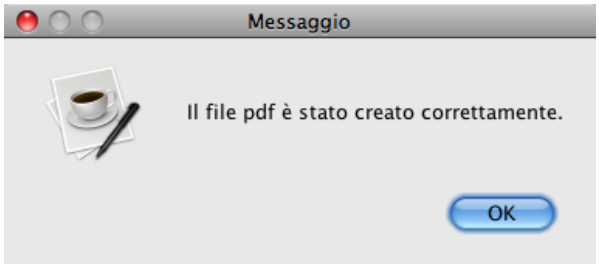

Figura 4.13: Messaggio di avviso di avvenuta creazione PDF

Aprendo il file si ha un esempio simile:

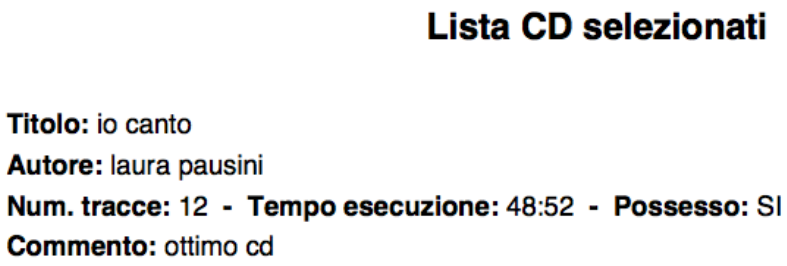

Figura 4.14: Esempio file PDF creato

Esporta XML: crea un file XML contenente i dati selezionati.

Il contenuto del file XML che potrà essere utilizzato per la fase di importazione è del tipo:

```
<?xml version="1.0" encoding="UTF-8"?>
<cds>
 <cd>
  <id>9</id>
  <titolo>Volare</titolo>
  <autore>Tenco</autore>
  <tempoesecuzione>12:23</tempoesecuzione>
  <numerotracce>23</numerotracce>
  <possesso>S</possesso>
  <immagine></immagine>
  <commento>ottimo album</commento>
 </cd>
 <cd>
  <id>9</id>
  <titolo>Alba chiara</titolo>
  <autore>V.Rossi</autore>
  <tempoesecuzione>3:34</tempoesecuzione>
  <numerotracce>12</numerotracce>
  <possesso>N</possesso>
  <immagine></immagine>
  <commento>da ascoltare</commento>
 </cd>
</cds>
```

Figura 4.15: Tracciato XML del file di importazione

Il cui DTD è il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT cds (cd)+>
<!ELEMENT singolocd (id, titolo, autore, tempoesecuzione, numerotracce, possesso, immagine,
commento)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT autore (#PCDATA)>
<!ELEMENT tempoesecuzione (#PCDATA)>
<!ELEMENT numerotracce (#PCDATA)>
<!ELEMENT possesso (#PCDATA)>
<!ELEMENT immagine (#PCDATA)>
<!ELEMENT commento (#PCDATA)>
```

Figura 4.16: DTD relativo al tracciato di importazione

4.1 Tecnologie e strumenti utilizzati

Le tecnologie usate per creare l'applicativo sono degli standard de facto, cioè che anche se non sono definiti standard dalle associazioni ufficiali molti che operano nel settore il adoperano come tali. Adesso passiamo a descrivere brevemente quali sono le tecnologie usate.

Eclipse

Eclipse è un ambiente di sviluppo open source che rappresenta lo state of art di Java. Per questa piattaforma è stata adottata la filosofia open source in modo che possa essere ampliato facilmente da terze parti. Eclipse è un Integrated Development Environment (IDE) perchè fornisce strumenti per gestire il workspace; per generare, eseguire e compiere il debug delle applicazioni che vengono create; condividere ciò che si è fatto con altre persone e facilitare la N-version programming; e customizzare facilmente l'esperienza di programmazione. Eclipse è una piattaforma perché non un'applicazione finita ma è progettato per essere espanso in modo indefinito con tool sempre più sofisticati e avanzati. Eclipse è utilizzabile per "qualunque cosa" in quanto viene usato con successo per costruire ambienti che hanno molti contenuti, come lo sviluppo di programmi in Java, i Web Service, la programmazione di dispositivi embedded, e l'uso in contesti di programmazione di giochi. La predominanza come tool per la programmazione in Java è solamente per ragioni storiche. La piattaforma Eclipse non ha nessun supporto esplicito o implicito per la programmazione in Java come definito dallo Java Development Tool (JDT). Lo JDT segue le stesse regole che hanno tutti i plugins che sono all'interno della piattaforma.

MySQL Version: 5.4.1 (Mac os X)

MySQL è un Relational database management system (RDBMS), composto da un client con interfaccia a caratteri e un server, entrambi disponibili sia per sistemi Unix come GNU/Linux che per Windows, anche se prevale un suo utilizzo in ambito Unix.

Dal 1996 supporta la maggior parte della sintassi SQL e si prevede in futuro il pieno rispetto dello standard ANSI. Possiede delle interfacce per diversi linguaggi, compreso un driver ODBC, due driver Java e un driver per Mono e .NET.

Il codice di MySQL viene sviluppato fin dal 1979 dalla ditta TcX ataconsult, adesso MySQL AB, ma è solo dal 1996 che viene distribuita una versione che supporta SQL, prendendo spunto da un altro prodotto: mSQL. Il codice di MySQL è di proprietà della omonima società, viene però distribuito con la licenza GNU GPL oltre che con una licenza commerciale. Fino alla versione 4.0, una buona parte del codice del client era licenziato con la GNU LGPL e poteva dunque essere utilizzato per applicazioni commerciali. Dalla versione 4.1 in poi, anche il codice dei client è distribuito sotto GNU GPL. Esiste peraltro una clausola estensiva che consente l'utilizzo di MySQL con una vasta gamma di licenze libere. MySQL svolge il compito di DBMS nella piattaforma LAMP, una delle più usate e installate su Internet per lo sviluppo di siti e applicazioni web dinamiche.

Nel luglio 2007 la società svedese MySQL AB ha 385 dipendenti in numerosi paesi. I suoi principali introiti provengono dal supporto agli utilizzatori di MySQL tramite il pacchetto Enterprise, dalla vendita delle licenze commerciali e dall'utilizzo da parte di terzi del marchio MySQL. Il 16 gennaio 2008 Sun Microsystems ha acquistato la società per un miliardo di dollari, stimando il mercato del database in 15 miliardi di dollari. Il 20 aprile 2009 alla stessa Sun Microsystems è stata proposta l'acquisizione da parte di Oracle per 7,4 miliardi di dollari. L'accordo, approvato dall'antitrust USA, è ancora al vaglio degli organi corrispondenti dell'Unione Europea, preoccupati dal conflitto di interessi costituito dai database commerciali Oracle rispetto a MySQL.

MySQL Query Browser

MySQL Query Browse è uno strumento grafico fornito da MySQL AB per creare, eseguire ed ottimizzare query in ambiente grafico. Se da una parte tutte le query eseguite in MySQL Query Browser possono essere effettuate anche tramite l'utility da riga di comando mysql, MySQL Query Browser consente di eseguire query e modificare dati in maniera più intuitiva, tramite un'interfaccia grafica. MySQL Query Browser è progettato per funzionare con MySQL versione 4.0 e superiori. MySQL Query Browser è prevalentemente il risultato del feedback che MySQL AB ha ricevuto su un periodo di molti anni da molti utenti.

iTEXT

iText è una libreria java per la generazione e/o modifica dinamica di documenti PDF, con essa è possibile creare molto velocemente e facilmente report anche complessi contenenti tabelle ed altri tipi di formattazione.

È sufficiente creare un oggetto di tipo Document, ottenere un'istanza di PDFWriter ed iniziare ad aggiungere elementi al documento.

Gli elementi principali sono:

- Chunk: ovvero il più piccolo elemento di testo che può essere aggiunto, per esempio una frase
- Paragraph: ovvero una specie di ArrayList di Chunk, Phrase, Image, List, etc, può contenere testo formattato con stili diversi, e si può specificare un allineamento
- List: ovvero degli elenchi puntati e/o numerati
- Image: ovvero immagini, nei più disparati formati
- Table: ovvero tabelle, per le quali è possibile specificare lo stile dei bordi

MySQL Connector

Per le applicazioni Java, che utilizzano i driver JDBC, MySQL fornisce connettività attraverso MySQL Connector/J. Si tratta di driver JDBC di Tipo 4, che significa che è scritto in Java e comunica direttamente col server attraverso il protocollo MySQL.

Capitolo 5

Conclusioni

Volendo riassumere questo lavoro è possibile dire che è stata sviluppata una applicazione per eseguire dei test su un archivio di dati multimediali lavorando sia con file XML che con database relazionale.

Il punto di forza di XML è il fatto che permette di rappresentare contenuti in un formato compatibile con qualsiasi piattaforma hardware e software: il formato testuale puro. I dati in formato XML sono autodescrittivi e presentano notevoli similitudini con quelli semi-strutturati. È importante notare comunque che XML è semplicemente un linguaggio di markup, non è associato ad un modello di rappresentazione dei dati.

Rispondere alla domanda se è meglio utilizzare i dati in formato XML o memorizzarli su di un database relazionale, non è semplice nè tanto meno la risposta potrebbe essere universale. La scelta non può che dipendere dai dati che si devono trattare e dalle relazioni che esistono tra loro. Se i file contengono dati correlati e se si devono modificare/inserire/cancellare dei record forse è il caso di affidarsi ad un db relazionale.

Se i file invece contengono solo dati scorrelati tra di loro e prevalentemente statici ovvero che non devono essere modificati di frequente, può anche essere più adatto l'utilizzo di file xml. Si deve tuttavia tenere presente che potrebbero verificarsi dei problemi di concorrenza nell'accesso ai file e sicuramente un accesso continuo a tanti file non è

ottimale dal punto di vista prestazionale.

Inoltre se i dati non vengono scambiati con nessun altro sistema non risulta conveniente usare l'xml, l'esecuzione di marshalling e unmarshalling anche se i dati non verranno modificati abbassa sostanzialmente le prestazioni.

Il progetto è stato realizzato cercando di rendere il più possibile modulare il codice al fine di facilitarne un successivo sviluppo ed ampliamento delle funzionalità del sistema. Inoltre l'implementazione dei moduli di import ed export dei file xml evidenziano una limitazione dovuta al tracciato delle tabelle del DB. Cambiare la struttura di una tabella, aggiungendo o eliminando colonne, comporta necessariamente rivedere la funzionalità di import ed export dei file XML.

Una possibile evoluzione del progetto potrebbe essere quella di utilizzare per l'esportazione e l'importazione dei dati un meccanismo automatico che consenta la creazione di file XML ed il relativo caricamento a partire dalla struttura di una tabella. Tale tecnologia che sta sempre più prendendo piede in ambiente MySQL è nota come MySQL XQuery. XQuery (XML Query Language) è un linguaggio di programmazione definito dal W3C (World Wide Web Consortium) concepito per l'interrogazione di documenti e basi di dati XML. XQuery prende le mosse da SQL ma utilizza documenti XML come basi di dati. Mentre i dati memorizzati nei database relazionali tendono ad assumere una forma piatta oppure tabellare, i dati salvati con XML sono, per definizione, irregolari e nidificati. Oggi, anche per la gestione di semplici servizi web utilizzano DBMS più o meno complessi. Dal momento che XML sta divenendo sempre più usato, un meccanismo basato sull'uso di XQuery anziché su uno governato via SQL potrebbe rivelarsi radicalmente più semplice e probabilmente più performante.

Bibliografia

[1] [ACPT09] Atzeni P., Ceri S., Paraboschi S., Torlone R., Basi di dati - *Modelli e linguaggi di interrogazione* - terza edizione, Mc Graw Hill, 2009.

[2] [ACFPT07] Atzeni P., Ceri S.,Fraternali P., Paraboschi S., Torlone R., Basi di dati - *Architetture e linee di evoluzione* - seconda edizione, Mc Graw Hill, 2007

[3] [BMAR01] Benoit Marchal, XML practical Jackson, 2001

[4] [ZARG07] Zarrelli Giorgio, *Programmare con MySQL*, Edizioni FAG, 2007

[5] [KOFLM06] Kofler Michael,*MySQL 5 Guida Completa*, Apogeo, 2006

[6] [ARNE07] Arlow J., Neustadt I., *UML e Unified Process. Analisi e progettazione object-oriented*, McGraw-Hill, 2007

[7] [PEPO08] Perdita S., Pooley R., *Usare UML. Ingegneria del software con oggetti e componenti*, Pearson Education Italia, 2008

[8] [BUTSIM07] Buttarazzi B., Simonetta A., *Programmando Java. Introduzione a UML*, Carocci, 2007.

[9] [MAGD09] Maggiorini D., *Introduzione alla programmazione client*, Pearson Education Italia, 2009

[10] [PIGFER08] Pighizzini G., Ferrari M., *Dai fondamenti agli oggetti. Corso di programmazione Java*, Pearson Education Italia, 2008

[11] [FBEHR08] Forouzan Behrouz A., *Reti di calcolatori e Internet*, McGraw-Hill, 2008

[12] [SAVD09] Savini D., Java. *Tecniche di programmazione, Tecniche Nuove*, 2009

[13] [OPPA06] Oppel A., *SQL. Tecniche e soluzioni*, McGraw-Hill, 2006

[14] [CADPN05] Camagni P., Della Puppa M., Nikolassy R., *SQL. Il linguaggio per le basi di dati*, Hoepli, 2005

Risorse disponibili sul WEB

[15] [SISINF] `http://it.wikipedia.org/wiki/Sistema_informativo` 06/2010

[16] [MySQL] `http://www.mysql.it/why-mysql/white-papers/` 06/2010

[17] [JAVA] `http://java.sun.com/javase/reference/index.jsp` 06/2010

[18] [RDBMS]`http://it.wikipedia.org/wiki/Relational_database_management_system` 06/2010

[19] [XML]`http://www.mondodigitale.net/Rivista/05_numero_tre/Braga,_Ceri_p._45-58.pdf` 06/2010

[20] [OOP]`http://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti` 06/2010

[21] [DTD]`http://it.wikipedia.org/wiki/Document_Type_Definition` 06/2010

[22] [UML]`http://it.wikipedia.org/wiki/Unified_Modeling_Language` 06/2010

[23] [UCD] `http://en.wikipedia.org/wiki/Use_case_diagram` 06/2010

[24] [CLD] `http://en.wikipedia.org/wiki/Class_diagram` 06/2010

[25] [QBJD] `http://en.wikipedia.org/wiki/Object_diagram` 06/2010

[26] [STCD] http://it.wikipedia.org/wiki/Statechart_Diagram 06/2010

[27] [ACTD] http://it.wikipedia.org/wiki/Activity_diagram 06/2010

[28] [SEQD] http://en.wikipedia.org/wiki/Sequence_diagram 06/2010

[29] [DEPD] http://en.wikipedia.org/wiki/Deployment_diagram 06/2010

[30] [SQL] http://it.wikipedia.org/wiki/SQL 06/2010

Ringraziamenti

Ringrazio la mia famiglia per avermi dato la possibilità di arrivare fino a questo punto. Dedico queto momento anche a mio fratello Federico che auspico possa riprendere gli studi quanto prima. Mi scuso con lui di non essere stato sufficientemente presente in questi ultimi due anni poichè gli studi mi hanno parzialmente isolato. Ringrazio soprattutto mio padre che ha sempre creduto in me e nei momenti di difficoltà mi ha dato la forza di andare avanti con coraggio e fermezza. Ringrazio la mamma per aver condiviso con me momenti di gioia e di difficoltà durante questo percorso. Ringrazio la Prof.ssa Rita Fioresi, docente di matematica di questo corso, che è stata la mia prima insegnante del primo anno che mi ha introdotto in questo percorso. Ringrazio sentitamente il mio relatore, Professor Antonio Messina, che mi ha dato la possibilità di laurearmi prima della scadenza dell'anno accademico. Ringrazio ulteriormente il Professore Antonio Messina per avermi dato la possibilità di preparare in tempi brevi la tesi. Ringrazio anche il mio gruppo di amici di studio con il quale ho condiviso questa esperienza.

Stefano Ballardini, Maria Barracu, Beatrice Bottoni, Andrea Cinelli, Lorenzo Conti, Alessandro Degli Antoni, Nazareno De Luca, Alessandro De Troia, Matteo Fanciulli, Marco Gasperini, Eugenio Ligabue, Andrea Lugli, Giovanni Morbelli, Michele Nardi, Nioi Giorgia, Alessandro Rapisarda, Claudio Rossi, Isacco Rubini, Stefano Pagliarani.

Un particolare ringraziamento a:

Beatrice Bottoni

Lorenzo Conti

Alessandro Degli Antoni

Alessandro De Troia

Andrea Lugli

Alessandro Rapisarda