UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale in Ingegneria del Software II

Generazione di codice con SkyGen: un caso di studio

Anno Accademico 2013/2014

relatore
Ch.mo Prof. Porfirio Tramontana

correlatore
Dott.ssa Mariagabriella Mastroianni

candidato
Miriam Durazzo
matr. M63/000282

A mio padre, il mio punto di riferimento.
A mia madre,la mia carica.
Ad Alberico, la mia forza.

Indice

Introduzione

Le applicazioni web, con il passare degli anni, stanno assumendo un ruolo sempre più importante come supporto al mondo del lavoro, anche al di fuori del settore dell'informatica. Ormai, sono all'ordine del giorno processi aziendali che coinvolgono un numero elevato di attori dislocati in diverse sedi, sempre più spesso situate in diverse città o addirittura in diversi paesi. La necessità di avere degli strumenti che consentano di fornire un adeguato sostegno nella gestione di tali processi porta inevitabilmente all'esplorazione del terreno ad oggi più fertile nell'ambito della distribuzione di applicazioni: il web. Fino a pochi anni fa tutte le applicazioni si basavano sulle onerose architetture client-server: su ogni postazione utente doveva essere installato un software necessario per l'interazione con il sistema. In questo modo qualsiasi modifica costringeva le aziende a sostenere elevati costi, sia in termini economici che temporali.

L'arrivo delle applicazioni web ha rappresentato una vera e propria svolta nello sviluppo di software orientato al business. Infatti, attraverso il loro utilizzo le aziende possono ottenere dei sistemi integrati, distribuiti, facilmente mantenibili ed aggiornabili a costi sensibilmente ridotti. Per questo, anche nell'ambito della gestione dei processi, si tende a sfruttare le potenzialità delle applicazioni web. Nel contesto dei processi aziendali, inoltre, è sempre più frequente la necessità di supportare la gestione del workflow con degli strumenti in grado di facilitare tutti gli aspetti relativi alla sua progettazione, realizzazione

e modifica. È quindi necessario estendere le tradizionali applicazioni web con nuove funzionalità.

Con questa tesi, dunque, si sono voluti esplorare e approfondire tutti gli aspetti fondamentali dello sviluppo di applicazioni software orientate al web e la loro realizzazione mediante l'utilizzo di un Code Generator. L'attività di tirocinio, svolta presso l'azienda SkyIt s.r.l. di Roma, ha avuto come oggetto la progettazione e l'implementazione di un'applicazione web, all'interno di un progetto denominato "Space Classroom". Questa attività, fin da subito, ha avuto come obiettivo quello di acquisire sufficienti conoscenze riguardanti l'architettura J2EE, paradigmi di programmazione e strumenti utilizzati nell'ambito delle web application. Tali conoscenze hanno permesso l'implementazione di un'applicazione accessibile tramite browser web, che consentisse ai propri utenti di inviare delle comunicazioni, aggiornando una base dati. La piattaforma standard di sviluppo J2EE 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 una parte delle tecnologie di questa piattaforma, necessarie per lo sviluppo del progetto. Le componenti dello standard che verranno trattate in questa tesi sono: Servlet, JSP, JDBC e JavaBeans.

L'azienda ospitante, inoltre, nelle proprie scelte progettuali, è piuttosto consona nell'utilizzo di alcuni paradigmi di progettazione di applicazioni web. Infatti, dovendo rispondere alle attuali esigenze di mercato dell' Information Technology che impone la realizzazione, in tempi estremamente brevi, di soluzioni efficaci, complesse e funzionali, il tutto opportunamente documentato, l'azienda ha realizzato una suite di prodotti, Sky*, che è nata proprio con l'intento di soddisfare queste esigenze. Lo strumento SkyGen, Application Framework e Code Generation, è senza dubbio il componente più importante dell'intera suite di sviluppo. Esso è basato sul principio del Model Driven Development (MDD) e consente di trasformare modelli UML in artefatti. Tale strumento è stato oggetto, in questa tesi, di una approfondita analisi. L'obiettivo, infatti, era quello di capire quanto l'utilizzo di uno strumento CASE potesse essere vantaggioso nella realizzazione di

un'applicazione web. I CASE sono strumenti che supportano lo sviluppo de software attraverso interfacce grafiche e librerie di funzionalità. L'acronimo sta per Computer-Aided Software Engineering, ovvero sviluppo del software assistito dal computer. Tali sistemi hanno notevolmente semplificato la scrittura di linee di codice con relativi benefici anche alla conformità agli standard del software stesso. Per meglio comprendere la grande utilità dello strumento SkyGen nella realizzazione, in tempi brevi e con uno sforzo minore da parte degli sviluppatori, delle web-applications, si sono analizzati anche gli strumenti Eclipse Modeling Framework e Spring Roo, tool open source che permettono di ottenere una maggiore integrazione tra la fase di design e quella di implementazione, e sono in grado di generare codice a partire da un modello.

Successivamente a tale momento di confronto, si è osservato lo strumento SkyGen in azione. È stata, infatti, descritta dettagliatamente l'applicazione realizzata in fase di tirocinio, dall'analisi dei requisiti si passerà alla sua implementazione. Infine, si è cercato di capire, attraverso un' approfondita analisi del codice, quale sia la qualità del codice generato dallo strumento preso in esame e quale la sua effettiva utilità nella realizzazione di applicazione web.

Capitolo 1: J2EE

La tecnologia **Java 2 Enterprise Edition** (**J2EE**) è diventata, negli anni, sinonimo di sviluppo di applicazioni aziendali sicure, robuste, ed efficienti. Tali caratteristiche la rendono tra le più importanti piattaforme tecnologiche di sviluppo, soprattutto in ambiti in cui la sicurezza e la robustezza sono vincoli imprescindibili (ad esempio applicazioni bancarie).

Il motivo del grande successo di questa tecnologia è dovuto all' utilizzo del **linguaggio object oriented Java** e a come attraverso esso la J2EE è stata creata. La specifica è un continuo lavoro delle più importanti aziende di Information Technology. Oltre a Sun, madre del linguaggio Java, hanno collaborato alla sua definizione aziende, come ad esempio, IBM, Oracle, BEA.

Per capire i motivi che hanno spinto gli ingegneri della Sun a produrre la tecnologia J2EE (Java 2 Enterprise Edition, o nelle più recenti versioni solo **JEE**) bisogna soffermarsi sul significato della prima "E": *Enterprise*, tradotto in italiano vuol dire "impresa", "azienda", quindi si potrebbe dire che la tecnologia J2EE viene incontro alle esigenze aziendali, alla progettazione ed allo sviluppo di applicazioni che devono rispondere a criteri di affidabilità e robustezza in un contesto distribuito.

Di seguito ci occuperemo dei principali aspetti della tecnologia, il cui obiettivo principale è quello di permettere un modello di sviluppo basato su componenti semplici, modulari e sicuri.

1.1 La piattaforma di sviluppo J2EE

J2EE è una piattaforma per lo sviluppo di applicazioni software distribuite aziendali. Sin dalla nascita del linguaggio Java, essa ha subito enormi adozioni e una forte crescita. Infatti, sempre più tecnologie sono diventate parte della piattaforma Java, e nuove API e standard sono stati sviluppati per affrontare diverse e nuove esigenze . Alla fine, Sun e un gruppo di aziende leader del settore, sotto la tutela della Open Java Community Process (JCP), hanno unificato tutti gli standard e tutte le API connesse con le imprese nella piattaforma J2EE.

La piattaforma J2EE offre numerosi vantaggi per l'impresa:

- J2EE stabilisce gli standard per le aree di esigenze informatiche dell'azienda come la connettività ai database, i componenti di business aziendali, i middleware orientati ai messaggi (MOM), le componenti Web correlate, i protocolli di comunicazione e le interoperabilità.
- J2EE promuove le implementazioni "best-of-breed" basate su standard aperti, e proteggendo gli investimenti tecnologici.
- J2EE fornisce una piattaforma standard per la creazione di componenti software che sono portatili attraverso le implementazioni dei fornitori , evitando il fenomeno del "vendor lock-in".
- J2EE aumenta il time-to-market, in quanto gran parte delle infrastrutture vengono fornite e le organizzazioni IT possono, quindi, uscire dal business middleware e concentrarsi sulla creazione di applicazioni per il loro business.
- J2EE aumenta la produttività dei programmatori. Tutto il software enterprise, infatti, può essere realizzato sotto la piattaforma J2EE, utilizzando Java come linguaggio di programmazione.
- J2EE promuove l'interoperabilità all'interno di ambienti eterogenei esistenti.

J2EE rappresenta, quindi, una base solida per lo sviluppo di un'applicazione, dalla più semplice alla più complessa realtà aziendale. Come accennato in precedenza, la piattaforma J2EE fornisce un'ampia serie di specifiche e interfacce, Application

Programming Interface (API), che definiscono le componenti tecnologiche con cui è strutturata e la modalità di interazione tra di esse. Un'applicazione che implementa le specifiche J2EE è un'applicazione distribuita *"multi-tier"*, schematizzabile in quattro layer:

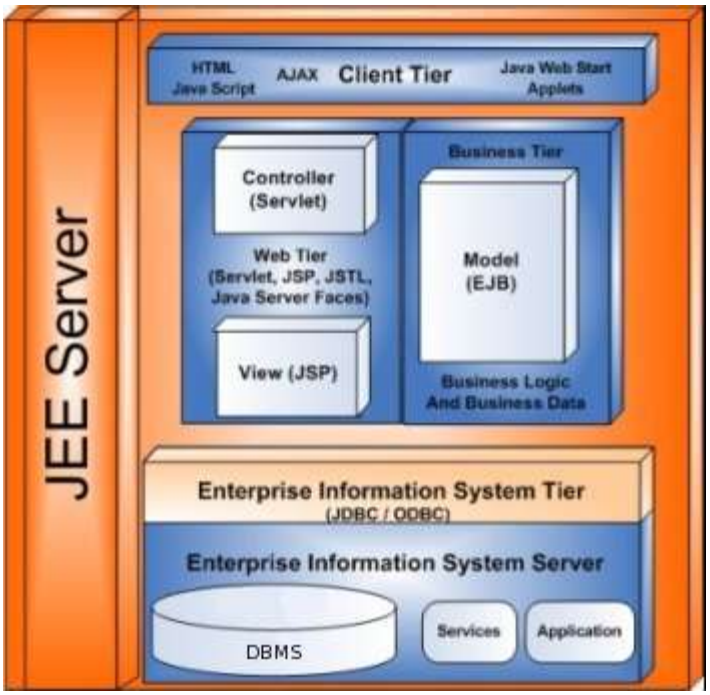


Figura 1. Architettura di una generica applicazione J2EE

1. *Client Tier*. Presenta all'utente finale i risultati dell'elaborazione del server. Spesso è rappresentato da un browser web, ma può talvolta essere costituito da client specifici, in grado di interagire direttamente con il Business Tier o il Data Tier.
2. *Web Tier.* In funzione nel server J2EE (o semplicemente in un Web Server), comprende una serie di componenti che riguardano il lato front-end dell'applicazione, mediando richieste e risposte del protocollo HTTP tra client e server.
3. *Business Tier.* In funzione nel server J2EE, dove viene implementata la logica di business dell'applicazione, organizzando i dati e l'accesso ad essi ed interagendo,

ad esempio, con un DBMS. Nelle applicazioni più semplici viene accorpato al Web Tier.

4. *EIS Tier*. Spesso rappresentato da un DBMS (Database Management System) o più in generale da un EIS (Enterprise Information System).

I principali vantaggi derivanti dall'uso delle specifiche J2EE per lo sviluppo di applicazioni web sono quattro:

- *Scalabilità:* è possibile aumentare le funzionalità di un software in continua evoluzione, grazie anche alla peculiare proprietà di distribuibilità della tecnologia Java.
- *Portabilità:* esigenza fondamentale dell'attività di sviluppo software, permette di utilizzare una stessa applicazione J2EE in Application Server diversi, purché questi implementino le specifiche J2EE.
- *Efficienza:* una strutturazione così intensa, facilita la gestione di progetti anche molto complessi. Inoltre, grazie all'utilizzo del multi-threading di Java è possibile ottenere elevate performance per l'interazione tra Client e Server.
- *Sicurezza.* È assicurata dall'elevata stratificazione dell'architettura e dalle proprietà intrinseche della tecnologia Java.

Queste proprietà rappresentano le esigenze dell'azienda che lavora con ambienti composti da tipologie di risorse differenti e distribuite, assicurando una buona robustezza software. Inoltre, attraverso il modello proposto, si rendono facile l'accesso ai dati e la loro rappresentazione in diverse forme (un browser web, un applet, un disposi-tivo mobile, un sistema esterno, ecc).

1.2 L'Application Server

L'**Application Server** è uno dei più importanti elementi di un'applicazione web. Esso, implementando le specifiche JEE, rappresenta uno strato software che fornisce i servizi e le interfacce per la comunicazione tra tutte le componenti, inoltre, permette

all'applicazione di funzionare. Molti degli Application Server presenti sul mercato implementano sia il livello Web-Tier che il Business-Tier.

Le applicazioni più semplici,ovvero quelle che non possiedono una vera e propria logica di business, dovrebbero utilizzare Application Server con il solo ruolo di *Web-Container*, componente fondamentale dell'architettura J2EE la cui funzione principale consiste nella presentazione dei dati. Tali Application Server sono chiamati Web Server e sono consigliati in quelle circostanze in cui le risorse richieste sono limitate. Tra i principali componenti Java gestiti da un Web-Container vi sono:

- *Servlet, JSP (Java Server Pages),*
- *JDBC (Java DataBase Connectivity),*
- *JNDI (Java Naming and Directory Interface).*

Un Application Server può fornire, inoltre, un ambiente detto *EJB-Container (Enter-prise Java Beans Container)* che contiene la logica di business dell'applicazione. Gli elementi fondamentali di questo ambiente sono gli Enterprise JavaBeans, le cui specifiche sono state introdotte per la prima volta da IBM nel 1997 e successivamente accorpate nelle API standard di Sun Microsystem. Il compito di un EJB-container è quello di gestire la sicurezza, i dettagli di persistenza e l' integrità delle transazioni, lasciando agli sviluppatori maggior concentrazione sui problemi del *core business*. La scelta di quale Application Server utilizzare è influenzata da numerosi fattori, tra cui:

- Tecnologie preesistenti. Alcuni case software dei principali DBMS commerciali propongono i propri Application Server. Scegliere una simile soluzione, di solito, 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 (molto più importanti).
- Supporto a pagamento. È un fattore importante soprattutto per le grandi aziende, che possono permettersi di sostenerne i costi.

- Supporto della rete. Gli Application Server molto diffusi comportano una maggior quantità di informazioni in rete e una maggior possibilità di reperire soluzioni a costi irrisori. E' il fattore principale delle piccole aziende.
- Portabilità. È uno degli aspetti fondamentali del successo di Java per gli Application Server. 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 J2EE (chi completamente e chi in parte) 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 J2EE.
- BEA Weblogic: soluzione commerciale proposta da Oracle.
- IBM WebSphere

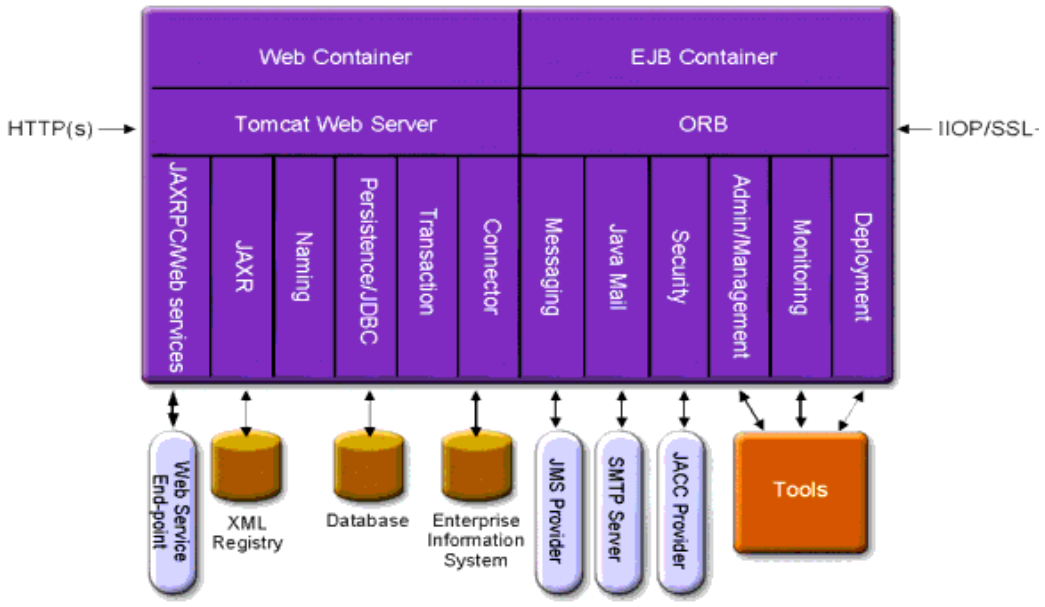


Figura 2. Architettura di un Application Server

Nelle diverse aziende, oggigiorno, la conoscenza di tecnologie e software diversi è importante anche al fine di poter integrare i propri software nelle più diverse realtà aziendali dei propri clienti.

1.3 Servlet

Una **Servlet** è un programma scritto in Java, residente su un server ed è in grado di gestire le richieste generate da uno o più client, attraverso uno scambio di messaggi tra il server ed i client stessi. Per eseguire una servlet è necessario disporre di un server che possa fungere da container, fornendo, in tal modo, un adeguato supporto alla loro attivazione ed esecuzione. I server più utilizzati a questo scopo sono Tomcat e JBoss. Dunque, le servlet, tipicamente, sono collocate all'interno di Application Server o Web Application Server come, ad esempio, Tomcat.

Essendo scritte in Java, le servlet possono avvalersi interamente delle Java API che, come è noto, consentono di implementare con relativa semplicità anche svariate funzionalità complesse ed importanti come l'accesso ad un database. Ad esse si aggiungono, inoltre, le più specifiche servlet API, che mettono a disposizione un' interfaccia standard utilizzata per gestire la comunicazione tra un client Web ed una Servlet, permettendo di programmare in maniera completamente indipendente dalle caratteristiche del server, del cliente e del protocollo di trasferimento.

È importante sottolineare che le Servlet non hanno delle GUI associate direttamente ad esse, pertanto, le librerie AWT e Swing non verranno mai utilizzate direttamente quando si desidera implementare una Servlet.

Altrettanto interessante è il fatto che, almeno in teoria, una Servlet non rappresenta unicamente (come invece si è spesso portati a ritenere) applicazioni basate sul protocollo HTTP (ovvero, un'applicazione di tipo Web). Per gli scenari non Web, infatti, si può fare riferimento alla classe javax.servlet.GenericServlet che, appunto, è in grado di utilizzare un protocollo generico. D'altra parte è necessario ammettere che, quasi sempre, quando si parla di Servlet si fa riferimento implicitamente ad una estensione particolare della classe

GenericServlet identificata, per la precisione, dalla classe HttpServlet. Entrambe queste classi (GenericServlet e HttpServlet) implementano l'interfaccia javax.servlet.Servlet.

1.3.1 Ciclo di vita di una Servlet

Il ciclo di vita di una servlet è scandito da una serie di chiamate, effettuate dal container, a particolari metodi dell'interfaccia Servlet:

1. Inizializzazione. Quando il container carica la servlet chiama il suo metodo *init*. Tipicamente, questo metodo viene usato per stabilire connessioni a database e preparare il contesto per le richieste successive. Secondo l'impostazione del contesto e/o del contenitore, la servlet può essere caricata immediatamente all'avvio del server o ad ogni richiesta
2. Servizio. Le richieste dei client sono gestite dal container tramite chiamate al metodo service. Richieste concorrenti corrispondono a esecuzioni di questo metodo in *thread* distinti. L'implementazione dovrebbe essere quindi *thread-safe*. Il metodo service riceve le richieste dell'utente sotto forma di una ServletRequest e invia la risposta tramite una ServletResponse.
3. Finalizzazione. Quando il container decide di rimuovere la servlet, chiama il suo metodo *destroy.* Quest'ultimo è solitamente utilizzato per chiudere connessioni a database, o scaricare altre risorse persistenti attivate dal metodo init.

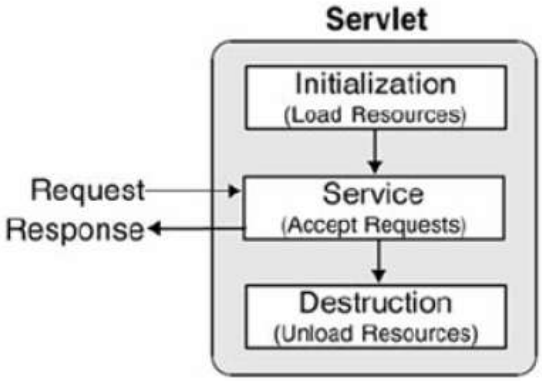


Figura 3. Flusso di dati in una Servlet

In particolare è la classe HttpServlet che specializza il sistema per la comunicazione http, fornendo, nello specifico, due metodi: *doGet* e *doPost*, corrispondenti alle due request più comuni in HTTP. Il metodo service della classe HttpServlet provvede automaticamente a smistare le richieste al metodo opportuno.

1.3.2 Flusso di una Servlet

Sarà ora illustrato come lavora una HttpServlet avvalendosi dello schema seguente:

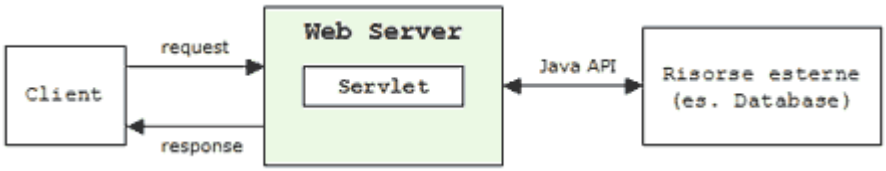


Figura 4. Flusso di dati in una Servlet

È possibile riassumere il flusso rappresentato nella figura sovrastante in questo modo:

1. Un client invia una richiesta (request) per una servlet ad un web application server.
2. Qualora si tratti della prima richiesta, il server istanzia e carica la servlet in questione avviando un thread che gestisca la comunicazione con la servlet stessa. Nel caso, invece, in cui la servlet sia già stata caricata in precedenza, il che, normalmente, presuppone che un altro client abbia effettuato una richiesta antecedente quella attuale, allora verrà, più semplicemente, creato un ulteriore thread che sarà associato al nuovo client, senza la necessità di ricaricare ancora la servlet.
3. Il server invia alla servlet la richiesta pervenutagli dal client.
4. La servlet costruisce ed imposta la risposta (response) e la inoltra al server
5. Il server invia la risposta al client.

È importante sottolineare che il contenuto della risposta non è necessariamente basato su un algoritmo contenuto all'interno della servlet invocata (questo può essere, semmai, il caso di applicazioni molto elementari) ma, anzi, è spesso legato ad interazioni della servlet

stessa con altre sorgenti di dati (data source) rappresentate, ad esempio, da Data Base, file di risorsa e, non di rado, altre Servlet.

Abbiamo visto, esaminando il flusso di esecuzione di una servlet, che il flusso stesso è incentrato su due componenti fondamentali: la richiesta (request, inviata dal client verso il server) e la risposta (response, inviata dal server verso il client). In Java, questi due componenti sono identificati, rispettivamente, dalle seguenti interfacce:

- javax.servlet.http.HttpServletRequest
- javax.servlet.http.HttpServletResponse

Un'istanza di ServletRequest viene passata dal contesto alla servlet quando questa viene invocata e contiene tutte le informazioni inerenti la richiesta effettuata dal client, ovvero i parametri GET e POST inviati dal client, le variabili di ambiente del server, gli header e i playload della richiesta http. Un'istanza di ServletResponse, invece, viene passata alla servlet quando le si richiede di restituire del contenuto da inviare al client. I metodi di questa classe permettono di scrivere su uno stream che sarà poi indirizzato al client, modificare gli header della risposta HTTP, ecc.

Un'altra importante interfaccia messa a disposizione dal Servlet engine è la javax.servlet.ServletContext. Attraverso di essa è possibile trovare un riferimento al contesto (context) di un'applicazione, ovvero ad una serie di informazioni a livello globale condivise tra i vari componenti che costituiscono l'applicazione stessa.

1.3.3 I metodi di una Servlet

Creare una Servlet vuol dire, in termini pratici, definire una classe che derivi dalla classeHttpServlet. I metodi più comuni per molti dei quali si è soliti eseguire l'overriding nella classe derivata sono i seguenti:

- void doGet (HttpServletRequest req, HttpServletResponse resp): gestisce le richieste HTTP di tipo GET. Viene invocato da service().
- void doPost (HttpServletRequest req, HttpServletResponse resp): gestisce le richieste HTTP di tipo POST. È invocato da service().

- void service(HttpServletRequest req, HttpServletResponse resp): viene invocato al termine del metodo.
- void doPut (HttpServletRequest req, HttpServletResponse resp): è invocato attraverso il metodo service () per consentire di gestire una richiesta HTTP di tipo PUT. Tipicamente, una richiesta del genere consente ad un client di inserire un file su un server, similmente ad un trasferimento di tipo FTP.
- void doDelete(HttpServletRequest req, HttpServletResponse resp): viene invocato attraverso il metodo service() per consentire ad una Servlet di gestire una richiesta di tipo HTTP DELETE. Questo genere di richiesta è utilizzato per rimuovere un file dal server.
- void init (): è invocato soltanto una volta dal Servlet Engine al termine del caricamento della servlet ed appena prima che la servlet stessa inizi ad esaudire le richieste che le pervengono.
- void destroy (): è invocato direttamente dal Servlet Engine per scaricare una servlet dalla memoria.
- String getServletInfo(): è utilizzato per ricavare una stringa contenente informazioni di utilità sulla Servlet (ad esempio nome della Servlet, autore, copyright). La versione di default restituisce una stringa vuota.
- ServletContext getServletContext (): è usato per ottenere un riferimento all'oggetto di tipo ServletContext cui appartiene la Servlet.

1.3.4 I vantaggi delle Servlet

Ci si potrebbe chiedere quale sia il motivo per cui possa essere più vantaggioso utilizzare le Servlet piuttosto che affidarsi a tecnologie, ancora abbastanza utilizzate, come la Common Gateway Interface (CGI). Le differenze tra le due soluzioni non sono trascurabili. I vantaggi nell'utilizzo delle servlet sono:

- **Efficienza**. Come già detto le servlet sono istanziate e caricate una volta soltanto, alla prima invocazione. Tutte le successive chiamate da parte di nuovi client sono

gestite creando dei nuovi thread che si prendono carico del processo di comunicazione (un thread per ogni client) fino al termine delle rispettive sessioni. Con le CGI, tutto questo non accadeva. Ogni client che effettuava una richiesta al server causava il caricamento completo del processo CGI con un degrado delle performance facilmente immaginabile.

- **Portabilità**: Grazie alla tecnologia Java, le servlet possono essere facilmente programmate e "portate" da una piattaforma ad un'altra senza particolari problemi.
- **Persistenza**: Dopo il caricamento, una servlet rimane in memoria mantenendo intatte determinate informazioni (come la connessione ad un database) anche alle successive richieste.
- **Gestione delle sessioni**: Come è noto HTTP è un protocollo stateless (senza stati) e, pertanto non in grado di ricordare i dettagli delle precedenti richieste provenienti da uno stesso client. Le servlet (lo vedremo in un altro articolo) sono in grado di superare questa limitazione.

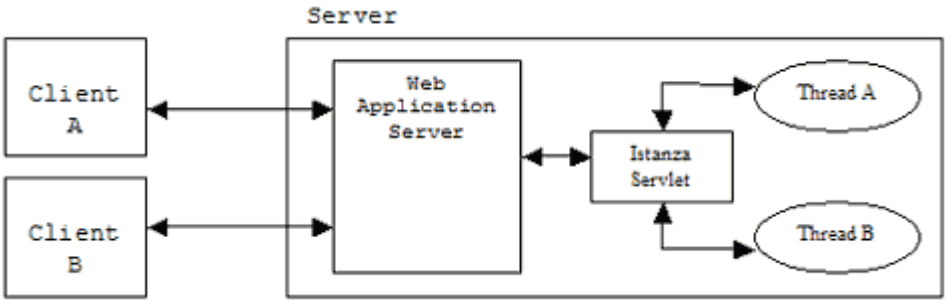


Figura 5. Un thread per ogni client

1.4 JSP

Quando si parla di creazione dinamica di pagine HTML, significa generalmente che tutto il lavoro di produzione è svolto sul lato server e quindi non sono richieste particolari funzionalità aggiuntive al browser, il quale deve semplicemente essere in grado di interpretare HTML standard.

Le servlet offrono una serie di vantaggi rispetto alle soluzioni concorrenti, legati al fatto che si utilizza Java come linguaggio di sviluppo. Le servlet, però, richiedono un po' di

lavoro in più per risolvere aspetti legati al lifecycle, al mantenimento dello stato, e passaggio di parametri fra oggetti differenti.

Le **Java Server Pages (JSP)** sono state pensate proprio con lo scopo di eliminare queste limitazioni, oltre che ad offrire ulteriori importanti funzionalità. Mentre le servlet sono classi Java con immersi tag HTML/XML, le JSP sono pagine HTML/XML all'interno delle quali è possibili inserire codice Java: questo codice verrà poi gestito dal server che provvederà alla sua compilazione ed esecuzione. Questo fatto permette al programmatore di inserire parti di codice Java in modo molto simile a come farebbe con script Java Script, VBscript o simili. Come conseguenza si ha un ulteriore impulso alle performance ed alla portabilità dato che il codice prodotto viene eseguito sul server, e non sul browser.

Le JSP, quindi, si basano su tecnologia Java ereditandone i vantaggi garantiti dalla metodologia object oriented e dalla quasi totale portabilità multipiattaforma. Quest'ultima caratteristica si rivela tanto più vantaggiosa quanto più ci fosse necessità di ambienti di produzione con sistemi operativi diversi tra loro (ad esempio Windows e Linux).

Essere object oriented significa anche avere nel DNA una predisposizione al riuso del codice: grazie ai Java Bean si possono includere porzioni di codice che semplificano l'implementazione di applicazioni anche complesse, senza approfondite conoscenze di Java, e ne rendono più semplice la modifica e la manutenzione.

La gestione delle sessioni introdotta con le JSP favorisce lo sviluppo di applicazioni di commercio elettronico fornendo uno strumento per memorizzare temporaneamente lo stato delle pagine fino alla chiusura del browser.

Una Java Server Page può essere invocata utilizzando due diversi metodi:

- la richiesta può venire effettuata direttamente ad una pagina JSP, che grazie alle risorse messe a disposizione lato server, è in grado di elaborare i dati di ingresso per ottenere e restituire l'output voluto
- la richiesta può essere filtrata da una servlet che, dopo l'elaborazione dei dati, incapsula adeguatamente i risultati e richiama la pagina JSP, che produrrà l'output.

1.4.1 Componenti e oggetti impliciti delle JSP

Analizziamo ora i principali tag di una pagina JSP.

− *<% statement %>* all'interno di questo tag viene scritto codice, per lo più Java, che verrà interprato dalla Java Virtual Machine;

− *<%= espressione %>* all'interno del quale viene calcolata l'espressione indicata e ne viene restituito il valore;

− *<%! dichiarazione %>* all'interno di questo tag sono presenti frammenti di codi-ce che vengono riportati al di fuori del metodo principale di gestione della richie-sta, ovvero a livello di classe;

− *<%-- commento --%>* quello che è racchiuso da questo tag rappresenta un com-mento al codice;

− *<!-- commento -->* questo rappresenta un commento HTML. Nella pagina JSP, i commenti HTML vengono comunque processati e fanno parte del flusso di rispo-sta. Sarà quindi possibile scrive un commento dinamico. Questa possibilità è sfruttata nelle applicazioni per scrivere codice JavaScript, che è eseguito dal Client, in modo dinamico secondo lo stato dell'applicazione Web;

− *<jsp:useBean ...>* serve ad identificare (o eventualmente istanziare, in caso non esistesse) un oggetto per utilizzarlo nella pagina;

− *<jsp:setProperty ...>* permette di assegnare un valore ad una proprietà di un og-getto;

− *<jsp:getProperty ...>* serve ad ottenere il valore della proprietà di un oggetto;

− *<jsp:include ...>* esegue un'altra pagina JSP e ne include l'output;

− *<jsp:forward...>* inoltra il controllo di richiesta e risposta ad un'altra pagina JSP.

Sono inoltre presenti tre differenti tipi di direttive per la traduzione e la compilazione di una pagina JSP, in particolare.

• *page* che serve a specificare alcune caratteristiche della pagina JSP;

• *taglib* che serve ad importare una libreria di tag esterna (trattati in §2.8);

• *include* che serve ad importare, senza l'esecuzione preventiva, una porzione di codice di un file.

Queste direttive sono dichiarabili attraverso il tag *<%@ direttiva %>*. Un'importante caratteristica degli oggetti di una pagina JSP si chiama *Scope*. Lo Scope di un oggetto rappresenta la visibilità che questo ha all'interno dell'applicazione web. La seguente tabella riassume i possibili valori per lo Scope.

Scope	Visibilità	Ispezione
page	all'interno della stessa pagina	-
request	per tutta l'elaborazione della richiesta corrente	HttpServletRequest.getAttribute()
session	per tutto l'arco di una stessa sessione	HttpSession.getAttribute()
application	per tutta la vita dell'applicazione web	ServletContext.getAttribute()

Request è un oggetto implicito, quindi è disponibile per la stesura del codice senza particolari inclusioni, per usufruire delle sue funzionalità è quindi sufficiente usare la tipica sintassi nomeOggetto.nomeMetodo.

L'oggetto request, in particolare, permette di accedere alle informazioni d'intestazione specifiche del protocollo http. Al momento della richiesta questo metodo incapsula le informazioni sulla richiesta del client e le rende disponibili attraverso alcuni suoi metodi. L'uso più comune è quello di accedere ai parametri inviati (dati provenienti da un form) con il metodogetParameter("nomeParametro"), che restituisce una stringa con il valore del parametro specificato.

Response è anch'esso un oggetto implicito, e fornisce tutti gli strumenti per inviare i risultati dell'esecuzione della pagina jsp. Ad esempio:

- setBuffersize(int): imposta la dimensione in byte del buffer per il corpo della risposta, escluse quindi le intestazioni.

- setContentType("tipo"): imposta il tipo MINE per la risposta al client.
- flushBuffer(): forza l'invio dei dati contenuti nel buffer al client.

Ulteriori oggetti impliciti, a cui si fa solo un accenno, sono:

- Out: quest'oggetto ha principalmente la funzionalità di stampa dei contenuti. Con il metodo *print*(oggetto/variabile) è possibile stampare qualsiasi tipo di dato, come anche per *println()* che a differenza del precedente termina la riga andando a capo. Si capisce comunque che l'andare o meno a capo nella stampa dei contenuti serve solo a migliorare la leggibilità del codice HTML.
- PageContext: quest'oggetto rappresenta in pratica il contesto di esecuzione del servlet creato dalla pagina JSP. In pratica è poco utilizzato dai programmatori di pagine JSP.
- Config: permette di gestire tramite i suoi metodi lo startup del servlet associato alla pagina JSP, di accedere quindi a parametri di inizializzazione e di ottenere riferimenti e informazioni sul contesto di esecuzione del servlet stesso.
- Exception: quest'oggetto è accessibile solo dalle pagine di errore (dove la direttiva isErrorPage è impostata a true). Contiene le informazioni relative all'eccezione sollevata in una pagina in cui il file è stato specificato come pagina di errore. Il metodo principale è *getMessage()* che restituisce una stringa con la descrizione dell'errore.

Una delle funzionalità più richieste per un'applicazione Web è mantenere le informazioni di un utente lungo tutto il tempo della sua visita al sito. Questo problema è risolto dall'oggetto implicito **Session** che gestisce appunto le informazioni a livello di sessione, relative ad un singolo utente a partire dal suo ingresso alla sua uscita. È possibile, dunque, creare applicazioni che riconoscono l'utente nelle varie pagine del sito e che tengono traccia delle sue scelte e dei suoi dati. Le sessioni sono memorizzate sul server e i dati di sessione sono quindi riferiti e riservati ad un utente a cui viene creata un'istanza dell'oggetto Session e non possono essere utilizzati da sessioni di altri utenti. Per memorizzare i dati all'interno dell'oggetto session è sufficiente utilizzare il metodo

setAttribute specificando il nome dell'oggetto da memorizzare e una sua istanza. La lettura di una variabile di sessione, precedentemente memorizzata, è possibile grazie al metodo *getAttribute* che ha come unico ingresso il nome della variabile di sessione con cui avevamo memorizzato il dato che ci interessa reperire.

Session.getAttribute("nomeUtente") restituisce il nome dell'utente memorizzato come visto in precedenza; se non vengono trovate corrispondenza con il nome dato in ingresso, restituisce *null*.

Queste due sono le operazioni fondamentali e più utilizzate per la gestione della sessione.

Infine si analizza l'oggetto implicito **Application**. Esso consente di accedere alle costanti dell'applicazione e di memorizzare oggetti a livello di applicazione e quindi accessibili da qualsiasi utente per un tempo che va dall'avvio del motore JSP alla sua chiusura, in pratica fino allo spegnimento del server. Gli oggetti memorizzati nell'oggetto application sono visibili da ogni utente e ogni pagina può modificarli.

Per memorizzare i dati all'interno dell'oggetto application è sufficiente utilizzare il metodo *setAttribute* specificando il nome dell'oggetto da memorizzare e una sua istanza. Ad esempio, per memorizzare il numero di visite alla pagina è sufficiente fare: *application.setAttribute("visite", "0")*.

La lettura, invece, di un oggetto application precedentemente memorizzato è possibile grazie al metodo *getAttribute* che ha come unico ingresso il nome dell'oggetto application con cui avevamo memorizzato il dato che ci interessa reperire.

Il metodo *application.getAttribute("visite")*restituisce l'oggetto corrispondente, in questo caso semplicemente il valore 0. Se non vengono trovate corrispondenze con il nome viene restituito il valore *null*. Anche l'oggetto application come session possiede il metodo *getAttributeNames()* che restituisce un oggetto di tipo enumerativo di stringhe contenente i nomi di tutti gli oggetti memorizzati nell'applicazione in esecuzione. Per rimuovere un oggetto si utilizza il metodo *removeAttribute("nomeoggetto")*.Infine, per accedere alle costanti di applicazioni accennate all'inizio, segnaliamo due metodi: il primo molto utile è *getrealPath("")* che restituisce (con quella stringa in ingresso) il percorso

completo su cui è memorizzato il file. Il secondo metodo restituisce delle informazioni, solitamente riguardanti la versione del motore jsp che si sta utilizzando per l'esecuzione della pagina: *application.getServerInfo().*

Infine, di seguito è riportato un esempio di utilizzo dei tag all'interno di una pagina JSP, che contiene anche i costrutti di formattazione HTML.

```
<%@ page contentType="text/html ; charset=iso-8859-1" language="java" %>
<%@ page import="java.sql.*,java.util.*,beans.*" %>
<%@ tagliburi="es.tld" prefix="es" %>
<jsp:useBean id="utente" scope="session" type="User" />
<html>
<head>
<title>Examples</title>
 <head>
 <body> Test Page </body>
</html>
```

In questo esempio è definito il tipo di documento, detto "*contentType"*, e sono im-portate alcune classi java che possono essere utilizzate all'interno della pagina, con il la parola chiave "*import*". E' poi importata una libreria di tag personalizzata, "*es.tld"*, e si utilizza un oggetto di tipo *JavaBean* con scope *Session*.

1.4.2 JavaBean

Una pagina JSP compilata potrà risultare ostica sia ad un programmatore Java (che avrà difficoltà a curare i tag HTML) sia ad un grafico, che si ritroveranno tra i tag HTML, del codice incomprensibile. Soluzioni a questi problemi sono i JavaBean, ovvero i componenti che distinguono le JSP dalle altre tecnologie.

"JavaBeans components are Java classes that can be easily reused and composed together into applications."

Questa è la definizione iniziale fornita da Sun Microsystem (nelle specifiche J2EE) che lascia intendere come nella progettazione dei JavaBean, la riusabilità e la modularità siano concetti essenziali. I JavaBeans sono delle classi Java che devono rispettare le seguenti regole:

- avere un costruttore privo di argomenti o esserne addirittura privo, (quando una classe Java non dichiara esplicitamente un costruttore, le viene fornito automaticamente uno privo di argomenti).
- possono avere delle proprietà a cui è possibile accedere con i classici metodi:
  - `public void set`*`PropertyName`* `(`*`PropertyType`* `value)`: per impostare la proprietà;
  - `public PropertyType getPropertyName()`: per ottenere un valore.

I metodi di accesso a queste proprietà devono essere pubblici, ma una proprietà non è detto che debba per forza avere il metodo *set* ed il metodo *get*, per esempio potrebbe essere di sola lettura e quindi avere solo il metodo *get*. I JavaBean sono, dunque, delle classi Java, il cui codice sorgente sarà inserito in un file *nomeClasse.java*. Compilando tale file se ne crea uno nuovo di nome *nomeClasse.class*. Tutti i JavaBean compilati per poter essere utilizzati dovranno essere inseriti nella directory WEB-INF della nostra Web Application.

Nello specifico, la pagina jsp che vorrà utilizzare il JavaBean dovrà inserire al suo interno l'azione *jsp:useBean* che unita alle altre due azioni *jsp:setProperty* e *jsp:getProperty*, permette la gestione completa del componente. Le sintassi possibili sono due:

```
1. <jsp:useBean id="nome" scope="valore" opzioni />
2. <jsp:useBean id="nome" scope=" valore " opzioni >
     Codice di inizializzazione
   </jsp:useBean>
```

Gli attributi utilizzati per scrivere un javabean sono:

- **id**: utilizzato per assegnare un nome identificativo all'istanza del bean. Tale nome sarà utilizzato per accedere alle proprietà del componente all'interno della pagina JSP.
- **scope**: indica l'ambito di visibilità del bean. Esso può assumere i seguenti valori:
  - **page:** è il valore di default. Il bean esisterà solo all'interno della pagina che lo istanzia. Esso sarà distrutto ogni volta che l'utente si

sposta in una pagina successiva e ricreato ogni volta che la pagina verrà richiesta.

- **request:** il bean sarà visibile all'interno della richiesta, questo significa che si potrà accedere ad esso anche in pagine incluse dalle azioni jsp:include e jsp:forward.
- **session:** il bean sarà visibile all'interno di una stessa sessione utente.
- **application:** utilizzato per creare componenti comuni a tutta la web application, quindi a tutti gli utenti.

- **opzioni**: le opzioni sono attributi particolari che indicano delle modalità differenti di funzionamento dell'azione *jsp:useBean*. Sono possibili le seguenti combinazioni:
  - **class:** con questo attributo si specifica il nome della classe del bean da istanziare o, se il bean esiste già, richiamare un'istanza precedentemente creata. È l'attributo che di solito si utilizza.
  - **type**: indica il tipo della classe bean o anche il tipo dell'interfaccia implementata dal bean. Viene usato raramente e solo nei casi in cui si vuole accedere ad un bean già istanziato e del tipo specificato.
  - **beanName** insieme all'attributo type è utile in questi tre casi:
    - il bean non esiste, si crea una nuova istanza;
    - il bean esiste, si ottiene un riferimento all'oggetto già esistente;
    - il bean è stato serializzato, si caricano i dati salvati e si istanzia il bean.

Analizziamo, infine, come poter gestire le proprietà di un JavaBean:

- Per poter accedere alle proprietà di un bean si utilizza l'azione: `<jsp:getProperty name="IDnome" property="nomeProprietà"/>`, che equivale all' espressione: `<%=Idnome.getNomeProprietà()%>`.
- Per assegnare un determinato valore ad una proprietà si utilizza: `<jsp:setProperty name="IDnome" property="nomeProprietà" value="valore"/>`.

- Per assegnare ad una proprietà del bean il valore del parametro di una richiesta:
  `<jsp:setProperty name="IDnome"property="nomeProprietà" />`
- È necessario che il nome del parametro sia uguale al nome della proprietà. In caso di nomi diversi si utilizza la seguente sintassi:
  `<jsp:setProperty name="IDnome" property="nomeProprietà" param="nomeParametro"/>`
- Se si creano dei form con i nomi dei controlli uguali ai nomi delle proprietà del bean, esiste una forma abbreviata che permette di impostare tutte le proprietà del bean in un'unica azione:
  `<jsp:setProperty name="IDnome" property="*" />`
- Se si modifica il codice di una classe JavaBean è necessario riavviare Tomcat affinché le modifiche abbiano effetto.

Dunque, i **JavaBean** sono semplicemente classi scritte in linguaggio di programmazione Java secondo alcune particolari convenzioni, in merito ai nomi, alla costruzione e al comportamento dei metodi. Queste convenzioni rendono possibile avere tool che può usare, riusare, sostituire e connettere JavaBean.

1.4.3 JDBC

Un DBMS (DataBase Management System) è un sistema software progettato per consentire la creazione, la manipolazione e l'interrogazione di database. La maggioranza dei Database attualmente utilizzati sono dei database relazionali e L'SQL è il linguaggio tipicamente utilizzato per accedere a tali collezioni di dati strutturati. L'accesso ai dati tramite SQL, se è efficiente per il recupero, l'aggiornamento e/o la modifica dei dati interni al DB, non consente lo sviluppo di applicazioni che abbiano un adeguato livello d'interazione con l'utente, e che siano semplici da sviluppare per il programmatore. Gli attuali linguaggi di programmazione, ad esempio Java, offrono un insieme rilevante di API (Application Programming Interface), supportati da adeguati ambienti di sviluppo, che

consentono invece il rapido sviluppo di applicazioni che operano sui dati ed interagiscono con l'utente in modo semplice ed efficiente. All'interno di tali linguaggi, sono state previste delle API opportune che consentono l'interfacciamento verso i database relazionali, unendo così la possibilità di sviluppo di applicazioni evolute offerte dai linguaggi di alto livello, con la capacità di gestione e recupero di dati persistenti offerti dai DBMS.

In particolare, il linguaggio di programmazione Java offre l'interfaccia JDBC che consente allo sviluppatore di applicazioni l'accesso ad ogni tipo di dato memorizzato all'interno di un DBMS relazionale.

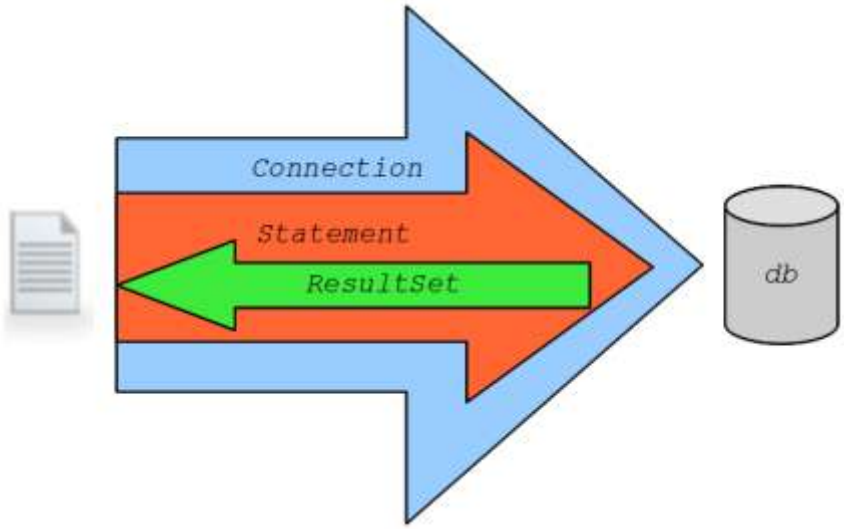
JDBC (Java Database Connectivity) è, dunque, un'interfaccia completamente Java utilizzata per eseguire istruzioni SQL. L'implementazione più utilizzata di questa interfaccia è quella detta *"bridge JDBC – ODBC"*. In tal caso, il driver JDBC funge da ponte per accedere al database attraverso il driver ODBC, che deve essere presente e configurato sul server.

L'architettura di JDBC, così come quella di ODBC, prevede l'utilizzo di un "driver manager", che espone alle applicazioni un insieme di interfacce standard e si occupa di caricare a "run-time" i driver opportuni per "pilotare" gli specifici DBMS. Le applicazioni Java utilizzano le "JDBC API" per parlare con il JDBC driver manager, mentre il driver manager usa le JDBC driver API per parlare con i singoli driver che pilotano i DBMS specifici.

Le classi implementative sono normalmente contenute in pacchetti *.jar* che lo sviluppatore dovrà caricare nella propria applicazione. Successivamente, per poter utilizzare i driver, si dovrà aprire una connessione con il database, creare uno *Statement* per interrogare la base dati e gestire i risultati ottenuti. Di seguito è riportata una porzione di codice che potrebbe popolare una pagina JSP, in cui è compiuta l'interrogazione di una base dati connessa tramite driver JDBC-ODBC.

```
<%@ page import="java.sql.*", java.util.*" %>
<%
// Viene caricato il driver
String driver = "sun.jdbc.odbc.JdbcOdbcDriver";
Class.forName(driver);
String url = "jdbc:odbc:myDataSource";
//Viene creata una connessione con username e password Connection
connection = DriverManager.getConnection(url, "myUser", "myPassword");
//Viene creato un oggetto Statement Statement std
connection.createStatement();
//Viene eseguita un'interrogazione e raccolti i risultati in un oggetto
ResultSet String query = "SELECT NomeColonna1, NomeColonna2 FROM
myTable"; ResultSet res = std.executeQuery(query); %>
<table>
<%
//Vengono stampati i risultati inseriti in una tabella
while (res.next()) {
%>
<tr>
  <td><%= res.getString("NomeColonna1"); %></td>
  <td><%= res.getString("NomeColonna2"); %></td>
</tr>
<%}
//Chiusura di tutti i flussi
res.close();
std.close();
connection.close();
%>
```

L'oggetto di tipo *Connection* consente di ottenere una connessione con un database. La classe *DriverManager* è incaricata di scegliere l'opportuno driver registrato, at-traversi l'URL e le credenziali di accesso al database passati come parametri. L'oggetto *Statement*, invece, permette di effettuare interrogazioni alla base dati in linguaggio SQL, attraverso la connessione precedentemente creata. Qualora lo stesso *Statement* SQL sia utilizzato più volte, è consigliato servirsi, in alternativa, dell'oggetto *PreparedStatement* che aumenta l'efficienza dell'esecuzione della query e l'allocazione delle risorse. Infine, il *ResultSet*, è l'oggetto che raccoglie i dati ri-chiesti all'interno dello *Statement* e li inserisce in una struttura dati opportuna. Il me-todo *getString("NomeColonna")* restituisce il valore correntemente puntato dal cur-sore *next()* della colonna specificata come parametro. Se il nome della colonna non è presente nel ResultSet, non viene lanciata alcuna eccezione. Esistono diversi metodi *getType("NomeColonna")*, dove *Type* indica il tipo di dato che si vuole estrarre dal *ResultSet* e che deve essere concorde al tipo di dato di *"NomeColonna"*.



F

Capitolo 2: Il Pattern MVC

2.1 L'architettura software "a livelli"

Nello sviluppo delle applicazioni software è possibile descrivere l'architettura del sistema utilizzando uno fra i molteplici paradigmi messi a disposizione, ma in linea generale, trova una maggiore applicazione la nota architettura "a livelli" (**Layered Application Architecture**). Quest'ultima prevede che un sistema software sia decomposto in tre livelli nettamente distinti, che comunque abbiano la possibilità di comunicare fra loro secondo un'opportuna gerarchia. Ciascuno dei tre livelli ha un proprio ruolo ed assolve ad uno specifico compito all'interno del sistema complessivo, senza interferire con gli altri livelli, ma scambiando con essi le informazioni necessarie all'esecuzione di elaborazione anche molte complesse. I tre livelli in questione sono i seguenti:

- *Presentation layer*: è il livello di presentazione, il cui compito è di interagire direttamente con l'utente del sistema, acquisire i dati di input immessi da quest'ultimo e visualizzare i risultati dell'elaborazione effettuata del sistema stesso. Esso, in pratica, definisce la GUI (Graphic User Interface) ossia l'interfaccia grafica dell'applicazione.
- *Application processing layer*: è il livello in corrispondenza del quale si trova la "business-logic" dell'applicazione e quindi tutti i moduli software che implementano le funzionalità che il sistema mette a disposizione. In sostanza, è il centro dell'elaborazione dei dati in cui avvengono tutte le computazioni;

- *Data management layer*: è il livello che si occupa della gestione della persistenza e dell'accesso ai dati, per cui è tipicamente caratterizzato da un DBMS (Database Management System);

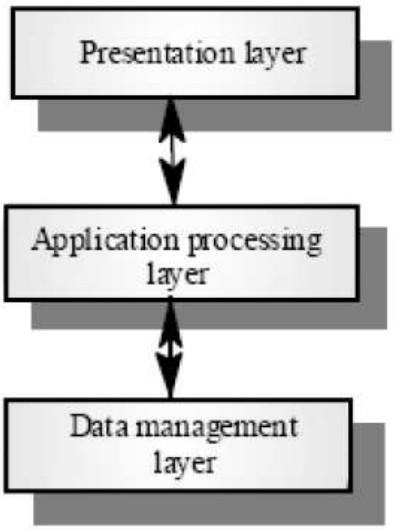


Figura 7. Architettura software "a livelli"

Sviluppando un'applicazione secondo questa architettura, ogni livello è indipendente dagli altri, per cui la modifica di uno di essi non ha effetto sui restanti. Tuttavia, è prevista la comunicazione fra loro e lo scambio di informazioni. Un tipico scenario di funzionamento del sistema può essere il seguente: un utente utilizza l'applicazione, interagendo direttamente con la GUI e fornisce quindi al Presentation layer,i dati su cui andrà eseguita l'elaborazione. Il Presentation layer, una volta acquisiti i dati di input, li trasferisce all'Application processing layer che esegue su di essi una determinata computazione. Durante l'elaborazione, la business-logic può prevedere la memorizzazione persistente dei dati oppure la necessità di acquisire ulteriori dati già memorizzati. In questo caso, c'è l'interazione con il Data management layer, il quale memorizza i dati che gli sono passati dal livello superiore, oppure recupera da un Data Source i dati richiesti e li trasmette alla business-logic. Al termine dell'elaborazione i risultati vengono passati al Presentation layer che li visualizza in una certa forma all'utente finale.

Facendo riferimento al paradigma Client-Server, notevolmente utilizzato nelle web-application, ma di gran di richiamo anche per le applicazioni desktop, i tre livelli del sistema devono essere correttamente ripartiti anche da un punto di vista hardware. Le principali architetture per la ripartizione sono:

- Two-Tier
- Three-Tier

L'architettura *Two-Tier* prevede un unico Client ed un unico Server ed i tre livelli dell'applicazione software sono distribuiti fra di essi secondo due possibili modalità:

- Thin Client: sul Client risiede il Presentation layer mentre sul Server gli altri due livelli (Application processing layer e Data management layer). Un vantaggio può risiedere nel fatto che una modifica alla business-logic va eseguita una sola volta sul Server, mentre lo svantaggio principale può essere caratterizzato dall'enorme carico di lavoro che deve supportare il Server stesso, dato il numero elevato di Client che possono accedere ad esso.
- Fat Client: sul Client risiedono i primi due livelli (Presentation layer e Application processing layer), mentre sul Server soltanto il Data Management layer. Il vantaggio è di ridurre il carico di lavoro sul Server che si occupa solo dell'accesso ai dati, delegando l'elaborazione degli stessi al Client. Lo svantaggio principale è la complessità maggiore dei Client e quindi la necessità di aggiornare ciascuno di essi nel caso in cui vengano apportate modifiche alla business-logic.

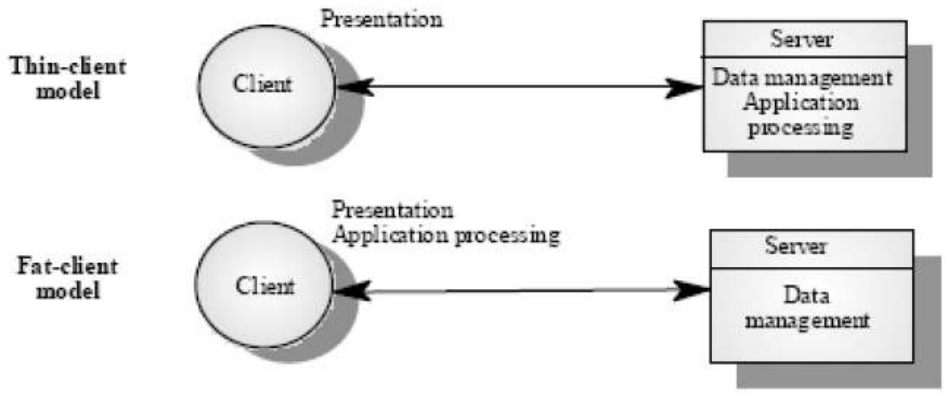


Figura 8. Architettura Two-Tier

L'architettura *Three-Tier*, maggiormente utilizzata, prevede la presenza di un unico Client ed una coppia di Server. Sul Client risiede il Presentation layer e su ciascuno dei due Server sono distribuiti i due restanti livelli (Application processing layer e Data management layer). Nell'ambito di una web application, il Client è caratterizzato da un nodo della rete sul quale è in esecuzione il browser, mentre i due Server, da un punto di vista software, sono tipicamente inglobati in un unico nodo della rete, che funge da Server fisico. In particolare, sulla stessa macchina sono in esecuzione il Web Server associato all'Application processing layer ed il Database Server associato al Data management layer.

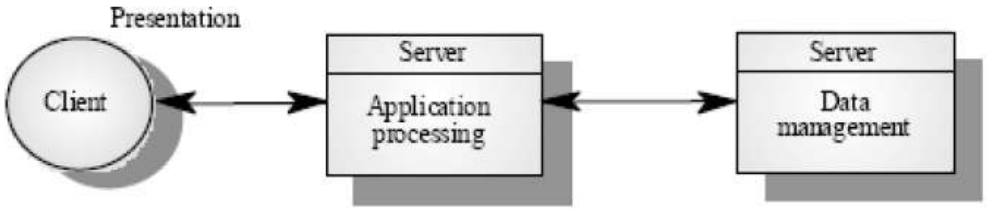


Figura 9. Architettura Three-Tier

In conclusione, è proprio su questa particolare architettura "a livelli" che si basa il **pattern MVC** (***Model-View-Controller***), che rappresenta il fondamento dei framework Struts e Spring, quest'ultimo sarà oggetto di un approfondito studio.

2.2 L'importanza dei "design patterns"

Quando si tenta di riutilizzare i componenti eseguibili, si è inevitabilmente compromessi dalle decisioni progettuali fatte dagli implementatori di tali componenti. Si va da algoritmi particolari che sono stati utilizzati per implementare i componenti, agli oggetti e ai tipi presenti nelle interfacce dei componenti.

Se una decisione progettuale, che si è stabilita di utilizzare, entra in contrasto con le esigenze del caso cui si sta lavorando, il riutilizzo del componente è impossibile oppure il componente potrà presentare inefficienze all' interno del sistema. Un modo per aggirare questo tipo di problemi è di riutilizzare i design astratti che non includono alcun dettaglio

implementativo. Sarà possibile, successivamente, implementare tali design per soddisfare le specifiche esigenze applicative. Il primo esempio di questo approccio al riuso si è avuto nella documentazione e pubblicazione di algoritmi fondamentali (Knuth , 1971) e, più tardi, nella documentazione di tipi di dati astratti come pile , alberi e liste (Booch, 1987). Più di recente, quest'approccio al riuso è stato incarnato in modelli di progettazione. Fu solo alla metà del ventesimo secolo, che l'architetto Christopher Alexander, osservando come i suoi colleghi tendevano a risolvere i medesimi problemi più o meno allo stesso modo, introdusse il concetto dei "design patterns".

"Un design pattern descrive un problema che si presenta frequentemente nel nostro ambiente, e quindi descrive il nucleo della soluzione in modo tale che sia possibile impiegare tale soluzione milioni di volte, senza peraltro produrre due volte la stessa realizzazione."

Fu questa la descrizione dei design patterns data da Christopher Alexander. Ovviamente, tale definizione era riferita all'ambito architetturale, ma nel 1994 Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, pubblicarono il libro "Design Patterns: Element of Reusable Object-Oriented Software", in cui applicarono l'intuizione di Alexander allo sviluppo del software.

Un pattern è, dunque, un modello che permette di definire la soluzione di un problema specifico che si ripresenta di volta in volta, in un contesto diverso. Presenta inoltre le seguenti caratteristiche:

- *Nome*: individua il pattern e la sua importanza non è secondaria, in quanto rientra a far parte del vocabolario dello sviluppo software
- *Descrizione*: spiega quando il pattern può essere applicato.
- *Soluzione*: descrive gli artefatti software per risolvere il problema, come ad esempio gli elementi che rientrano nello sviluppo, le classi e le relazioni fra esse, le associazioni e i ruoli, le modalità di collaborazione tra le classi coinvolte ed infine la distribuzione delle responsabilità nella soluzione del particolare problema di design considerato.

- *Conseguenze*: descrivono i risultati che si possono ottenere dall'applicazione del pattern per spingere uno sviluppatore a farne uso.

Questi sono gli elementi essenziali di una descrizione del modello che possono essere anche suddivisi. Ad esempio, Gamma ed i suoi coautori suddividono la descrizione del problema in motivazione (descrizione del perché il modello è utile) e applicabilità (descrizione di situazioni in cui può essere utilizzato il modello). Per quanto riguarda la descrizione della soluzione, essa la struttura del modello, i partecipanti, le collaborazioni e la realizzazione. Le rappresentazioni grafiche sono normalmente utilizzate per illustrare le classi di oggetti che vengono utilizzate nei modelli e nelle loro relazioni. Esse completano la descrizione del modello e aggiungano dettagli alla descrizione della soluzione. Un numero enorme di modelli pubblicati sono ora disponibili che coprono una vasta gamma di domini applicativi e linguaggi. La nozione di un modello come concetto riutilizzabile è stato sviluppato in una serie di aree da parte della progettazione software, compresa la gestione della configurazione, la progettazione dell'interfaccia utente e l'interazione tra gli scenari.

L'uso dei pattern è una forma efficace di riutilizzo. Tuttavia, solo gli ingegneri del software esperti che hanno una profonda conoscenza dei modelli li possono utilizzare in modo efficace. Questi sviluppatori possono riconoscere le situazioni generiche in cui può essere applicato un pattern. I programmatori inesperti troveranno sempre difficile decidere se possono riutilizzare un modello o la necessità di sviluppare una soluzione per un fine particolare.

Al giorno d'oggi, infatti, i pattern sono diventati di grande interesse in virtù del fatto che rappresentano un' evoluzione della programmazione Object Oriented, poiché nelle soluzioni proposte vengono presentate delle classi o delle interfacce con precise responsabilità, e sono combinate classi ed oggetti atomici per fornire una base più ampia per la risoluzione di un problema, nella fattispecie per lo sviluppo di un'applicazione software. Un design pattern fornisce, infatti, allo sviluppatore:

- una soluzione codificata e consolidata per un problema ricorrente;

- un'astrazione di granularità e livello di astrazione più elevati di una classe;
- un supporto alla comunicazione delle caratteristiche del progetto;
- un modo per progettare software con caratteristiche predefinite;
- un supporto alla progettazione di sistemi complessi;
- un modo per gestire la complessità del software

Un ulteriore obiettivo, spesso perseguito, è proprio l'indipendenza della soluzione rispetto alla sua implementazione in modo tale da permettere un effettivo riutilizzo del codice.

Infine, si può concludere affermando che, può essere considerato un "buon" pattern un pattern che descrive una soluzione "assodata" per un problema "ricorrente" in un contesto "specifico". Naturalmente, numerose sono le tipologie di design patterns, ma nell'ambito dello sviluppo delle web application, in riferimento all'obiettivo proposto in questa tesi, assume un ruolo rilevante il *pattern MVC* (*Model-View-Controller*).

2.3 Il Design Pattern MVC

Il design pattern MVC ha le sue origini nell'ambiente Smalltalk, in cui era utilizzato per la realizzazione della GUI (Graphic User Interface) di applicazioni desktop e non orientate al web. Tale pattern si basa sull'idea di separare i dati dalla presentazione, poiché mantenere un forte accoppiamento tra essi e comporta che la modifica dell'uno implica automaticamente un aggiornamento dell'altro.

Si immagini di scrivere una applicazione consistente in una singola pagina Web, che prelevi i dati da un database, li elabori e li restituisca al client. È evidente che riunire tutte queste operazioni in un unico blocco di codice (la pagina) creerà presto molta confusione, oltre a portare problemi di manutenzione e infinite sessioni di debug per risolvere i problemi. Si provi, allora, ad organizzare il codice in modo più logico, dividendolo in tre parti: la prima si occuperà dei dati e fornirà quindi i metodi per accedere al database, la seconda sarà responsabile della creazione del codice html, mentre la terza farà da intermediario fra le prime due.

Lo sviluppatore, organizzando il codice secondo questo schema, potrà concentrarsi su un problema specifico ed avere la sicurezza che l'intervento rimanga circoscritto al blocco di codice di cui si sta occupando, lasciando intatti gli altri. Pensando, poi, ad un progetto di grandi dimensioni, in cui presumibilmente ogni parte sarà creata e mantenuta da persone diverse, diventa evidente come la divisione logica del codice in zone distinte aumenti l'efficienza complessiva.

Lo schema così identificato è esattamente quello proposto dal **pattern MVC**. Esso, quindi, prevede che un sistema software sia realizzato secondo l'architettura "a livelli", stabilendo un disaccoppiamento fra i dati e le rappresentazioni, mediante la elementi: il Model, la View e il Controller.

Il ***Model*** è il responsabile della gestione dei dati e del comportamento dell'applicazione (data e behaviour). Esso coordina la business-logic dell'applicazione, l'accesso alle basi di dati e tutte le parti critiche "nascoste"a del sistema. Incapsula lo stato dell'applicazione ed espone le funzionalità di quest'ultima. È indipendente dalle specifiche rappresentazioni dei dati sullo schermo e dalle modalità di input dei dati stessi da parte dell'utente. Ad esso fanno riferimento l'Application processing layer ed il Data management layer nel design del software "a livelli".

Il Model può essere scomposto in tre sottolivelli puramente concettuali:

- *External interface*. Esso è caratterizzato dal codice che definisce un'interfaccia mediante la quale il codice esterno comunica con il Model. Generalmente il codice esterno è determinato dal framework adottato per sviluppare la web application, come ad esempio Struts e JavaServer Faces.
- *Business logic*. Rappresenta il cuore del Model contenente il codice che realizza le funzionalità dell'applicazione.
- *Data access*. È costituito dal codice che permette di accedere ad un datasource, come ad esempio una base di dati.

I tre sottolivelli descritti non rappresentano necessariamente dei set di classi separate, ma, dei set di responsabilità differenti che ha il Model. Lo sviluppatore può decidere di

realizzare i tre sottolivelli mediante una o più classi e raggruppando due o più sottolivelli. Tipicamente, si preferisce realizzare il sottolivello Data Access con una o più classi che hanno come unico scopo quello di permettere l'accesso ad una base di dati. Gli altri due sottolivelli possono essere realizzati in due modi:

- separazione: ci sono una serie di classi che realizzano la Business logic ed ulteriori classi che definiscono l' External interface;
- fusione: sono definite una serie di classi che definiscono la Business logic e contengono all' interno anche le funzionalità di interfacciamento dell' External interface.

La scelta può dipendere dalla complessità dell'applicazione e, eventualmente, dalla tecnologia che è adottata sul Model per realizzare il web applicativo.

La ***View*** ha il compito di visualizzare i dati e presentarli all'utente anche in forme diverse, sul dispositivo utilizzato per accedere al sistema. Ciò vuol dire che, pur partendo dagli stessi dati, è possibile effettuare "rendering" diversi ed ottenere viste multiple dello stesso modello. Ad esso fa riferimento il Presentation layer.

Il ***Controller*** definisce il meccanismo mediante il quale il Model e la View comunicano. Realizza la connessione logica tra l'interazione dell'utente con l'interfaccia applicativa e i servizi della business-logic nel back-end del sistema. È responsabile della scelta di una tra molteplici viste dello stesso modello, in base al tipo di dispositivo utilizzato dall'utente per accedere al sistema, ma anche in relazione alla localizzazione geografica dell'utente stesso. Una qualsiasi richiesta (request) fatta dal sistema è acquisita dal Controller, che individua all'interno del Model il gestore della richiesta (request handler). Ottenuto il risultato dell'elaborazione (response), il Controller stesso determina a quale View passare i dati per la presentazione degli stessi all'utente.

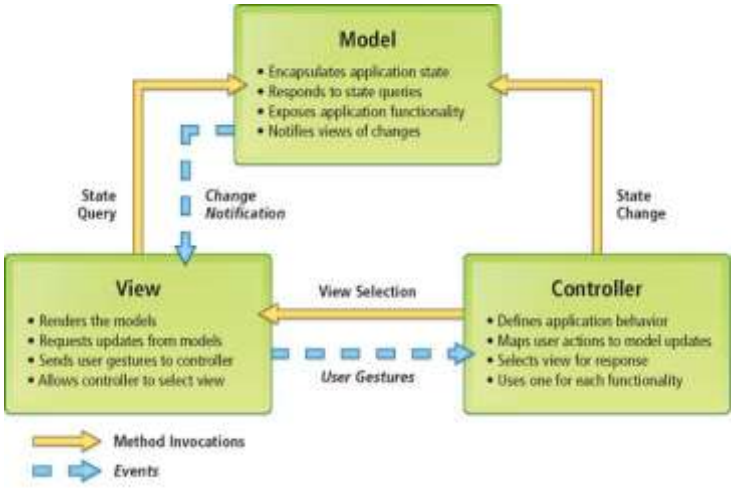


Figura 10. Model, View e Controller

Il principale vantaggio che scaturisce dall'utilizzo di quest'architettura è che la business-logic definita all'interno del Model è separata dal Presentation layer, che si trova all'interno della View. Tutto ciò favorisce il riuso dei componenti e la possibilità di apportare delle modifiche ad un livello senza avere degli effetti sull'altro.

2.4 L'evoluzione dell'architettura delle Web Applications

Il linguaggio Java è considerato uno dei migliori linguaggi per lo sviluppo delle applicazioni Web, attraverso l'uso delle Servlets e delle pagine JSP, l'architettura delle Web Applications ha subito una notevole evoluzione nel corso degli anni, seguendo un iter di questo tipo:

1. Assenza del pattern MVC
2. Utilizzo del pattern MVC secondo il Model 1
3. Utilizzo del pattern MVC secondo il Model 2
4. Web Application Framework
5. Web Application Framework basato su uno standard

Tal evoluzione ha previsto un aumento della complessità e della robustezza di ciascuna applicazione e può essere schematizzata nel seguente modo, fino all'introduzione del Model 1.

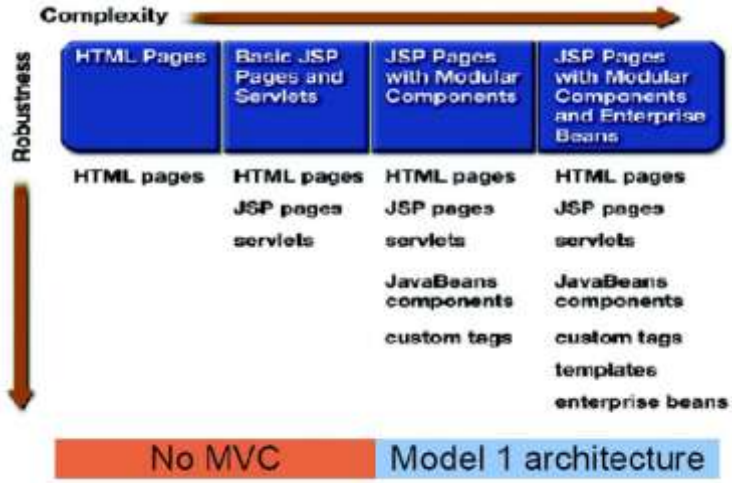


Figura 11. Evoluzione dello sviluppo di Web Application

Il *Model 1* del pattern MVC è detto Page-Centric, poiché l'architettura della Web application è basata sulle pagine JSP. Il sistema complessivo è composto da una serie di pagine JSP legate tra loro, ciascuna delle quali gestisce tutti gli aspetti principali dell'applicazione tra cui la presentazione, il controllo ed i processi di business. È evidente che la business-logic ed il controllo sono fortemente accoppiati all'interno di ciascuna pagina JSP, attraverso l'utilizzo dei JavaBeans, Scriptlets ed Espressioni. I tre elementi del pattern, il Model, la View ed il Controller, pur essendo distinti, sono inglobati all'interno di una stessa struttura, che in questo caso è rappresentata da una pagina JSP.

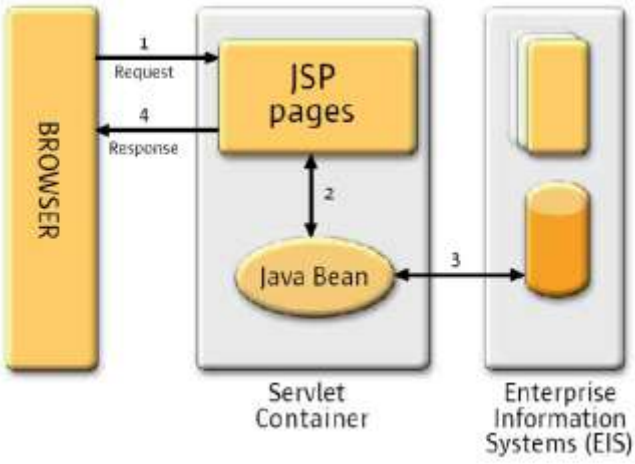


Figura 12. MVC Model 1

Il modello prevede uno scenario di interazione di questo tipo: il browser invia una richiesta (request) di una risorsa al Web Server, nella maggior parte dei cadi per visualizzazione di una pagina JSP. Il Web Server, tipicamente, funge da Web Container e quindi da Servlet Container in quanto va ricordato che una pagina JSP, una volta richiesta, viene sempre trasformata in una corrispondente servlet. All'interno della pagina JSP, ci sono i tag che ne definiscono la presentazione all'utente e quindi l'aspetto grafico, ma anche gli elementi per l'esecuzione delle elaborazioni. Queste ultime possono essere eseguite all'interno della pagina stessa, attraverso il codice Java immerso nei tag oppure mediante dei componenti esterni, ai quali si fa riferimento da tale pagina. I componenti in questione sono tipicamente dei JavaBeans, i quali effettuano una qualsiasi computazione, comunicando eventualmente con il back-end del sistema, ad esempio per l'accesso a basi di dati. Il risultato di ciascuna elaborazione sarà così integrato all'interno della pagina HTML prodotta, che sarà inviata nella risposta (response) al browser. Da quanto detto, si evince che il Model, la View e il Controller sono in pratica integrati all'interno di ciascuna pagina JSP e che non c'è una netta separazione fra essi. I principali limiti di questo modello sono:

- è incoraggiata una struttura a "spaghetti" delle pagine JSP, poiché la business-logic si "perde" all'interno di ciascuna pagina e la navigazione nella Web application complessiva viene fatta pagina per pagina;
- è difficile eseguire il debug, poiché tutti gli errori riportati dal Web Container fanno riferimento al codice compilato della pagina JSP in una Servlet e quindi sono difficili da individuare all'interno della pagina stessa.

Tutto ciò rappresenta un primo passo verso il modello definitivo del pattern MVC.

Il *Model 2* del pattern MVC, invece, è detto Servlet-Centric, poiché l'architettura della Web application si basa fondamentalmente sull'utilizzo di una Servlet. Il sistema complessivo è composto, infatti, da una Servlet principale che svolge il ruolo di Controller, da una serie di pagine JSP che rappresentano la View ed infine da un insieme di JavaBeans che costituiscono il Model. I tre elementi del pattern MVC sono, quindi,

nettamente separati tra loro, pur mantenendo la possibilità di comunicare per scambiarsi informazioni. In particolare, le pagine JSP si occupano esclusivamente della presentazione dei dati dell'utente, senza contenere un minimo di business-logic. La Servlet "master" ha il compito di acquisire tutte le richieste provenienti dalla rete e funge da "dispatcher", inoltrando ciascuna di esse verso il corrispondente "handler" per permetterne la gestione. Al termine dell'elaborazione, è la stessa Servlet che determina a quale pagina JSP restituire il controllo, eseguendo il "redirecting". Infine, i JavaBeans incapsulano le funzionalità dell'applicazione, interagiscono con il back-end del sistema ed eseguono tutte le elaborazioni richieste.

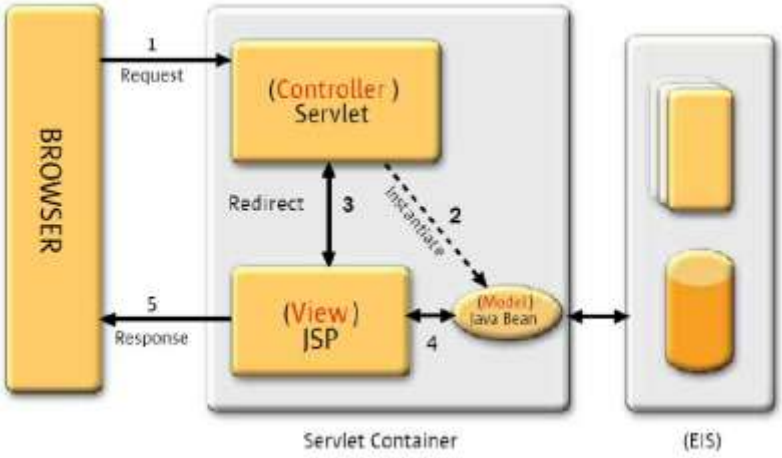


Figura 13. MVC Model 2

Il modello prevede uno scenario di interazione di questo tipo: il browser invia una richiesta (*request*) di una risorsa al Web Server, nella maggior parte dei casi per la visualizzazione di una pagina JSP. Il Web Server, tipicamente funge da Web Container e quindi da Servlet Container, in quanto è in esecuzione su di esso, la "master" Servlet della Web application. Quest'ultima funge da Controller ed acquisisce la richiesta sulla base della quale individua l' handler che dovrà gestirla. In particolare, vengono istanziati una serie di JavaBeans, costituenti il Model, che eseguono le elaborazioni richieste ed eventualmente interagiscono con il back-end del sistema, accedendo ad una base di dati. Al termine della computazione, il controllo ritorna alla Servlet che, sulla base del risultato,

determina la View e quindi la pagina JSP verso la quale eseguire il redirecting, la quale estrae dai JavaBeans i risultati e li mostra all'utente. Infine, la pagina html prodotta viene trasmessa al browser come risposta (*response*) definitiva.

Da quanto detto, si evince che il Model, la View e il Controller sono nettamente separati tra di loro e che il Controller funge da tramite per la comunicazione tra i primi due. Da ciò scaturisce anche che il debug può essere eseguito in maniera piuttosto semplice su ciascun JavaBeans, estrapolandolo dal sistema complessivo. Per quanto concerne la struttura del Controller, è possibile pensare di utilizzare una sola o anche più Servlets. La scelta dipende dal livello di granularità della Web application ed è possibile pensare a tre diverse soluzioni:

- una sola "master" Servlet;
- una Servlet per ciascun caso d'uso dell'applicazione oppure per ogni macrofunzionalità offerta;
- combinazione delle due precedenti soluzioni: una "master" Servlet per gestire le funzioni comuni a tutti gli ambiti dell'applicazione, che delega alle Servlets "figlie" la gestione di particolari macrofunzionalità del sistema.

Capitolo 3: Spring

"Spring è un framework open source nato con l'intento di gestire la complessità nello sviluppo di applicazioni enterprise."

È forse questa la definizione più calzante di Spring, data dagli stessi autori. Tale definizione, però, suscita alcuni interrogativi in chi conosce già il mondo Java e si avvicina a Spring per la prima volta, il primo tra i quali è: «perché imparare questo framework e preferirlo alla miriade di quelli già esistenti ed affermati?» La domanda, che inizialmente può apparire banale, offre, invece, una chiave per illustrare in pieno i vantaggi offerti da Spring, che se pur sviluppato dopo molti altri framework si contraddistingue come uno dei migliori e completi. La risposta alla domanda sopra posta, in realtà non è semplice e in parte può essere riassunta nei punti seguenti:

- *Spring è un framework "leggero"*: grazie alla sua architettura estremamente modulare é possibile utilizzarlo nella sua interezza o solo in parte. L'adozione di Spring in un progetto è molto semplice, può avvenire in maniera incrementale e non ne sconvolge l'architettura esistente. Questa sua peculiarità ne permette anche una facile integrazione con altri framework esistenti, come ad esempio Struts.
- *Spring è un lightweight container*: si propone come alternativa/complemento a J2EE. A differenza di quest'ultimo, Spring propone un modello più semplice e

leggero (soprattutto rispetto ad EJB) per lo sviluppo di entità di business. Tale semplicità è rafforzata dall'utilizzo di tecnologie come l'Inversion of Control e l'Aspect Oriented che danno maggiore spessore al framework e favoriscono la focalizzazione dello sviluppatore sulla logica applicativa essenziale.

- *Spring è un framework nato con la concezione che il codice di qualità debba essere facilmente testato*. Questa filosofia fa sì che, con Spring, sia molto facile testare il codice e, grazie a questa peculiarità, il framework si è ritagliato uno spazio importante in quegli ambiti dove il testing è considerato parte fondamentale del progetto software.

A differenza di molti framework che si concentrano maggiormente nel fornire soluzioni a problemi specifici, Spring mette a disposizione una serie completa di strumenti atti a gestire l'intera complessità di un progetto software. Analizzeremo in dettaglio gli strumenti offerti da questo framework, per ora è sufficiente affermare che Spring fornisce un approccio semplificato alla maggior parte dei problemi ricorrenti nello sviluppo software (accesso al database, gestione delle dipendenze, testing, etc.).

Spring è un framework open source per lo sviluppo di applicazioni Java. Permette di costruire applicazioni utilizzando "plain old java object" (POJO) applicandovi i servizi enterprise in maniera non invasiva secondo il modello di programmazione di JAVA SE ed EE. Parlando in generale Spring si propone di demandare alcune responsabilità dello sviluppatore al framework sia offrendo dei package che permettono un'integrazione semplice delle funzionalità più diffuse nel mondo enterprise (ad esempio i package DAO, ORM, WEB), sia promuovendo l'uso di Best Practice come la programmazione verso interfacce piuttosto che verso classi, mantenendo bassa la dipendenza dell'applicazione dalle API del framework e spostando la configurazione dei vari componenti in file di contesto XML. Quest'ultima impostazione insieme all'uso dell'Inversion of Control permette di evitare la proliferazione di classi singleton, che complicano i test, mantenendo semplice la manutenzione della configurazione dei vari aspetti dell'applicazione.

Spring nasce nel 2002 come allegato al libro Expert One-on-One J2EE Design and Development di Rod Johnson ed è ad oggi un framework maturo e molto diffuso nella comunità Java. Il libro di Johnson promuove un modello di sviluppo semplice in antitesi a framework invasivi, come gli EJB, secondo il motto "J2EE should be easier to use". A differenza di questi Spring si basa sull'utilizzo di POJO senza obbligare le classi a seguire dei contratti spesso inutilmente complessi.

Spring è basato su un'architettura a strati ed offre una integrazione semplice con le tecnologie più diffuse. Questo permette al progettista di scegliere quali moduli utilizzare integrandoli con altri prodotti senza essere vincolato a utilizzare il framework nella sua interezza. Inoltre ogni modulo è stato disegnato tenendo a mente la scrittura di unit test. Per quanto ritenuti una best practice, spesso non sono eseguiti, perché comportano, di fatto, una riscrittura del codice. Spring invece, grazie alla programmazione tramite interfacce e all'IoC (Inversion of Control) rende questa operazione semplice e più circoscritta, di fatto comportando solo la modifica di pochi file di configurazione e non del codice vero e proprio.

3.1 L'architettura di Spring

Nello schema seguente è mostrata la struttura del framework.

Figura 14. I componenti di Spring

Tra i moduli che lo compongono, possiamo trovare:

- **Inversion of Control Container**: un componente che gestisce la creazione e la risoluzione delle dipendenze di un Bean utilizzando il pattern IoC. Spring è un

"lightweight container", perchè i Bean non devono aderire a nessun tipo di contratto2 e possono essere rappresentati da una qualsiasi classe Java

- **Programmazione Aspect-Oriented**: un paradigma di programmazione basato sulla creazione di entità software che sovrintendono alle interazioni fra oggetti per realizzare funzionalità che coinvolgono più ambiti e che quindi provocherebbero una duplicazione del codice negli oggetti coinvolti.
- **Data Access**: tramite le API JDBC e strumenti di object-relational mapping è possibile utilizzare i più diffusi database relazionali e NoSQL
- **Controllo delle transazioni**: unifica diverse API per la gestione delle transazioni.
- **Model-View-Controller**: implementazione del pattern MVC basato sul protocollo HTTP e sull'uso delle servlet.
- **Convention-over-configuration**: un paradigma di programmazione che prevede una configurazione esplicita da parte del programmatore solo per quegli aspetti che si differenziano dalle implementazioni standard.
- **Batch Processing**: un framework per l'esecuzione di processi batch di grosse dimensioni che comprende funzioni per il logging, tracing, gestione delle transazioni e il controllo della schedulazione.
- **Sicurezza**: un insieme di processi per il supporto all'autenticazione, autorizzazione e a vari protocolli e standard per la gestione della sicurezza.
- **Remote Management**: configurazione e controllo di oggetti Java locali e remoti tramite JMX.
- **Messaging**: una modalità di invio di messaggi tramite le API standard di JMS che permettte di registrare degli oggetti listener per un utilizzo trasparente del consumo dei messaggi tramite message queues.
- **Testing**: classi di supporto per la scrittura di unit e integration test.

Nel seguito sono approfonditi i due aspetti del framework Spring che sono stati più rilevanti per il progetto svolto, ovvero l'inversione di controllo e il Model View Controller.

3.2 Inversion of Control (IoC)

Per comprendere a fondo le potenzialità di Spring bisogna prima introdurre i concetti di Inversion of Control (IoC) e Dependency Injection (DI), che pur significando due cose diverse, spesso sono utilizzati come sinonimi provocando non poca confusione.

L'Inversion of Control è un principio architetturale nato alla fine degli anni ottanta, basato sul concetto di invertire il controllo del flusso di sistema (Control Flow) rispetto alla programmazione tradizionale. Questo principio è molto utilizzato nei framework e ne rappresenta una delle caratteristiche basilari che li distingue dalle API.

Nella programmazione tradizionale la logica di tale flusso è definita esplicitamente dallo sviluppatore, che si occupa tra le altre cose di tutte le operazioni di creazione, di inizializzazione ed di invocazione dei metodi degli oggetti. IoC invece inverte il control flow facendo in modo che non sia più lo sviluppatore a doversi preoccupare di questi aspetti, ma sia il framework, che reagendo a qualche "stimolo" se ne occuperà per suo conto. Questo principio è anche conosciuto come Hollywood Principle ("Non chiamarci, ti chiameremo noi").

Il pattern di design Inversion of Control ha, quindi, come fine quello di produrre un codice fortemente disaccoppiato, leggero e adatto a essere sottoposto agli unit test. L'idea su cui si basa è che un oggetto invece di "procurarsi" da solo ciò di cui ha bisogno per funzionare si limiti a esporre le sue richieste tramite una qualche forma di contratto demandando così al container il compito di fornirgli il necessario. Per essere più precisi l'oggetto espone le proprie dipendenze, che comprendono sia le risorse del sistema sia altri oggetti. Questo porta alla creazione di un grafo di oggetti annidati in cui ciascun oggetto espone le proprie dipendenze all'oggetto chiamante. L'oggetto al livello più alto della gerarchia è quindi a conoscenza delle dipendenze di tutti gli oggetti utilizzati dall'applicazione. Quest'oggetto top-level è tipicamente utilizzato come entry point del sistema per assemblare tutti gli oggetti con le loro dipendenze prima di attivarli.

Come illustrato da Martin Fowler in un suo famoso articolo del 2004, esistono diverse implementazioni di IoC e la domanda da porsi è quale aspetto del Control Flow stanno invertendo.

In questo stesso articolo Fowler conia il termine Dependency Injection (DI) per riferirsi ad una specifica implementazione dell' Inversion of Control. Essa è rivolta ad invertire il processo di risoluzione delle dipendenze, facendo in modo che queste vengano iniettate dall'esterno. Banalmente, nel caso della programmazione Object Oriented, una classe A si dice dipendente dalla classe B se ne usa in qualche punto i servizi offerti.

Figura 15. La dipendenza tra le classi

Perché questo tipo di collaborazione abbia luogo, la A ha diverse alternative: istanziare e inizializzare la classe B, ottenere un'istanza di B attraverso una factory oppure effettuare un lookup attraverso un servizio di naming (ad esempio JNDI). Ognuno di questi casi implica che nel codice di A sia "scolpita" la logica di risoluzione della dipendenza verso B. Per chiarire introduciamo un esempio. Si vuole creare un semplice generatore di report in grado di generare output in formato testuale.

```
public class TxtReport {

  public void generate(String data) {
    System.out.println("genera txt report");
  }
}

public class ReportGenerator {

  TxtReport report = null;
```

```
  public Report generate(String data) {
    report = new TxtReport(); // risoluzione della dipendenza
    report.generate(data);
    return report;
  }
}
```

Nell'esempio sopra possiamo notare come la classe ReportGenerator abbia una dipendenza verso TxtReport e come questa sia risolta nel corpo del metodo generate (). Questo modo di operare, oltre a vincolare la creazione della dipendenza nel codice limitandone il riuso, tende a generare un forte accoppiamento tra le classi.

Il problema risiede nel fatto che senza l'ausilio di un apposito sistema la risoluzione delle dipendenze è ad esclusivo appannaggio delle classi stesse, che dovranno preoccuparsi di creare gli oggetti da cui dipendono o di ottenerne delle reference attraverso operazioni di lookup.

3.2.1 Dependency Injection

L'idea alla base della Dependency Injection è di avere un componente esterno che si occupi della creazione degli oggetti, delle loro relative dipendenze e di assemblarle mediante l'utilizzo dell'injection. In particolare esistono tre forme di injection:

1. **Constructor Injection**, dove la dipendenza è iniettata tramite l'argomento del costruttore.
2. **Setter Injection**, dove la dipendenza è iniettata attraverso un metodo "set".
3. **Interface Injection** che si basa sul mapping tra interfaccia e relativa implementazione (non utilizzato in Spring).

L'iniezione di dipendenza può essere realizzata in molteplici modi, tra cui il più semplice consiste nell'utilizzo di una factory. Riprendiamo l'esempio del generatore di report e cerchiamo di capire come realizzare una semplice iniezione della dipendenza.

Creiamo un'interfaccia Report che definisce le operazioni di base che un report deve avere e facciamola implementare alla classe `TxtReport`:

```
// Interfaccia Report
```

```
public interface Report {
  public void generate(String data);
  public void saveToFile();
}
// Classe TxtReport
public class TxtReport implements Report {
  String path;
  public TxtReport(String path) { this.path = path; }
  public void generate(String data) {
    System.out.println("genera txt report");
  }
  public void saveToFile() {
    System.out.println("File salvato!");
  }
}
```

Si modifichi, quindi, il `ReportGenerator` in modo che sia in grado di utilizzare oggetti di tipo Report e si aggiunga il relativo metodo setter:

```
public class ReportGenerator {
Report report;
  public Report generate(String data) {
    // report = new TxtReport();
    report.generate(data);
    return report;
  }
  public void setReport (Report report) {
    this.report = report;
  }
}
```

Si crei, infine, una classe factory, che ha il compito di istanziare oggetti di tipo `ReportGenerator` e di risolverne le dipendenze:

```
public class ReportGeneratorFactory {
  public static ReportGenerator createTxtReportGenerator() {
    ReportGenerator rg = new ReportGenerator();
    rg.setReport(new TxtReport()); // risoluzione della dipendenza
    return rg;
  }
}
```

Il client della nostra applicazione si presenterà in questo modo:

```
public class ReportClient {
  public static void main(String[] args) {
    String data = null;
    // reperimento dati
    ReportGenerator                          gen                          =
ReportGeneratorFactory.createTxtReportGenerator();
    gen.generate(data).saveToFile();
  }
}
```

La soluzione proposta ha permesso di disaccoppiare le classi ReportGenerator e TxtGenerator attraverso l'utilizzo della classe ReportGeneratorFactory. In particolare la factory, dopo aver creato l'oggetto ReportGenerator, ha iniettato, attraverso il relativo metodo setter (setter injection), un oggetto di tipo Report.

Pur essendo efficace, questo tipo "manuale" di DI continua ancora a non risolvere il problema di avere scolpito nel codice la creazione del report. Il problema è, infatti, stato semplicemente trasferito nella classe factory, che dovrà essere ogni volta modificata se si vorrà cambiare il tipo di report da utilizzare. Un modo migliore per lavorare con la Dependency Injection è di utilizzare come assembler uno IoC Container, che è in grado di compiere operazioni di injection. Per definizione un container è un componente esterno che si prende carico di una serie di compiti esonerando così lo sviluppatore dal preoccuparsene. Uno IoC Container non è altro che un container specializzato nella dependency injection, che basandosi su apposite configurazioni definite dall'utente (generalmente attraverso l'utilizzo di un file xml) è in grado di compiere opportune operazioni di injection.

Per spiegare, più nello specifico, come funziona il pattern DI in Spring è utile seguire l'esempio classico di Fowler. Supponiamo di voler scrivere un componente che esegua una ricerca in un database e restituisca tutti i film girati da un certo regista. Questa funzionalità la implementiamo con un singolo metodo della classe *MovieLister* riportata in seguito.

```
class MovieLister {
      public Movie[ ] moviesDirectedBy( String myDir ){
        List allMovies = finder.findAll();
        for ( Iterator it = allMovies.iterator(); it.hasNext(); ) {
             Movie movie = ( Movie ) it.next();
             if ( ! movie . getDirector().equals( myDir ))
               it.remove();
       }
 return ( Movie[] ) allMovies.toArray ( new Movie [ allMovies.size()]) ;
}
```

La classe è volutamente semplice, perchè l'oggetto d'interesse è l'oggetto finder, anzi per essere più precisi è il modo in cui la classe MovieLister viene collegata all'oggetto finder.

Quello che fa il metodo moviesDirectedBy non è altro che ottenere dall'oggetto finder tutti i film presenti sul database ed eliminare quelli che non hanno il valore directory uguale al parametro passato per la ricerca. L'oggetto finder responsabile di accedere al database implementa l'interfaccia MovieFinder riportata in seguito.

```
public interface MovieFinder {
    List findAll() ;
}
```

Quello che succede senza usare il DI è che al momento dell'instanziazione della classe MovieLister le deve essere passato un riferimento a una classe che implementa questa interfaccia.

```
class MovieLister{
   …
  private MovieFinder finder;
  public MovieLister() {
      finder = new ColonDelimitedMovieFinder("movies1.txt");
  }
```

L'ovvio problema di questa soluzione è che nel momento in cui si decide di usare una nuova implementazione dell'interfaccia MovieFinder si deve andare a modificare il codice. Già da questo esempio si vede come un approccio del genere comporta che scelte indipendenti dalle funzionalità del componente implichino delle modifiche al codice sorgente. Ovvero se, ad esempio, si decidesse di passare a un'implementazione basata su un altro database si dovrebbe modificare la classe MovieLister, senza che le sue funzionalità siano state intaccate da questa modifica. Utilizzando il DI la classe potrebbe essere riscritta nel modo seguente:

```
class MovieLister{
...
private MovieFinder finder
public MovieLister(MovieFinder finder ) {
    this.finder = finder;
}
class ColonMovieFinder{
...
private String fName;
public ColonMovieFinder(String fName) {
    this.fName = fName;
}
```

La configurazione è gestita in un file di contesto da Spring. In questo file troviamo indicata quale implementazione dell'interfaccia MovieLister deve essere utilizzata e anche quali parametri devono essere utilizzati nella creazione di questi oggetti.

```
<beans>
 <bean id="MovieLister "class="spring.MovieLister">
  <constructor-arg>
     <ref bean="MovieFinder" />
  </constructor-arg>
 </bean>
 <bean id="MovieFinder" class="spring.ColonMovieFinder">
   <constructor-arg>
     <value>movies1.txt</value>
   </constructor-arg>
 </bean>
</beans>
```

In generale la dependency injection può essere realizzata in tre modi ovvero tramite costruttore, tramite metodi setter e tramite iniezione di metodi astratti. Quest'ultima modalità non è supportata da Spring. Come si capisce del nome nella Constructor Injection il riferimento è iniettato nel costruttore. Quindi nell'esempio del MovieFinder passerei l'implementazione del finder nel costruttore del lister ed il nome del file nel costruttore del finder:

```
class MovieLister{
...
private MovieFinder finder
public MovieLister(MovieFinder finder ) {
    this.finder = finder;
}
class ColonMovieFinder{
...
private String fName ;
public ColonMovieFinder(String fName ) {
    this.fName = fName ;
}
```

Il file XML di configurazione conterrebbe quindi i seguenti tag:

```
<beans>
 <bean id="MovieLister" class="spring.MovieLister">
   <constructor-arg>
     <ref bean="MovieFinder"/>
   </constructor-arg>
 </bean>
 <bean id="MovieFinder" class="spring.ColonMovieFinder">
   <constructor-arg>
```

```
      <value>movies1.txt</value>
    </constructor-arg>
  </bean>
</beans>
```

In questo modo specifichiamo al container che stiamo utilizzando due bean MovieLister e MovieFinder che come costruttori hanno rispettivamente il finder e una Stringa che rappresenta il nome del file. Nel Setter Injection la dipendenza è passata come parametro di un metodo setter. È necessario utilizzare un costruttore vuoto e dei metodi setter per valorizzare le proprietà opportune. Quindi la class MovieLister dovrebbe essere modificata come nel codice seguente:

```
class MovieLister{
...
private MovieFinder finder ;
public void setFinder(MovieFinder finder){
    this.finder = finder ;
   }
}
class ColonMovieFinder {
...
public void setFilename(String filename) {
    this.filename = filename;
   }
}
```

E l'XML di configurazione diventerebbe il seguente.

```
<beans>
 <bean id="MovieLister"class="spring.MovieLister ">
   <property name="finder">
     <ref local="MovieFinder"/>
   </property>
 </bean>
 <bean id="MovieFinder" class=" spring.ColonMovieFinder">
   <property name="filename">
     <value>movies1.txt</ value>
   </property>
 </bean>
</ beans>
```

Già da questi esempi molto schematici è possibile costatare l'utilità e la potenza del pattern Dependency Injection. Infatti, le classi che andiamo a scrivere non implementano nessuna interfaccia o classe astratta e non hanno nessuna dipendenza con l'Ioc Container. Ogni modifica che andremo a fare, quindi, non impatterà in nessun modo sul codice già scritto, ma comporterà solo una modifica delle classi che implementano le interfacce utilizzate e

dei file di configurazione. Da questo si capisce come sia possibile eseguire dei test nel modo più efficiente sostituendo le classi "reali" con classi di test che implementano le stesse interfacce.

3.2.2 IoC Container

IoC Container è considerato il cuore di Spring, fornisce, infatti, un contesto altamente configurabile per la creazione e la risoluzione delle dipendenze di componenti che in tal contesto sono chiamati bean (da non confondere con i JavaBean).

A differenza di quanto avviene nei cosiddetti "heavyweight container" dove i componenti gestiti devono rispecchiare caratteristiche particolari o implementare opportune classi astratte fornite dal framework, in Spring un bean non deve aderire a nessun tipo di contratto e può essere rappresentato da una qualunque classe Java. Tecnicamente parlando lo IoC Container è realizzato da due interfacce:

- BeanFactory, che definisce le funzionalità di base per la gestione dei bean
- ApplicationContext, che estende queste funzionalità basilari aggiungendone altre tipicamente enterprise come ad esempio la gestione degli eventi, l'internazionalizzazione e l'integrazione con AOP

L'interfaccia BeanFactory rappresenta la forma più semplice di IoC Container in Spring e ha il compito di: creare i bean necessari all'applicazione, inizializzare le loro dipendenze attraverso l'utilizzo dell'injection e gestirne l'intero ciclo di vita.

Per svolgere questi compiti, il container si appoggia a configurazioni impostate dall'utente che, riflettendo lo scenario applicativo, specificano i bean che dovranno essere gestiti dal container, le dipendenze che intercorrono tra questi oltre alle varie configurazioni specifiche. In Spring esistono diverse implementazioni di BeanFactory, la più comune delle quali è senza dubbio la XmlBeanFactory che permette di utilizzare uno o più file XML per descrivere la configurazione da utilizzare.

I file di configurazione della XmlBeanFactory hanno la seguente forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                            http://www.springframework.org/schema/beans/s
pring-beans-3.0.xsd">
  <bean id="..." class="...">
    <!-- eventuali risoluzioni di dipendenze e proprietà -->
  </bean>
  <bean id="..." class="...">
    <!-- eventuali dipendenze e proprietà -->
  </bean>
  <!-- altri bean applicativi -->
```

Dopo l'intestazione del file XML, racchiuse tra i tag <beans>, troviamo le definizioni dei bean e, per ognuno di questi, le eventuali proprietà che ne descriveranno la struttura e il comportamento. Osserviamone alcune:

- l'Id associato al bean per essere richiamato tramite invocazioni al container.
- Il nome della classe che implementa il bean in caso questa venga istanziata direttamente o della relativa factory che si occuperà della sua creazione.
- Risoluzione di dipendenze attraverso metodi setter o costruttori appositi.
- Proprietà comportamentali che definiscono come il bean deve essere trattato dal container (scope, ciclo di vita, etc.).

3.3 Gestione dell'accesso ai dati

Quando si crea un'applicazione, uno degli aspetti fondamentali da considerare è la persistenza dei dati, ovvero la possibilità di conservare i dati tra le sessioni di un'**applicazione** in un formato che non sia l'archiviazione non volatile. Nel mondo Java, per realizzare la persistenza dei dati si ha a disposizione una numerosa varietà di API e framework, come ad esempio JDBC e Hibernate.

Spring si inserisce in questo contesto mettendo a disposizione tutte queste tecnologie in un ambiente altamente flessibile, sfruttando pienamente le potenzialità offerte dalla Dependency Injection e dall'Aspect Oriented. In particolare, Spring supporta:

1. JDBC
2. Java Persistence API (JPA)
3. Java Data Objects (JDO)

4. Hibernate
5. Common Client Interface (CCI)
6. iBATIS SQL Maps
7. Oracle TopLink

Nei prossimi paragrafi sarà illustrato, attraverso frammenti di codice, come poter utilizzare Spring per l'accesso ai dati mediante l'utilizzo di JDBC e di Hibernate.

Lo scopo degli esempi sarà di rendere persistente un database di libri. Il primo passo sarà, quindi, quello di realizzare il modello da persistere.

Il modello sarà composto di una semplice classe Book contenente le proprietà necessarie per rappresentare un libro e dai relativi metodi accessori.

```
Public class Book {

  String sib;
  String author;
  String title;

  public String getIsbn() { return isbn; }
  public void  setIsbn(String isbn) { this.isbn = isbn; }

  public String getAuthor() { return author; }
  public void  setAuthor(String author) { this.author = author; }

  public String getTitle() { return title; }
  public void  setTitle(String title) { this.title = title; }
}
```

Questa classe rappresenta l'entità base che dovrà essere salvata nel database denominato. La persistenza del modello sarà effettuata nella tabella books del database library. Di seguito sono riportati i dettagli della connessione e della creazione della tabella.

```
CONNECT 'jdbc:derby://localhost:1527/library;create=true';

CREATE TABLE BOOKS ( ISBN VARCHAR(13) NOT NULL,
                     AUTHOR VARCHAR(20),
                     TITLE VARCHAR(20),
                     PRIMARY KEY (ISBN));
```

Una delle peculiarità principali di Spring è di facilitare la scrittura di codice modulare in modo tale da favorirne il riuso. È proprio per questo motivo che Spring incoraggia l'utilizzo del **DAO** (Data Access Object), un pattern architetturale per la gestione

della persistenza e ha come scopo quello di separare le logiche di business da quelle di accesso ai dati. Questo si ottiene spostando la logica di accesso ai dati dai componenti di business ad una classe DAO rendendo gli Enterprise JavaBean indipendenti dalla natura del dispositivo di persistenza. Si ricordi che gli **Enterprise JavaBean** (**EJB**) sono i componenti software che implementano, lato server, la logica di business di un'applicazione web all'interno della piattaforma Java EE espletando servizi a favore della parte di front-end, ovvero per la logica di presentazione di un'applicazione web. Rappresentano dunque uno strato software residente su un application server all'interno di un'architettura software di tipo multi-tier. L'approccio adottato dal pattern DAO 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.

Quindi, l'idea alla base di questo pattern è quello di **descrivere le operazioni necessarie per la persistenza del modello** in un'interfaccia e di implementare la logica specifica di accesso ai dati in apposite classi.

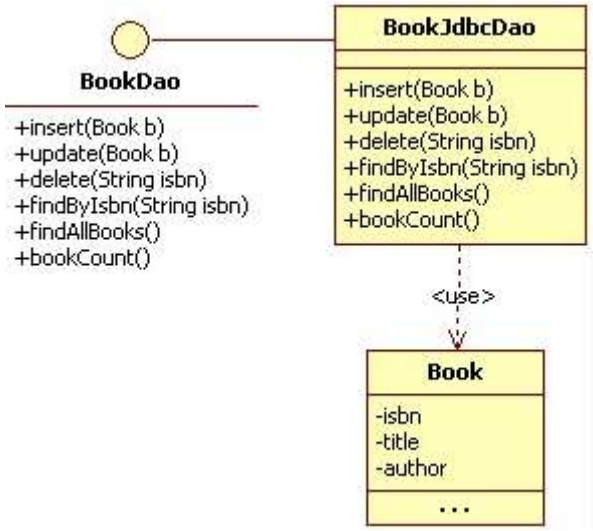


Figura 16. Diagramma UML del Data Access Object

In figura è mostrato il diagramma UML dell'architettura che sarà utilizzata nell' esempio. La logica di business necessaria per la persistenza del modello è descritta nella classe BookDao, mentre nella classe BookJdbcDao si trova un'implementazione specifica di questa interfaccia rivolta a gestire le logiche di accesso ai dati mediante tecnologia JDBC. In questo scenario è possibile notare come grazie all'utilizzo del pattern DAO sia possibile con poco sforzo fornire ulteriori implementazioni della classe BookDao per introdurre nuove logiche di accesso ai dati (ad esempio per l'utilizzo di Hibernate un ipotetico BookHibernateDao).

L'interfaccia DAO che potrà essere utilizzata sarà di questo tipo:

```
public interface BookDao {
  public void insert(Book book);
  public void update(Book book);
  public void delete(String isbn);
  public Book findByISBN(String isbn);
  public List<Book> findAllBooks();
  public int bookCount();
}
```

3.3.1 Spring JDBC

La forma primaria di accesso ai dati in Java è senza dubbio JDBC, un'API creata per rendere l'interazione con i *Relational database management system* (RDBMS) indipendente dalla piattaforma. Il suo utilizzo offre notevoli vantaggi in termini di performance e facilità di utilizzo, ma richiede allo sviluppatore di occuparsi dell'intera gestione dei processi di accesso ai dati, ovvero:

1. Richiesta della connessione al *datasource*.
2. Creazione del *PreparedStatement* e valorizzazione dei parametri.
3. Esecuzione del *PreparedStatement*.
4. Estrazione dei risultati dal *ResultSet* (in caso di interrogazioni).
5. Gestione delle eccezioni.
6. Chiusura della connessione.

Se per applicazioni di piccole dimensioni queste "complicazioni", possono essere accettabili, sicuramente provocano dei grossi disagi in ambito enterprise, dove nel corso degli anni sono nati appositi framework che, astraendo JDBC, tendono a semplificare questi aspetti. È in quest' ambito che si colloca Spring JDBC.

Spring offre diverse possibilità per la persistenza dei dati mediante JDBC, la principale delle quali è l'utilizzo della classe JdbcTemplate. Questa classe implementa l'intero processo di accesso ai dati attraverso *template methods*, rendendo possibile la personalizzazione di tutte le fasi di tale processo mediante l'ovverride dei metodi specifici.

Vediamo ora attraverso dei frammenti di codice come eseguire operazioni sul Database mediante JdbcTemplate.

Iniziamo con la creazione dell'implementazione dell'interfaccia *BookDao*, visto precedentemente, specifica per JDBC. Com'è possibile notare dal codice, tale implementazione possiede un riferimento alla classe JDBCTemplate, inizializzata attraverso il metodo *setDataSource()* che provvede a fornire *il DataSource* necessario.

```
public class BookJdbcDao implements BookDao {

  private JdbcTemplate jdbcTemplate;

  public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
  }
}
```

Attraverso il metodo update, invece, è possibile eseguire tutte quelle operazioni che comportano una modifica dei dati nel database, come ad esempio l'inserimento, la modifica e l'eliminazione.

```
public class BookJdbcDao implements BookDao {

 //Inserimento
 public void insert(Book book){
         jdbcTemplate.update("insert into books (isbn,autore,titolo)
values (?,?,?)", new Object[]{book.getIsbn(), book.getAutore(),
book.getTitolo()});
}

  //Modifica
  public void update(Book book){
    jdbcTemplate.update("update books set autore = ?, titolo = ? where
isbn=?",new Object[]{book.getIsbn(),book.getAutore(),book.getTitolo()});
```

```
  }

  // Eliminazione
  public void delete(String isbn){
    jdbcTemplate.update("delete from books where isbn=?",new Object[]{
isbn});
  }
}
```

Passando alle interrogazioni, il caso più semplice è la ricerca di un intero.

```
public class BookJdbcDao implements BookDao {

  // Query di un intero
  public int bookCount() {
    int rowCount=jdbcTemplate.queryForInt("select count(1) from books");
        return rowCount;
      }
}
```

Passando alle interrogazioni di oggetti, oltre alla query e i suoi parametri, è necessario specificare come deve avvenire il mapping tra i risultati presenti nel *ResultSet* e le proprietà dell'oggetto. Per fare ciò si utilizza l'interfaccia *RowMapper*, implementata nell'esempio dalla classe *BookRowMapper*.

```
public class BookJdbcDao implements BookDao {

  //Query di un singolo oggetto
  Public Book findByISBN (String isbn) {
    Book book = (Book) jdbcTemplate.queryForObject("select * from books
where isbn = ?",new Object[] { isbn },new BookRowMapper());
    return book;
  }

 // Query di una lista
  public List<Book> findAllBooks() {
    List<Book> books = (List<Book>) jdbcTemplate.query("select * from
books",new BookRowMapper());
    return books;
  }
}
```

Si può facilmente notare come la classe *BookRowMapper* contiene le regole di mapping tra una riga del *ResultSet* e la classe *Book*, infine essa è utilizzata dal JdbcTemplate per ritornare i valori della query. Infatti, grazie all'utilizzo *dell'Inversion of Control* è possibile iniettare nella classe un oggetto di tipo JdbcTemplate già inizializzato con un data source, quindi il file di configurazione risulterà essere di questo tipo:

```
 <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                  http://www.springframework.org/schema/beans/spring-beans 2.0.xsd">

    <bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName"
value="org.apache.derby.jdbc.ClientDriver" />
        <property name="url"
value="jdbc:derby://localhost:1527/library;create=true" />
        <property name="username" value="app" />
        <property name="password" value="app" />
    </bean>

    <bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource" />
    </bean>

    <bean id="bookDao" class="it.html.spring.book.BookJdbcDao">
        <property name="jdbcTemplate" ref="jdbcTemplate" />
    </bean>
</beans>
```

La classe *JdbcDaoSupport* permette di agevolare l'utilizzo di JdbcTemplate implementando direttamente i metodi *setDatasource()* e *setJdbcTemplate()*. Facendo estendere questa classe ai propri DAO sarà sufficiente iniettare un datasource per utilizzare un JdbcTemplate attraverso il metodo *getJdbcTemplate()*, ereditato anch'esso da *JdbcDaoSupport*.

```
public class BookJdbcDao extends JdbcDaoSupport implements BookDao {

  //Inserimento
  public void insert(Book book){
    getJdbcTemplate().update("insert into book (isbn,autore,titolo)
values(?,?,?)",new Object[]{ book.getIsbn(), book.getAuthor(),
book.getTitle()});
  }
}
```

Infine, la classe *SimpleJdbcTemplate* aggiunge alcune funzionalità offerte dalla versione 1.5 di Java alla classe base JdbcTemplate, semplificando ulteriormente le procedure di accesso ai dati. Grazie al supporto dei parametri a lunghezza variabile è possibile passare direttamente i valori ai metodi di update senza dover utilizzare un array di oggetti.

```
public class BookJdbcDaoSupport extends SimpleJdbcDaoSupport implements
BookDao {

  public void insert(Book book){
    getSimpleJdbcTemplate().update("insert into book
(isbn,autore,titolo) values (?,?,?)",book.getIsbn(),book.getAutore(),
book.getTitolo());
```

```
  }
```

Effettivamente anche le query possono essere notevolmente semplificate, in particolare Spring fornisce la classe *ParameterizedBeanPropertyRowMapper*, implementazione dell'interfaccia *RowMappe*r che si occupa di effettuare le operazioni di mapping viste in precedenza in maniera trasparente allo sviluppatore.

```
//Query di un singolo oggetto

public Book findByISBN(String isbn) {
  Book book=getSimpleJdbcTemplate().queryForObject("select * from books
where
isbn=?",ParameterizedBeanPropertyRowMapper.newInstance(Book.class),isbn)
;
  return book;
}

// Query di una lista di oggetti

public List<book> findAllBooks() {
  List<book> books=(List<book>) getJdbcTemplate().query( "select * from
books",ParameterizedBeanPropertyRowMapper.newInstance(Book.class));
  return books;
}
</book></book></book>
```

Sempre all'insegna della semplicità, Spring aggiunge alla programmazione JDBC il supporto ai parametri nominali, consentendo l'utilizzo di label come segnalibri al posto del classico punto interrogativo (?). Questo da un lato favorisce la leggibilità delle query e dall'altro elimina possibili complicazioni dovute alla posizionalità dei parametri. In Spring questa funzionalità è offerta sia attraverso l'utilizzo diretto della classe *NamedParameterJdbcTemplate*, che grazie al supporto del *SimpleJdbcTemplate* visto in precedenza. Il *binding* tra i parametri identificati dalle label e i relativi valori avviene tramite la classe *SqlParameterSource*, la cui implementazione base è la *MapSqlParameterSource*.

```
public class BookSimpleJdbcDaoSupportNamedValue extends
SimpleJdbcDaoSupport {

  public void update(Book book) {
      MapSqlParameterSource parameters = new MapSqlParameterSource();
      parameters.addValue("isbn", book.getIsbn());
        parameters.addValue("author", book.getAuthor());
        parameters.addValue("title", book.getTitle());
      getSimpleJdbcTemplate().update("update books set author =: author,
title =: title where isbn =: isbn", parameters);
```

```
        }
}
```

È così che il framework Spring si integra con la tecnologia JDBC, fornendo uno strumento molto flessibile per l'accesso ai dati. Nei prossimi paragrafi si vedrà, invece, come integrare il framework di persistenza Hibernate in Spring.

3.3.2 Spring e Hibernate

Si è già visto come nell'ambito informatico il termine persistenza sia utilizzato per indicare la possibilità dei dati di sopravvivere anche dopo la cessazione del programma che li ha creati, tale proprietà non è immediata in quanto all'atto dell'esecuzione i dati sono salvati nella memoria RAM, che per definizione è volatile. Il salvataggio dei dati si può ottenere utilizzando principalmente due metodi, ovvero mediante file system oppure attraverso un DBMS (Database Management System). Il secondo mezzo è considerato migliore per applicazioni che lavorano con grandi moli di dati in quanto fornisce un buon controllo sulla ridondanza, una riduzione dei tempi per lo sviluppo di applicazioni, flessibilità d'uso, funzioni di backup, assistenza e ripristino, e molti altri vantaggi.

Come precedentemente analizzato, nel linguaggio Java per accedere ai database (e di conseguenza permettere il recupero e il salvataggio degli oggetti) si usa ricorrere alle API JDBC, che svolgono il compito di collegare l'applicazione alla base di dati. Per fare ciò il programmatore deve aggiungere alle classi dei metodi che implementano le cosiddette operazioni CRUD (Create, Read, Update, Delete). Questa tecnica occupa mediamente però un terzo del tempo totale per lo sviluppo dell'intera applicazione, dovuto alla scrittura del codice che implementa il database ma soprattutto per il mantenimento delle connessioni. Un ulteriore punto di debolezza è dato dalla differenza tra la rappresentazione dei dati nel mondo object-oriented e quella del mondo relazionale.

Per cercare di ridurre tale inefficienza, sono state create delle tecniche di programmazione atte a fornire delle interfacce per implementare i dati dal modello relazionale al modello ad oggetti e viceversa, snellendo di fatto notevolmente il lavoro svolto dal programmatore per rendere gli oggetti persistenti. La principale di queste tecniche è la Object Relation

Mapping (ORM). Attraverso questo approccio ogni oggetto viene reso persistente nel database tramite l'inserimento di nuovi record i cui campi contengono i valori degli attributi dell'oggetto stesso. Si viene quindi a creare la relazione tra oggetto Java e tabella SQL. Una soluzione ORM consiste di quattro componenti caratteristici:

1. Una API per eseguire le operazioni CRUD sugli oggetti delle classi del modello.
2. Un linguaggio o una API per specificare query che fanno riferimento alle classi e alle proprietà delle classi.
3. Uno strumento per la specifica del mapping mediante metadati.
4. Una tecnica per l'implementazione di ORM per interagire con oggetti transazionali al fine di eseguire funzioni di ottimizzazione.

L'associazione fisica tra la classe e la tabella è ottenuta mediante l'utilizzo di file di descrizione, file di mapping, in cui si specificano le modalità di conversione tra gli attributi dell'oggetto e i campi della tabella. I principali vantaggi derivanti dall'uso di una soluzione ORM sono:

- snellimento della parte di codice riservata alla persistenza dei dati;
- maggiore facilità per quanto riguarda la manutenzione del codice grazie alla separazione tra il modello ad oggetti e il modello relazionale;
- performance più efficienti grazie alle numerose opzioni di ottimizzazione presenti;
- elevata portabilità rispetto alla tecnologia DBMS utilizzata.

L'esempio più noto di ORM per il linguaggio Java è Hibernate. Hibernate è un progetto sviluppato nel 2001 da Gaving King e Christian Bauer e sono rispettivamente del 2003 e 2010 le versioni Hibernate 2 e Hibernate 3.x. Attualmente l'ultima versione rilasciata è la 4.2.4. Esso è inserito all'interno del contesto dell'application server open source Jboss. Hibernate è una piattaforma middleware open source, fornisce supporto non solo per quanto riguarda la mappatura di un modello di dominio orientato agli oggetti in un classico database relazionale, ma anche per quanto riguarda il reperimento dei dati da esso, ovvero la gestione delle query. Essenzialmente l'utilizzo di Hibernate permette tre approcci di programmazione:

- Top Down. Si parte da un diagramma delle classi di dominio e dalla sua implementazione in Java e si ha completa libertà rispetto allo schema della base di dati. Si specificano poi i file XML con i mapping ed, eventualmente, si può usare in particolare lo strumento di Hibernate *hbm2ddl*, per generare lo schema della base di dati. Questa metodologia si applica in quei contesti dove non è già presente un database definito.
- Meet in the middle. Questa metodologia unisce, attraverso una completa mappatura, una base di dati esistente e un diagramma delle classi di dominio dell'applicazione.
- Bottom up. Si parte da una base di dati esistente e si ha completa libertà rispetto al dominio dell'applicazione. Si usa una serie di strumenti per generare lo schema di base del codice Java del modello a oggetti per la gestione della persistenza, ovvero il codice delle classi persistenti.

Hibernate può essere utilizzato principalmente in modalità Native o Provider. Nel primo caso si utilizzano dei file XML di configurazione per realizzare la mappatura tra le classi da rendere persistenti con le tabelle del database: precisamente ad ogni classe Java si associa il suo file di mappatura XML. Questo rende il codice molto leggibile e facilmente modificabile. La seconda modalità, invece, prevede l'inserimento diretto all'interno delle classi del codice inerente alla mappatura utilizzando le Java Annotation. Questo ha il vantaggio di ridurre i file utilizzati, ma rende il codice molto meno chiaro, di conseguenza difficilmente modificabile.

Soffermiamoci in modo particolare sull'integrazione di Spring con Hibernate, senza dimenticare che gli stessi principi valgono anche per gli altri ORM supportati. Perché un ORM sia in grado di generare codice SQL è necessario specificare il mapping tra gli oggetti da persistere e le tabelle nel database. In accordo con le specifiche Hibernate, uno dei metodi per effettuare questa operazione è utilizzare un file XML che, per convenzione, ha nome: <classe>.hbm.xml. Di seguito è illustrato il file book.hbm.xml responsabile del mapping tra la classe Book vista nei paragrafi precedenti e la relativa tabella nella base dati.

```
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="it.html.spring.book">
  <class name="Book" table="BOOK">
    <id name="isbn" type="string" column="ISBN"></id>
    <property name="titolo" type="string" column="TITOLO"></property>
    <property name="autore" type="string" column="AUTORE"></property>
  </class>
</hibernate-mapping>
```

La gestione del mapping per mezzo dei file XML è relativamente semplice, ma può risultare difficile da gestire, sopratutto quando il numero delle classi da persistere è alto. Un'alternativa è data dall'adozione delle JPA annotations introdotte da Sun a partire da Java 5 per permettere di specificare il mapping all'interno delle stesse classi da persistere. Le JPA annotations possono essere utilizzate da tutti gli ORM JPA-compliant come Hibernate. Di seguito è mostrata la classe Book con aggiunte le annotations necessarie per la sua persistenza.

```
@Entity
@Table(name = "BOOK")
public class Book {

  @Id
  @Column(name = "ISBN")
  String isbn;

  @Column(name = "AUTORE")
  String autore;

  @Column(name = "TITOLO")
  String titolo;

  public String getIsbn() { return isbn; }
  public void setIsbn(String isbn) { this.isbn = isbn; }

  public String getAutore() { return autore; }
  public void setAutore(String autore) { this.autore = autore; }

  public String getTitolo() { return titolo; }
  public void setTitolo(String titolo) { this.titolo = titolo; }

  public String toString() {
    return isbn + " - " + autore + " - " + titolo;
  }
}
```

Dal codice risultante si nota subito come l'utilizzo di queste annotations renda le operazioni di mapping più semplici e manutenibili rispetto ai rispettivi file XML. Per questa ragione, d'ora in poi, prenderemo in considerazione quest'alternativa.

3.3.3Hibernate Configuration

Si consideri ora una possibile implementazione di Hibernate relativamente al BookDao visto in precedenza. Come per JDBC, Spring ci viene in aiuto fornendo HibernateTemplate, uno specifico template che si fa carico delle operazioni di gestione, tra le quali:

- Richiesta della connessione alla factory
- Apertura della transazione
- Gestione delle eccezioni
- Commit/Rollback della transazione
- Chiusura della connessione

Unitamente a HibernateTemplate Spring mette a disposizione la classe di supporto HibernateDaoSupport per facilitare l'iniezione all'interno dei DAO. Estendendo questa classe sarà sufficiente richiamare il metodo getHibernateTemplate() per poter avere un hibernate template da utilizzare.

```
public class BookHibernateDaoSupport extends HibernateDaoSupport
implements BookDao {

  @Transactional
  public int bookCount() {
    return findAllBooks().size();
  }

  @Transactional
  public void delete(String isbn) {
    Book book = (Book) getHibernateTemplate().get(Book.class, isbn);
    getHibernateTemplate().delete(book);
  }

  @Transactional(readOnly = true)
  public List<book> findAllBooks() {
    return getHibernateTemplate().find("from Book");
  }

  @Transactional(readOnly = true)
```

```
  public Book findByISBN(String isbn) {
    return (Book) getHibernateTemplate().get(Book.class, isbn);
  }

  @Transactional
  public void insert(Book book) {
    getHibernateTemplate().saveOrUpdate(book);
  }

  public void update(Book book) {
    getHibernateTemplate().saveOrUpdate(book);
  }
}
</book>
```

Fino ad ora non è stato, però, ancora affrontato il problema delle transazioni. Nella maggior parte dei casi, infatti, quello che si vuole ottenere è che le operazioni effettuate dai metodi del DAO siano transazionali, cioè abbiano un senso compiuto nella loro interezza. Questo comportamento è reso possibile dall'utilizzo dell'apposita annotation @Trasactional, con la quale è possibile richiedere al framework che tutte le chiamate ai metodi coinvolti siano trattate nella stessa transazione.

Precedentemente è stato detto che, grazie ad un HibernateTemplate è possibile implementare, attraverso Hibernate, un DAO per la persistenza. In realtà, i componenti che entrano in gioco per permettere a HibernateTemplate di svolgere il proprio lavoro sono molteplici, anche se non direttamente visibili, in quanto sono mascherati dall'utilizzo dell'Inversion of Control. È proprio qui che l'utilizzo di IoC esprime un ruolo fondamentale, semplificando notevolmente le operazioni di persistenza. Dunque, il primo bean da configurare è quello responsabile della connessione verso la sorgente dati. Come per l'esempio relativo a JDBC anche ora sarà utilizzato il DriverManagerDataSource, un semplice datasource in grado di aprire una connessione ogni volta che è richiesta. Questo modo di operare non è certamente dei migliori per quanto concerne la gestione delle risorse, per questo motivo, in ambienti più complessi, si suggerisce l'utilizzo di datasource più efficienti come il SingleConnectionDataSource.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
  <bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource">

    <property name="driverClassName"
              value="org.apache.derby.jdbc.ClientDriver" />
    <property name="url"
              value="jdbc:derby://localhost:1527/books;create=true" />
    <property name="username" value="app" />
    <property name="password" value="app" />
  </bean>
</beans>
```

Per far sì che Hibernate svolga il proprio lavoro di persistenza sono necessarie alcune configurazioni che indicano al framework quali parametri utilizzare per la creazione del SessionFactory, ovvero l'oggetto responsabile dell'apertura delle sessioni verso il database. Tali parametri includono:

- Datasource da utilizzare per la connessione con il database.
- Dialetto SQL utilizzato da Hibernate per l'ottimizzazione delle query.
- Lista degli eventuali file di mapping.
- Altre informazioni accessorie

Nell'utilizzo classico di hibernate queste configurazioni possono essere espresse sotto forma di file xml (utilizzando il file hibernate.cfg.xml) oppure tramite annotations. A queste modalità Spring aggiunge una terza alternativa sfruttando il meccanismo dell'**Inversion of Control per la creazione di SessionFactory** e la relativa injection dei parametri necessari. Di seguito sono mostrate due possibili configurazioni per il bean sessionFactory da impiegare nell'applicazione presa in esempio. In particolare, la prima è un caso di utilizzo delle annotations JPA per il mapping delle classi da persistere, la seconda, invece, un caso di adozione di file XML.

```
<!-- beans ... -->

  <!-- Session Factory da utilizzare per mapping attraverso JPA
Annotations -->
  <bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFa
ctoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="annotatedClasses">
      <list>
        <value>it.html.spring.book.Book</value>
      </list>
```

```
    </property>
    <property name="hibernateProperties">
      <props>
        <prop
key="hibernate.dialect">org.hibernate.dialect.DerbyDialect</prop>
        <prop key="hibernate.show_sql">true</prop>
        <prop key="hibernate.hbm2ddl.auto">update</prop>
      </props>
    </property>
  </bean>


  <!-- Session Factory da utilizzare per mapping attraverso file xml-->
  <!--                      bean                      id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mappingResources">
      <list>
        <value>book.hbm.xml</value>
      </list>
    </property>
    <property name="hibernateProperties">
      <props>
        <prop
key="hibernate.dialect">org.hibernate.dialect.DerbyDialect</prop>
        <prop key="hibernate.show_sql">true</prop>
        <prop key="hibernate.hbm2ddl.auto">update</prop>
      </props>
    </property>
  </bean>
</beans>
```

Si è accennato alla transazionalità delle operazioni, senza però specificare quale componente la implementasse. Il responsabile delle transazioni è il TransactionManager, un componente in grado di creare e gestire le transazioni attraverso un datasource e al cui utilizzo si affida HibernateTemplate.

Spring fornisce diversi transaction manager, uno per ogni tecnologia ORM supportata. Nel caso preso in esempio quello da utilizzare è l'HibernateTransactionManager in grado di operare tramite una session factory.

Oltre al transaction manager occorre anche specificare, attraverso il tag tx: annotation-driven che le direttive di transazione sono fornite attraverso l'utilizzo di annotations.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
  <tx:annotation-driven />

  <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
    <property name="sessionFactory" ref="sessionFactory" />
  </bean>

</beans>
```

A questo punto l'ultimo bean rimasto da configurare è semplicemente quello rappresentate l'implementazione specifica del Book Dao.

```
<!-- ... -->

  <bean id="bookDao" class="it.html.spring.book.BookHibernateDaoSupport">
    <property name="hibernateTemplate" ref="hibernateTemplate" />
  </bean>

  <!-- ... -->
</beans>
```

In caso di utilizzo della classe HibernateDaoSupport è possibile iniettare nel BookDaodirettamente un TransactionManager rendendo di fatto inutile configurare HibernateTemplate. Sarà poi la stessa classe di supporto responsabile di istanziare un oggetto di tipo HibernateTemplate.

3.4 Spring MVC

Spring MVC è un framework per realizzare applicazioni web basate sul modello MVC sfruttando i punti di forza offerti dal framework Spring come l'inversion of control e l'aspect oriented programming. Esso si occupa di mappare i metodi e le classi Java con determinati url, di gestire differenti tipologie di "viste" restituite al client, di realizzare applicazioni internazionalizzate e di gestire i cosiddetti temi per personalizzare al massimo l'esperienza utente.

Per comprendere al meglio il framework è necessario ricordare il pattern teorico che esso implementa ovvero il modello MVC. MVC rappresenta un acronimo per Model View Controller ovvero le tre componenti principali di un'applicazione web. Grazie a questo pattern i compiti sono separati verso questi componenti:

- i Model (in italiano Modelli) si occupano di accedere ai dati necessari alla logica di business implementata nell'applicazione (nel caso di un'applicazione per una biblioteca potrebbero essere le classi Libro, Autore, Scaffale);
- le View (in italiano Viste) si occupano di creare l'interfaccia utilizzabile dall'utente e che espone i dati da esso richiesti (nel caso bibliotecario potrebbero essere le pagine HTML del catalogo, le form di ricerca oppure i PDF contenenti ricerche o appunti);
- i Controller (in italiano Controllori) si occupano di implementare la vera logica di business dell'applicazione integrando le due componenti precedenti, ricevendo gli input dell'utente, gestendo i modelli per la ricerca dei dati e la creazione di viste da restituire all'utente (nel nostro caso potrebbero essere il motore di ricerca interno oppure il sistema di login al sito Internet della biblioteca).

Spring MVC implementa perfettamente approccio mantenendo sia i concetti sia la nomenclatura del pattern. All'interno di un'applicazione Spring MVC avremo quindi:

- i Model, che sono rappresentati dalle classi che a loro volta rappresentano gli oggetti gestiti e le classi di accesso al database;
- le View, che, invece, sono rappresentate dai vari file JSP (che vengono compilati in HTML) e da eventuali classi per l'esportazione in formati diversi da HTML (PDF, XLS, CSV);
- i Controller, che sono rappresentati da classi che rimangono "in ascolto" su un determinato URL e, grazie ai Model e alle View, si occupano di gestire la richiesta dell'utente.

Spring MVC, secondo la documentazione ufficiale, presenta, inoltre, molti altri vantaggi:

- è adattabile, flessibile e non intrusivo grazie alla presenza di comode e chiare Java Annotations;
- permette di scrivere codice riusabile;
- utilizza una libreria JSP sviluppata ad hoc per facilitare alcune operazioni ripetitive;

- è dotato di nuovi scope per i bean (request e session) che permettono di adattare i container base di Spring anche al mondo web.

3.4.1Moduli di Spring MVC

In Spring, il modello Model-View Controller è progettato attorno ad un front controller chiamato DispatcherServlet, unica vera variante rispetto al modello "classico", il quale ha il compito di inoltrare le richieste alle varie interfacce attive durante una fase HTTP request. I moduli più importanti di Spring MVC sono:

- HandlerMapping: seleziona quale Controller dovrà gestire la richiesta in arrivo secondo i suoi attributi.
- HandlerAdapter: permette l'esecuzione della logica del Controller selezionato dal HandlerMapping.
- Controller: nel mezzo tra Model e View, amministra le richieste in arrivo e reindirizza le risposte al View.
- View: è responsabile di far dialogare l'applicazione con l'utente, gestendo le risposte che possono provenire dal Controller direttamente o dal Model.

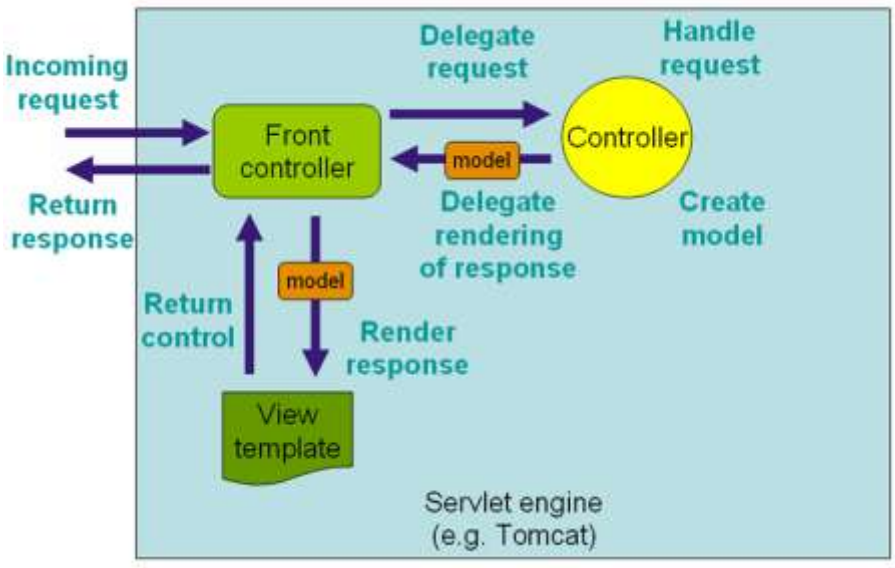
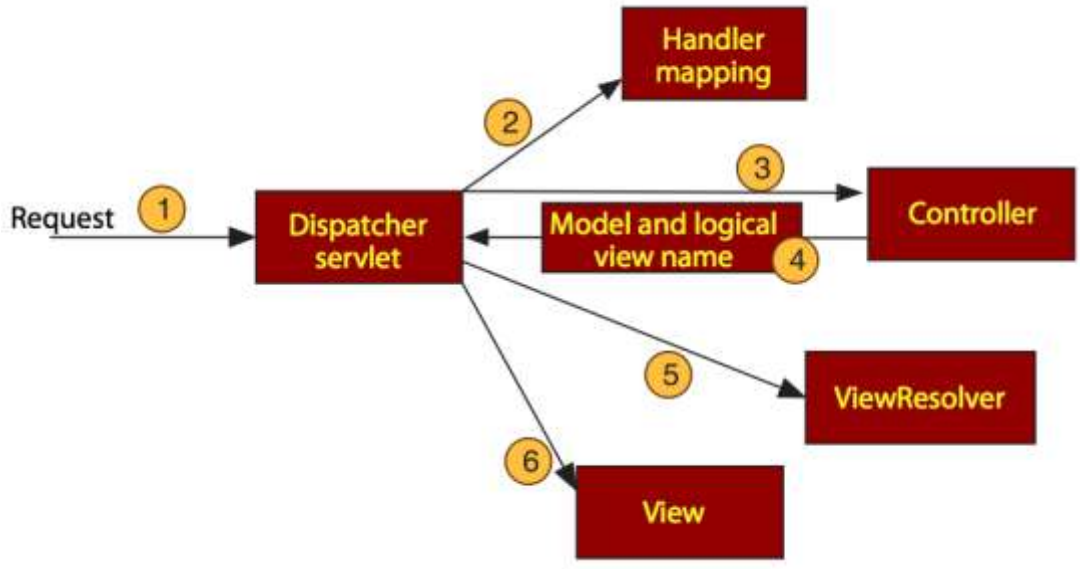


Figura 17. Il flusso di richieste in Spring MVC

3.4.2Gestire una richiesta con Spring MVC

Ogni volta che un utente clicca su un link o dà il comando di invio in un determinato form, effettua una richiesta alla web application.

Con Spring MVC, tale richiesta è letta inizialmente dal DispatcherServlet, il quale, una volta averla ricevuta, delega la responsabilità della sua gestione al Controller appropriato. Solitamente in una web application esso non è unico, così il DispatcherServlet si affida all'Handler Mapping per riconoscere il destinatario, secondo una mappa che associa l'URL della richiesta ai controller. Una volta identificato, il DispatcherServlet gli invia la richiesta. Arrivata a destinazione, la richiesta è eseguita interagendo con il Model, producendo spesso informazioni che dovranno poi essere visualizzate dall'utente. Quindi, il controller restituirà tali informazioni, sotto forma di un oggetto ModelAndView, al DispatcherServlet, che le inoltra prima al ViewResolver, il quale sceglierà la pagina corretta, poi allo strato View che restituirà tali informazioni all'utente in un formato user-friendly (come HTML).



Capitolo 4: Application Framework & Code Generation

4.1 Generatori di codice

Qualsiasi attività, tranne alcune rare eccezioni, legata allo sviluppo dei prodotti software necessita di un'adeguata organizzazione. È opportuno adottare strumenti idonei che consentano di gestire anche i flussi di informazioni relative allo sviluppo. Sono attualmente disponibili numerosi strumenti specifici per supportare le varie fasi di produzione, tra i quali sono di notevole importanza e rilievo:

- i sistemi di workflow management, utilizzati per descrivere e gestire la sequenza di attività,
- i sistemi di controllo di versione (CVS),
- i Computer-Aided Software Engineering (CASE).

I CASE rappresentano una nuova generazione di strumenti che applica rigorosi principi dell'ingegneria del software allo sviluppo e all'analisi delle specifice. I primi strumenti per lo sviluppo di sistemi risalgono agli anni Settanta; furono introdotti per automatizzare e supportare la produzione e la manutenzione dei diagrammi strutturali. Attualmente i sistemi più comunemente utilizzati sono i CASE. Se l'automazione interessa l'intero ciclo di vita è più indicato designare il sistema di sviluppo come I-CASE (Integrated-Computer-Aided Software Engineering), mentre si parla semplicemente di CASE se il ciclo di vita del software viene coperto solo parzialmente, indirizzandosi soltanto ad alcuni processi interessati allo sviluppo. I metodi computer-assisted per la realizzazione di software vengono utilizzati in particolare per progetti molto vasti dove sono implicati diversi

componenti software e il lavoro viene svolto da un gruppo di più persone. Gli strumenti CASE permettono di avere a disposizione una visioe comune dello stato di avanzamento dei lavori e poterla condividere all'interno del team di sviluppo. In questo modo le fasi tendono a essere disciplinate e controllate in man iera più rigorosa. Tramite alcuni CASE è possibile anche ottenere una rappresentazione grafica dei progressi compiuti, o per conto, dell'involuzione nello sviluppo del progetto. Questi strumenti vengono spesso a far parte dell'insieme dei processi designati per garantire la qualità del prodotto software; proprio per questo vengono sempre più comunemente adottati visto che le organizzazioni tendono sempre più a specializzarsi sulla qualità. Inoltre, gli strumenti integrati risultano molto efficaci poiché riescono a generare applicazioni intere solo dalle specifiche di progettazione. Esistono CASE per ogni fase del processo di realizzazione di un software, come ad esempio:

- i generatori di documentazione
- i generatori di database
- i generatori di codice
- i costruttori di test

Avendo a disposizione i generatori di codice, il team di programmazione può concentrarsi solo su alcune parti più importanti e utilizzare moduli preconfezionati per adeguarli alle esigenze specifiche, con un notevole risparmio di tempo e risorse. I CASE più recenti sono stati ampliati utilizzando la metodologia rapid prototyping per sviluppare applicazioni più veloci, a un costo minore e di qualità più elevata. Con riferimento alle fasi del processo a cui vengono applicati, i CASE si possono suddividere in due grandi categorie:

- upper-CASE, che supportano le fasi iniziali del processo, come l'analisi dei requisiti e la progettazione
- lower-CASE, che supportano le fasi finali, come la programmazione, il debugging e il testing.

I generatori rivestono un ruolo di fondamentale importanza anche nell'ambito del riuso.

Il processo di progettazione, nella maggior parte delle discipline ingegneristiche si basa sul riutilizzo dei sistemi o componenti esistenti. L'ingegneria del software basata sul riutilizzo è confrontabile con le strategie di ingegneria del software dove il processo di sviluppo si impernia sul riutilizzo del software esistente. Per quanto i benefici del riutilizzo siano riconosciuti da diversi anni, solo negli ultimi dieci anni c'è stata una transazione graduale dallo sviluppo software originale allo sviluppo software basato sul riutilizzo.

Il riutilizzo di concetti attraverso schemi si basa sulla descrizione del concetto in modo astratto e lasciando allo sviluppatore del software la creazione di un'implementazione. Un approccio alternativo è il riutilizzo basato su generatori (Biggerstaff, 1998): la conoscenza riutilizzabile è integrata in un generatore di programmi che può essere programmato da esperti del dominio utilizzando un linguaggio di dominio orientato o uno strumento CASE interattivo che supporta la generazione di sistemi.

La descrizione dell'applicazione specifica, in modo astratto, quali componenti riutilizzabili devono essere utilizzati, come devono essere combinati ei parametri. Utilizzando queste informazioni, un sistema software operativo può essere generato (Figura 18.8).

Il riutilizzo basato su generatori sfrutta il fatto che le applicazioni nello stesso dominio, come i sistemi aziendali, hanno architetture comuni e svolgono funzioni comparabili. Per esempio, i sistemi di elaborazione dati normalmente seguono un modello input-elaborazione-output e di solito includono operazioni quali la verifica dei dati e la generazione di rapporti. Pertanto, i componenti generici per la selezione di elementi da un database, per il controllo che questi siano nel raggio d'azione e per la creazione di report, possono essere creati e inseriti in un generatore di applicazione. Per riutilizzare questi componenti, il programmatore deve semplicemente selezionare i dati da utilizzare, i controlli da applicare e il formato dei report.

Nei paragrafi successivi saranno analizzati tre generatori di codice. I primi due, Eclipse Modeling Framework e Spring Roo, sono i generatori di codice integrati con i principali strumenti IDE; essi, inoltre, offrono anche un editor per rappresentare il modello definito

dall'utente. Infine, sarà analizzato SkyGen, application framework e code generation, realizzato ed adoperato con ottimi risultati dall'azienda SkyIT e con il quale è stata realizzata un'applicazione web, che sarà oggetto principale del prossimo capitolo. I tre generatori saranno descritti e confrontati in modo tale da capire quale siano i motivi che spingono all'utilizzo di uno di essi.

4.2 Eclipse Modeling Framework

Eclipse Modeling Framework (EMF) è un framework per la generazione di tools e applicazioni basate su un modello strutturato, incentrato sul concetto del Model-Driven Architecture Development. L'idea che sta alla base di questo framework è la ricerca della fusione tra il mondo della modellazione e quello della programmazione. La funzionalità principale di EMF, infatti, è ricevere in input un modello (sotto forma di diagramma UML, schema XML, interfacce Java annotate...) e fornire come output una serie di classi Java completamente implementate, che realizzano i vincoli, le relazioni e le associazioni descritte nel modello di partenza. Il compito del programmatore è così alleggerito, dato che il suo lavoro consisterà innanzitutto nel creare il modello (operazione piuttosto semplice grazie alle interfacce Java annotate, di cui parleremo pi`u avanti), e quindi, dopo aver generato il codice tramite EMF, nell'implementare i comportamenti specifici che il modello non è in grado di rappresentare (ad esempio la funzionalità di un metodo in una classe Java). Le parti più ripetitive e noiose (metodi getter e setter, gestione delle relazioni tra oggetti, gestione di eventi...) vengono invece create in maniera automatica da EMF. Oltre alla generazione di codice, EMF fornisce anche altre funzionalità, quali il supporto alla customizzazione del codice, una Reflective API per generare dinamicamente modelli, il supporto per serializzazione e deserializzazione dei dati (in formato XMI3) e la possibilità di generare semplici editor grafici ad albero per le nostre applicazioni.

4.2.1 I componenti di EMF

L'Eclipse Modeling Framework `e costituito da tre componenti principali: *EMF Core*, *EMF Edit* ed *EMF Codegen*.

- **EMF Core** include il metamodello *ECore*, ovvero il "formato" in cui vengono convertiti tutti i modelli utilizzati. È interessante notare che l'implementazione di ECore usata da EMF è generata dallo stesso code generator di EMF, e ha quindi tutte le sue caratteristiche e vantaggi.

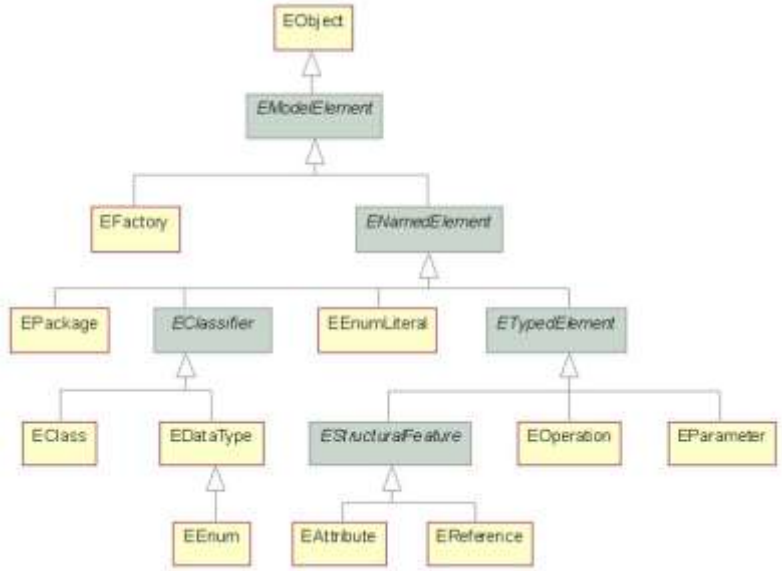


Figura 19. La gerarchia del modello Ecore

Come si vede nella *Figura 19*, la gerarchia di ECore ricorda abbastanza quella di Java: la classe di base è EObject, mentre a scendere si trovano varie classi che rappresentano oggetti tipici di un class diagram (classi, attributi, operazioni). La classe EObject fornisce tutta una serie di funzioni di base, ereditate da tutti gli altri oggetti; in particolare, il metodo eClass() restituisce informazioni sull'istanza corrente, mentre eGet e eSet consentono operazioni riflessive sugli attributi dell'oggetto. Inoltre EObject estende l'interfaccia Notifier, predisponendo quindi ogni oggetto a inviare notifiche a osservatori registrati in caso di modifica del proprio stato: tutti gli EObject, infatti, hanno una lista di observers, che si possono

registrare esplicitamente o creare tramite un'apposita factory; la chiamata del metodo eNotify() scorre questa lista inviando notifiche a ciascun observer. EMF Core si occupa anche di validazione, persistenza, serializzazione, reflection e supporto runtime per i modelli generati.

- **EMF Edit** fornisce classi riutilizzabili di supporto che consentono la visualizzazione di oggetti del modello usando il framework JFace4, e mette a disposizione un framework per la gestione di comandi. Usando il code generator di EMF si possono creare editor grafici completi per il proprio modello sotto forma di plugin per Eclipse; se invece non si vuole che il proprio editor sia dipendente da Eclipse si possono comunque sfruttare e personalizzare le classi create e le funzionalità che esse offrono.
- **EMF Codegen** ha due importanti scopi: per prima cosa fornisce un framework estensibile per l'importazione di modelli, inoltre è il luogo *"where the magic happens"*, dato che si occupa della generazione vera e propria di codice per i componenti Core ed Edit. Il primo passo per utilizzare EMF è l'importazione di un modello. Attualmente sono quattro i formati che si possono utilizzare:
    1. **XMI**, anche se questa scelta `e decisamente sconsigliata a causa della sua complessità.
    2. **XMLSchema**, un modello di questo tipo, però, è meno espressivo rispetto a UML, ad esempio non supporta riferimenti bidirezionali; anche questa scelta non è particolarmente consigliata, a meno di avere il modello già pronto.
    3. **UML**, si possono, dunque, importare direttamente i class diagram UML**.**
    4. **Java annotated code**: se non si dispone di un editor grafico UML, o se non si è pratici del linguaggio e si è più propensi a scrivere codice, questa opzione è un'interessante alternativa.

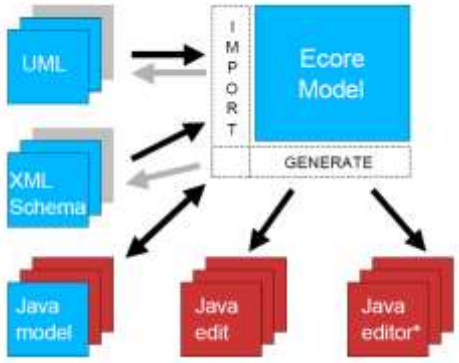


Figura 20. Il funzionamento di EMF

4.2.2 Personalizzare il codice generato

Una volta generate le implementazioni delle nostre classi, potremmo limitarci a utilizzare l'editor di default che EMF ha preparato per noi, sfruttando anche il grande vantaggio che ogni successiva possibile modifica al modello può essere rapidamente trasformata in codice semplicemente rigenerando il tutto grazie al file genmodel, che inoltre supporta anche la rigenerazione parziale. Nella maggior parte dei casi, però, il modello sarà abbastanza complicato da richiedere anche un po' di customizzazione e di modifiche, se non altro per implementare i metodi di cui EMF ha fornito solo lo scheletro. Naturalmente, si può personalizzare il codice senza rischiare di perderlo alla successiva rigenerazione. Aggiungere, infatti, un metodo ex-novo non comporta alcun problema: il codice aggiunto sarà preservato anche in seguito a una rigenerazione. Invece, per i metodi generati automaticamente da EMF, indicati dal tag @generated, tutte le modifiche fatte dal programmatore saranno perse alla successiva rigenerazione. Per ovviare al problema è possibile rimuovere il tag, ma è chiaro che questo significa rinunciare alle modifiche automatiche del codice di quel metodo in caso di cambiamento del modello. Il sistema corretto è quindi una sorta di overload.

4.2.3 Funzionalità avanzate

EMF, oltre a generare le implementazioni delle varie classi, all'interno del modello Java crea anche alcune classi di servizio, in particolare le classi Factory e Package. La classe Factory, come si intuisce dal nome, contiene tutti i metodi per istanziare gli oggetti facenti parte del modello; la classe Package, invece, consente l'accesso ai metadati del modello, ciò è particolarmente utile quando si usa la Reflective API. Grazie ai metodi eSet e eGet offerti da EObject è, quindi, possibile accedere in maniera generica e dinamica a oggetti e attributi: le prestazioni sono leggermente inferiori a una normale invocazione, ma il vantaggio in termini di generalità è notevole. Tutto questo consente a EMF Edit, ad esempio, di creare comandi riutilizzabili e indipendenti da un particolare modello. Infine, per quanto concerne le capacità di validazione di modello offerte da EMF si può affermare che EMF non è in grado di generare automaticamente le condizioni che facciano fallire un check di validità. Il compito spetta quindi al programmatore, che come al solito troverà comunque una struttura di validazione predisposta e pronta per essere implementata.

4.2.4 Un esempio di utilizzo

Il modello, *"Library"*che si userà come esempio è preso dal Tutorial ufficiale di EMF. Esso è costituito dalle entità: Author, Book e Library. Il modello è molto semplice, infatti:

- Author ha un unico attibuto: String name
- Book ha un attibuto String title e un attributo author di tipo Author
- Library ha, invece, una collection di oggetti Author e una di oggetti Book

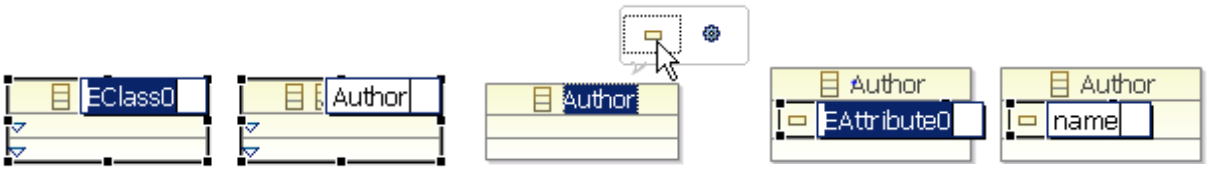
Si inizia con il creare un nuovo Empty EMF Project, che sarà pronto per ospitare il modello EMF, che sarà realizzato.

Successivamente si deve espandere il progetto, si seleziona la cartella model e al suo interno si crea un diagramma EMF, attraverso un nuovo Ecore Diagram, ed infine cambiamo il valore del campo "Domain File name" in "Library.ecore".

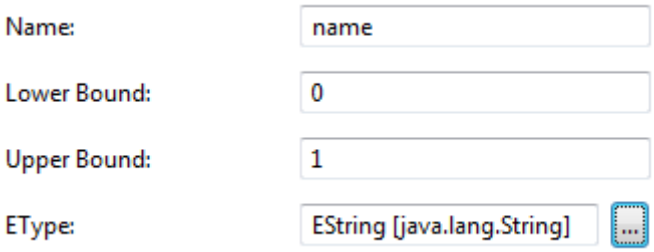
A questo punto ci si troverà davanti ad un foglio bianco, dove sarà possibile disegnare letteralmente il modello.

Mentre si edita questo file .ecorediag, il corrispondente file .ecore verrà automaticamente aggiornato. Si noti che il file.ecore è il file principale di EMF e che è possibile, anche, partire direttamente da esso (invece che dal Diagram) per gestire il modello. In questo tutorial si è deciso di esplorare l'utilizzo degli Ecore Tools ed avere quindi il file .Ecore automaticamente generato e sincronizzato in seguito all'editing del diagramma.

Iniziamo a costruire il modello selezionando un oggetto EClass dalla Palette e rinominiamolo in Author. Poi muoviamo il cursore sopra il box e selezioniamo l'icona corrispondente all'azione "Add EAttribute". Infine, rinominiamo l'attributo in name.

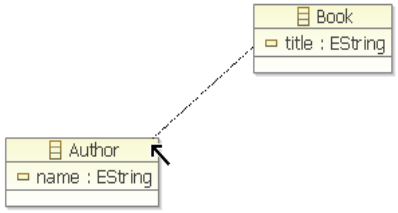


Tasto destro sull'attributo name e selezioniamo Show Properties View, andiamo, quindi, alla Properties View, per scegliere tra i tipi EString. In questo modo si avrà la seguente situazione nella Properties View:

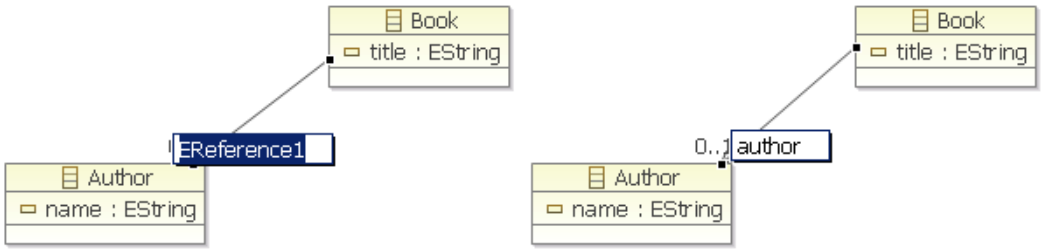


Lo stesso procedimento si farà per l'entità Book e il suo attributo String title.

Ora, per aggiungere all'entità Book un attributo con riferimento ad Author, clicchiamo sull'icona EReference sulla Palette. Clicchiamo, poi, sull'entità Book e rilasciamo il mouse sull'entità Author.

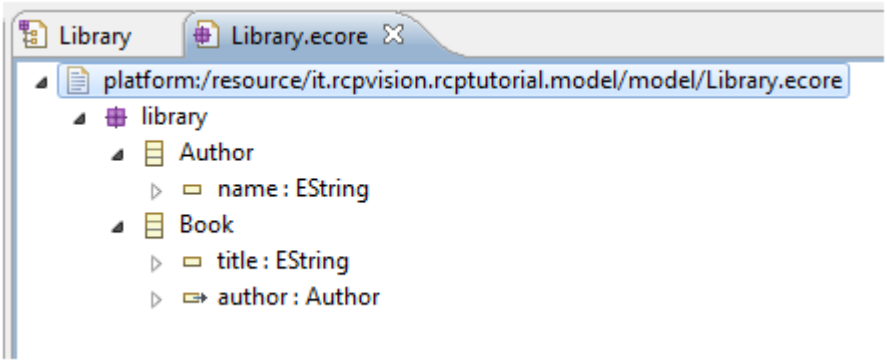


Infine, rinominiamo il riferimento come author:



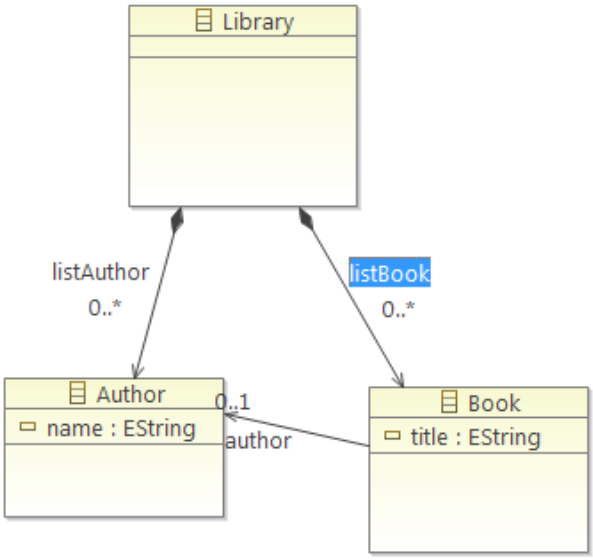
È stato così creato all'interno della classe Book, un attributo di tipo Author chiamato author con cardinalità 0..1.

Se ora si salvasse il file .ecorediag e apriamo il file .ecore, si vedrebbe questo:



Tuttavia, il modello non è finito. Si deve creare un'entità Library capace di gestire una lista di Author e una lista di Books. Torniamo, quindi, sul diagramma e creiamo

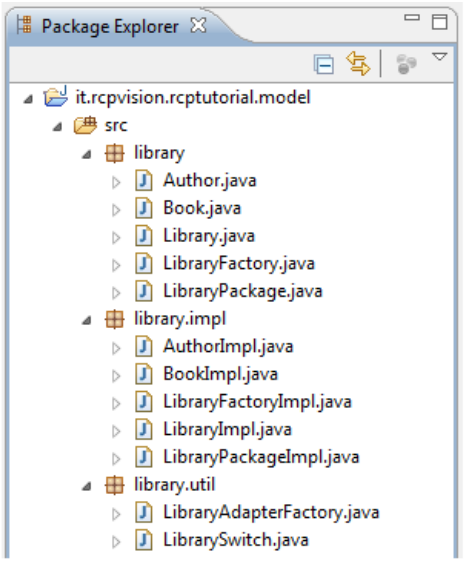
una EClass di nome Library. Successivamente, creiamo sulla Library un riferimento verso Author, chiamando il legame listAuthor. Se ci fermassimo ora, avremmo solamente inserito in Library un riferimento, al massimo, ad un singolo Author. Invece il riferimento è multiplo, una Library contiene una collection di oggetti Author, quindi inseriamo "*" nel campo Upper Bound. In realtà, è possibile anche mettere anche un numero intero per indicare il numero massimo di elementi Author; in ogni caso, se tale numero è superiore ad 1, tale attributo verrà gestito come una lista, altrimenti verrà semplicemente gestito come un attributo singolo. Un'ultima modifica a tale collection è, tuttavia, ancora opportuna: va marcata come "Is Containment", ciò significa che il ciclo di vita della collection creata è strettamente legata al ciclo di vita dell'oggetto Libray. La stessa cosa sarà fatta con Book.



Si potrebbe preferire di gestire il Modello EMF direttamente attraverso l'editing del file.ecore, è semplicemente una questione di preferenze.

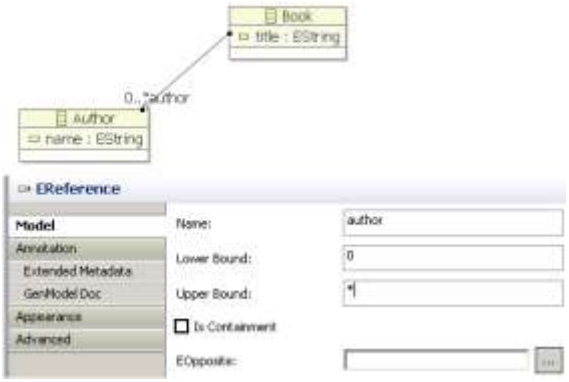
Ciò che conta è che si ha un Ecore Model pronto per essere utilizzato per generare il sorgente del Modello. Finora, infatti, non è stata scritta nemmeno una singola linea di codice, per questo motivo selezioniamo il file.ecore, e successivamente EMF Generator Model. In questo modo si è ottenuto un file Genmodel, dal quale è possibile generare il

sorgente del Modello, semplicemente selezionando Generate Model Code. È così che si ottiene il codice del modello, scritto interamente da EMF.



Osservando il codice generato ci si rende conto, effettivamente di quanto tempo ci sarebbe voluto per creare a mano tale codice. Solo per menzionare un aspetto, si consideri che questo codice contempla, fra le altre cose, un meccanismo di notifica associato ad ogni metodo setter.

Infine, EMF risulta essere efficace anche quando si tratta di modificare il Modello stesso. Supponiamo, ad esempio, che un'entità Book possa avere più di un Author, si ha, dunque bisogno di una Collection al posto di un semplice attributo. Per apportare tale modifica, avendo a diposizione EMF, basterà selezionare il legame "autore" da Book ad Author e cambiare il valore di Upper Bound da 1 a * (illimitato), e poi rigenerare per avere il codice del modello aggiornato.



4.2.5 Conclusioni

L'introduzione di EMF, come si è potuto capire, ha portato molti vantaggi, ad esempio:

- EMF permette di realizzare il modello di dominio esplicito che contribuisce a fornire una chiara visibilità del modello, fornisce, inoltre, funzionalità di notifica in caso di modifiche del modello;
- EMF genererà interfacce e factory per creare gli oggetti; quindi aiuterà a mantenere l'applicazione pulita dalle individuali implementazioni delle classi;
- il Modello può essere progettato da un non-sviluppatore (magari l'analista, che probabilmente ha una profonda conoscenza del dominio applicativo del Cliente, acquisita magari in decenni di esperienza, ma non conosce quasi per nulla un linguaggio di programmazione, per non parlare di Java);
- il codice del Modello viene generato automaticamente a tempo zero (o meglio: il tempo viene speso nella progettazione del Modello);
- il codice del Modello viene creato e mantenuto in sincronia col progetto del Modello automaticamente, quindi è potenzialmente ridotto a zero il rischio di scrivere codice non corretto;
- si può rigenerare il codice Java dal modello in un qualsiasi momento.

Esistono, però, anche degli svantaggi nell'utilizzo di EMF, quali:

- il codice automaticamente generato è unicamente Java;
- non vi è un concetto di repository, ovvero un ambiente in cui vengono gestiti i metadati, attraverso tabelle relazionali; l'insieme di tabelle, regole e motori di calcolo tramite cui si gestiscono i metadati prende il nome di metabase;
- non ci sono vincoli OCL. L'Object Constraint Language (OCL) è un linguaggio di specifica formale inizialmente proposto come estensione per il linguaggio di modellazione object-oriented UML e successivamente è entrato a far parte del nuovo standard del linguaggio (UML 2.0). Il nucleo di OCL può essere descritto come un linguaggio mutuato dal calcolo dei predicati del primo ordine per l'espressione di condizioni logiche inerenti allo stato e alle operazioni di oggetti in

un contesto *object-oriented*. Con la potenza del calcolo dei predicati, OCL consente di descrivere invarianti che legano il valore degli attributi di una classe, precondizioni e postcondizioni delle operazioni, e via dicendo. A partire dalla versione 2.0 il linguaggio è stato arricchito di elementi che consentono di descrivere la semantica di operazioni di quali le *interrogazione* (*query*), ovvero prive di effetti collaterali. Gran parte delle informazioni che si possono descrivere in OCL non sono esprimibili in nessun altro modo *formale* nel contesto di UML (ovvero non possono essere rappresentate dai diagrammi UML).

4.3 Spring Roo

Spring Roo è uno strumento estremamente utile per tutti gli sviluppatori Java EE che basano le loro applicazioni sulla famiglia di prodotti SpringSource.

È uno strumento RAD (Rapid Application Development) basato sul web che permette uno sviluppo rapido di applicazioni Java EE appoggiate allo SpringFramework. RAD è un concetto per cui i prodotti possono essere sviluppati velocemente (e quindi con un'alta produttività) attraverso:

- la raccolta di requisiti tramite workshop;
- la prototipazione, con test reiterati più volte sul disegno;
- il riuso dei componenti software;
- lo schedule rigido che permette di posticipare miglioramenti del design;
- la comunicazione tra team e poco formalismo.

In generale gli strumenti RAD permettono un:

- prototipazione veloce, che può essere anche reiterato più volte;
- test automatico (e continuo) su tutte le componenti software;
- riuso dei componenti software.

Spring Roo è basato anche sul principio *"Convention over Configuration"*, ciò vuol dire che lo sviluppatore deve specificare solamente gli aspetti "non convenzionali" dell'applicazione, lasciando gli aspetti più "convenzionali" a Roo che sa gestirli al meglio.

Per esempio, supponiamo di avere una tabella users sul database; allora il corrispondente entity bean si chiamerà per convenzione Users, e questa assunzione viene fatta da Roo senza alcuna specifica.

4.3.1 Potenzialità

Spring Roo è uno strumento che permette di generare automaticamente un progetto Java EE basato su Spring, di effettuare il reverse-engineering dal DB per ottenere, gli entity beans, le funzioni CRUD, le funzioni di ricerca, di generare automaticamente i test di verifica ed, infine, di generare automaticamente un CRUD web basato su JSP. In pratica, tutto questo consente di poter creare e ricreare un progetto web Java EE partendo da un database in meno di 10 minuti. Il progetto risultante (possibilmente modulare) è basato su Maven, un tool collaborativo di gestione dei progetti, quindi, il layout dei sorgenti è basato sulle convenzioni Maven. Il progetto, inoltre, potrà essere testato in tutte le sue parti. Spring Roo permette, infine, di riutilizzare molte componenti software open source (Spring, Hibernate, GWT, etc...) in maniera "convenzionale", cioè in modo rigido e ben compreso da tutti. È interessante notare che Roo non è dotato di un run-time environment: di conseguenza non entra in gioco quando il programma è in esecuzione.

Nel mondo web, Spring Roo è solo uno degli ultimi arrivato tra i RAD che permettono di automatizzare la creazione e stesura di un web-CRUD. La differenza che vi è tra Roo e gli altri tool risiede principalmente nel fatto che esso è basato sul framework Spring, di conseguenza Roo è ideale per chi utilizza la suite Spring per i propri progetti Java EE. Un ulteriore punto di forza di Roo è che non interferisce in alcun modo durante l'esecuzione del programma risultante: infatti tutto il codice è generato da Roo a tempo di compilazione, e non a tempo di esecuzione.

4.3.2 Struttura di un progetto basato su Spring Roo

Un progetto basato su Spring Roo ha una struttura rigida e ben definita. Anzitutto si tratta di un progetto basato su Maven, possibilmente modulare, inoltre è multi-layer e orientato a servizi. Un tipico progetto multi-layer orientato a servizi è formato dai seguenti layer:

1. Persistence Layer: è il layer che accede al database; inoltre i database a cui si accede possono essere di natura diversa: SQL o anche no-SQL;
2. Service Layer: è il layer in cui si implementano i servizi business del progetto Java EE;
3. Consumer Layer: è il layer, dove i servizi esposti nel layer precedente sono utilizzati da delle pagine web (JSP, JSF, GWT...).

La *Figura 21* descrive un tipico progetto Java EE multi-layer:

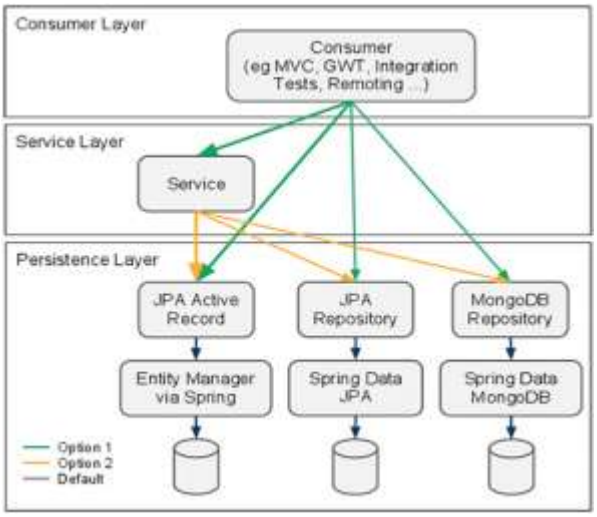


Figura 21. La struttura multi-strato di un tipico progetto Java EE in Spring Roo

Lo strato dei servizi è stato introdotto in Spring Roo per dare l'opportunità agli sviluppatori di avere un proprio layer dove implementare le proprie logiche di business (per esempio è in questo layer che è indicato implementare un servizio che esponga metodi via SOAP).

L'ultimo layer, il consumer layer, è quello utilizzato per esportare le funzionalità del progetto via web. L'insieme di tecnologie web a nostra disposizione è principalmente il seguente:

- **JSP**: Spring Roo è stato sviluppato principalmente per supportare questa tecnologia, quindi le JSP (con utilizzo di Spring MVC e Apache Tiles) sono supportate in maniera ottimale;
- **JSF**: sono state introdotte di recente, per ora è supportata solo la libreria PrimeFaces, ma comunque in generale il supporto JSF appare ancora immaturo;
- **GWT**: anche per GWT vale lo stesso discorso fatto per le JSP: il supporto dato è sicuramente soddisfacente.
- **Flex**: anche la tecnologia Adobe Flex è supportata tramite il progetto Spring-Flex.

4.3.3 Primi passi con Spring Roo

Spring Roo è una shell interattiva da utilizzare per dare dei comandi di creazione e configurazione del progetto. La shell ha un notevole sistema di help che permette di:

- visualizzare tutti i comandi disponibili;
- suggerire il comando più appropriato da dare;
- visualizzare l'help per ogni comando;
- suggerire l'installazione di add-ons per effettuare alcune operazioni (ad esempio, installazione di driver per database forniti da terze parti).

Inoltre, essa può anche comportarsi in maniera non interattiva. È possibile, infatti, dare uno script di comandi e Roo li esegue senza intervento del programmatore. Quest' opportunità è molto utile per poter rigenerare dei progetti velocemente. In pratica, la shell Roo scrive su un file di log (roo.log) tutti i comandi che ha eseguito nella sessione, quindi se si volesse definire un progetto in funzione di uno script Roo sarebbe possibile farlo in modo molto veloce.

Un aspetto molto importante e da non sottovalutare è che usando Roo utilizziamo un modo condiviso di gestire i file di risorse, log4j.properties e applicationContext.xml, in modo che il progetto sarà facilmente comprensibile anche da altri sviluppatori, senza alcun intervento da parte nostra. L'utilizzo di Maven, inoltre, garantisce la portabilità del

progetto Java. Un progetto Maven, infatti, è utilizzabile in qualsiasi IDE (Netbeans, Eclipse, IntelliJ IDEA) e non memorizza alcun path assoluto delle librerie di terze parti, quest'ultime sono tutte scaricate correttamente dal loro sito di origine con le versioni e le dipendenze correttamente specificate.

Creato il progetto, si nota subito che mancano completamente i metodi getter e setter di un entity bean. A questo punto sembrerebbe che Roo sia un framework che crea classi e metodi e li rende inaccessibili all'utente. In realtà, come detto in precedenza, Roo non è dotato di un runtime environment, quindi a runtime non entra in gioco: di conseguenza tutto il codice deve per forza essere generato a tempo di compilazione.

Quindi, tutto il codice "mancante" viene generato a tempo di compilazione da AspectJ (uno dei modi utilizzati, in ambito Java, per avvalersi dell'Aspect Oriented Programming) utilizzando le inter-type declarations, chiamate anche mix-ins; questi ultimi sono file sorgenti che vengono compilati da AspectJ e che generano il codice mancante.

Pertanto, il codice "nascosto" da Roo è solamente scritto sotto forma di regole comprese dal compilatore AspectJmediante le quali generano i metodi richiesti. Questo codice (creato automaticamente) è la parte più noiosa e ripetitiva, quindi è bene che sia nascosto e generato da un RAD quando si compila.

4.3.4 Un esempio di utilizzo

Spring Roo è uno strumento di sviluppo, non un framework. Anche se il nome di " Spring" fa spesso pensare che si tratti di un framework, esso, invece, è uno strumento di creazione di codice. Un progetto in Java non fornisce alcuna libreria da utilizzare durante l'esecuzione, né richiede un particolare stile di codifica o di architettura, pertanto, non può essere considerato come un framework. Le applicazioni che, invece, possono essere sviluppate utilizzando Spring Roo sono fondamentalmente applicazioni basate su Spring in Java (web), oppure applicazioni GWT, o ancora applicazioni Google App Engine.

Analizziamo nello specifico un esempio per vedere come si possa accelerare il processo di sviluppo attraverso l'utilizzo di Spring Roo.

Iniziamo con il creare un nuovo progetto Roo. Si noti che le prime versioni di Spring Roo potevano essere trattate solo da console, ma con l'introduzione del pacchetto STS è stata aggiunta una funzione di supporto per cui non c'è più bisogno di lavorare da console. Infatti, una volta creato il progetto, esso sarà visualizzato nel Package Explorer e nella parte inferiore di STS comparirà la Roo Shell.

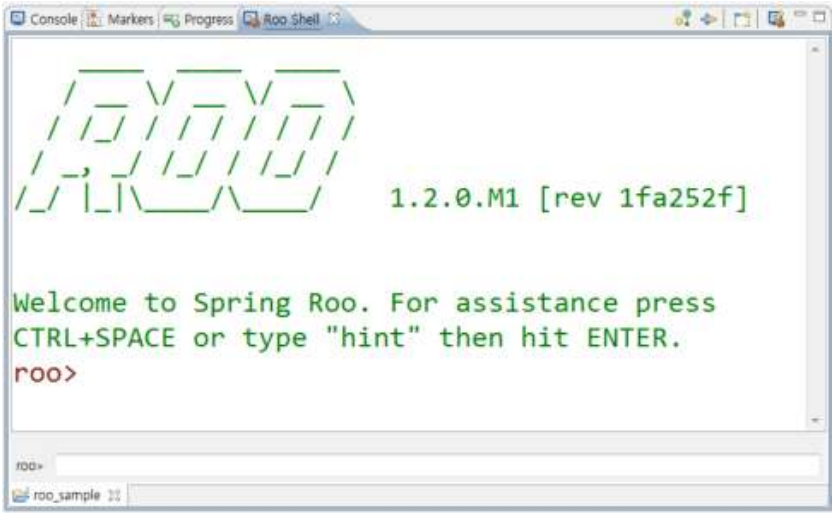


Figura 22. Shell di Roo

È questa la view che ha sostituito la console. Utilizzando il DSL fornito da Roo, è possibile effettuare diverse operazioni qui, quali la configurazione JPA, la creazione delle classi entità, l'aggiunta di un campo per ogni entità, la creazione di un controller e la creazione di test.

Il progetto creato ha la normale struttura di un progetto Maven, e fin da subito sarà possibile consultare il file applicationContext.xml, un file di configurazione di Spring. La creazione di un progetto Maven, la configurazione base di Spring e la compilazione del file pom.xml richiede molto tempo senza l'utilizzo di Maven Archetype. Si ricorda che Maven è un software usato principalmente per la gestione di progetti Java e build automation. Per funzionalità è similare ad Apache Ant, ma basato su concetti differenti. Il progetto Maven è ospitato da Apache Software Foundation, dove faceva parte dell'ex progetto Jakarta. Maven usa un costrutto conosciuto come Project Object Model (POM); un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse. In questo modo si separano le

librerie dalla directory di progetto utilizzando questo file descrittivo per definirne le relazioni. Maven, infine, effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti scaricandoli in locale o in un repository centralizzato lato sviluppo. Questo permette di recuperare in modo uniforme i vari file JAR e di poter spostare il progetto indipendentemente da un ambiente all'altro avendo la sicurezza di utilizzare sempre le stesse versioni delle librerie.

Diamo ora un'occhiata al file di configurazione del bean Spring.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<beans xmlns="http://www.springframework.org/schema/beans" …>
    <context:property-placeholder location="classpath*:META-
INF/spring/*.properties"/>
    <context:spring-configured/>
    <context:component-scan base-package="whiteship">
        <context:exclude-filter expression=".* Roo .*" type="regex"/>
        <context:exclude-filter-
expression="org.springframework.stereotype.Controller"
type="annotation"/>
    </context:component-scan>
</beans>
```

In tal file, la parte più importante correlata a Spring Roo è: `<context:spring-configured/>`. Utilizzando, infatti, questa particolare configurazione si permette alla *Spring dependency injection* di poter essere utilizzata su oggetti non registrati come beans nell'ApplicationContext. Non è un compito semplice inserire la gestione della Spring *dependency* in un oggetto non registrato come bean. Ad esempio, se un oggetto "Book" è stato creato (`Book book= new Book ();`) in un punto qualsiasi del codice, significherebbe che un bean di tipo BookRepository si riferirà a questo oggetto. Per fare ciò, un task supplementare deve essere eseguito al momento della creazione della classe book, ma questo compito non può essere effettuato da Spring AOP. Questa operazione, infatti, è possibile solo utilizzando AspectJ, che fornisce tutte le funzionalità AOP. Il generatore di codice non fa altro che chiamare il metodo Joinpoint per AspectJ, esso sarà utilizzato per inserire un bean di tipo BookRepository nell'ApplicationContext di Spring utilizzando la funzione di *dependency injection* nel momento in cui è creata una classe book.

Fino ad ora, Spring Roo ha insistito sull'utilizzo del pattern ActiveRecord, introdotto nei modelli di Enterprise Application Architecture (PEAA) secondo il quale:

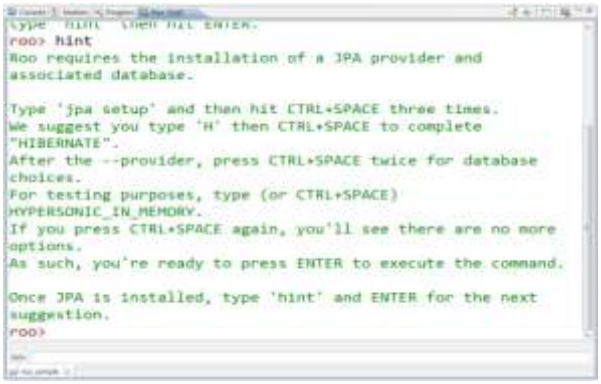
- una tabella di un database relazionale è gestita attraverso una classe
- una singola istanza della classe corrisponde ad una riga della tabella
- alla creazione di una nuova istanza viene creata una nuova riga all'interno della tabella, e modificando l'istanza la riga viene aggiornata

La classe è descritta in buona parte dalla tabella che rappresenta ed espone i metodi necessari per leggere e scrivere i record. Le colonne della tabella sono rappresentate come attributi della classe e ogni istanza è in grado di effettuare operazioni "su sé stessa" come salvarsi, cancellarsi, aggiornare il valore dei propri attributi e renderli persistenti nel database. Quindi, si può creare un'istanza di un oggetto, modificarne gli attributi e rendere persistente il dato operando esclusivamente con un oggetto e lasciare che ActiveRecord faccia il resto.

Inoltre, il pattern Active Record è scelto peri i suoi vantaggi nello sviluppo di applicazioni. Dal momento che tutte le richieste vengono semplicemente elaborate da chiamati semplici come "Controller - > Domain Object ". Eppure, molti hanno sottolineato che è un possibile difetto di questo modello è la difficoltà di effettuare i test, in quanto il metodo statico, che non è preferito da Spring, viene utilizzato per implementare il finder, la posizione per impostare i limiti delle transazioni è ambigua, e ci sono un certo numero di dipendenze inserite all'interno delle entità oggetto. A causa di tutto ciò, il pattern Active Record è utilizzato in modo selettivo.

Arrivati a questo punto, ci si può occupare della configurazione JPA. Per farlo basterà digitare nella shell il seguente comando:

```
jpa setup --provider HIBERNATE --database HYPERSONIC_IN_MEMORY
```



Non è stato inserito manualmente alcun comando dopo aver digitato "jpa setup", Infatti, dopo la sua scrittura è bastato selezionare tra i comandi suggeriti. In questo modo, è possibile immettere comandi lunghi senza memorizzare nulla. Si noti che la digitazione di questo comando comporta:

- l'attivazione delle dipendenze per hsqldb1.8.0.10 e Hibernate 3.6.7;
- la correzione dei contenuti nel file applicationContext.xml;
- la creazione del file persistence.xml.

Per svolgere tutte queste operazioni manualmente si sarebbe impiegato ben più di qualche minuto. Tutti questi compiti, invece, sono stati completati in pochissimi secondi, questa è la bellezza dello sviluppo rapido offerto da Spring Roo.

Nell'applicationContext.xml sono registrati i bean "DataSource", "TrasactionManager", "tx:annotation-driven" e "EntityManagerFactory", mentre la configurazione di Hibernate, che è una implementazione JPA, è stata aggiunta nel file persistence.xml. Per la configurazione del "DataSource" è stata utilizzata la registrazione context:property-placeholder, nel momento in cui è stato creato il progetto, per spostare il valore di configurazione del bean verso l'esterno, come se fosse un file di proprietà. Quindi, se si desidera cambiare il DB con un altro, bisognerà modificare la configurazione in modo tale che essa sia compatibile con quello che sarà utilizzato.

Quando si crea un'entità, è possibile selezionare uno dei due tipi di architettura precedentemente citati. In questo esempio, si userà l'architettura che è spesso utilizzata anche in Java. Il comando da digitare è il seguente:

```
entity --class ~.domain.Book --activeRecord false
```

L'intero comando verrà composto in modo automatico, anche se bisognerà immettere manualmente "~. Domain.Book". Il carattere "~" sta ad indicare il pacchetto di base configurato al momento della creazione del progetto in Spring Roo. È possibile selezionare anche le opzioni relative all' ActiveRecord dall' elenco dei comandi a disposizione, basterà, infatti, digitare "Ctrl + Spazio" per avere a disposizione l'intero elenco dei valori di opzione dell' ActiveRecord, noi selezioneremo, ad esempio, il valore "false".

Come sappiamo è stata creata una classe di dominio chiamata "Book", insieme a due file AspectJ. Guardiamo il codice creato:

```
package whiteship.domain;
import org.springframework.roo.addon.entity.RooJpaEntity;
import org.springframework.roo.addon.javabean.RooJavaBean;
import org.springframework.roo.addon.tostring.RooToString;
@RooJavaBean
@RooToString
@RooJpaEntity
public class Book {
}
```

Dalla lettura del codice sembra essere una classe vuota con tre annotazioni allegate ad essa, eppure in questa classe, toString () e getter() / setter() sono pronti per essere auto-generati. Le annotazioni @RooJavaBean e @RooToString ricoprono proprio questo ruolo. Invece, @RooJpaEntity aggiunge l'ID necessaria, le variabili membro versione e tutto ciò che è necessario per l'entità JPA. Così, dopo aver creato il metodo principale all'interno della classe Book e l'oggetto book, le funzioni setId / getID e setVersion / getVersion possono a questo punto possono essere verificate.

```
@RooJavaBean
@RooToString
@RooJpaEntity
```

```
public class Book {
    public static void main(String[] args) {
        Book book = new Book();
        book.setId(1l);
        book.getVersion();
    }
}
```

È stato creato il metodo principale come un test, quindi eliminiamo i dati specifici e aggiungiamo un campo. Per far ciò si digiti, quanto segue nella shell di Roo.

```
field string --fieldName name --class whiteship.domain.Book
```

Ora i codici della classe Book potranno essere cambiati come riportato di seguito:

```
@RooJavaBean
@RooToString
@RooJpaEntity
public class Book {
    private String name;
}
```

Naturalmente, questo è abbastanza semplice e può essere codificato direttamente senza l'utilizzo della shell di Roo, ma si è voluto mostrare com'è possibile utilizzare questa importante funzionalità di Spring Roo.

Tutti i comandi visti finora possono anche essere raccolti in uno script in modo tale che possano essere riutilizzati in un qualsiasi momento. Dopo aver raccolto i comandi di Roo, infatti, basterà creare un file script e salvarlo nel formato *"sample.roo"* e per eseguire gli script si utilizzerà il comando "*script – file filename*".

Il passo successivo del nostro esempio sarà la creazione di una repository.

Si esegua il seguente comando nella shell di Roo:

```
repository jpa --interface ~.modules.book.BookRepository --entity ~.domain.Book
```

```
@RooRepositoryJpa(domainType = Book.class)
    public interface BookRepository {
}
```

In questo momento, le operazioni CRUD e di accesso ai dati per la ricerca sono implementate. Le funzioni che devono quasi sempre essere attuate possono essere implementate utilizzando Spring Data JPA. Per quanto riguarda, invece, la realizzazione di un servizio, il comando da digitare nella shell di Roo sarà:

```
service --interface ~.modules.book.BookService --entity ~.domain.Book
```

Si dovrebbe, oramai, aver acquisito una certa familiarità con i comandi Roo. Probabilmente si è anche in grado di prevedere, in una certa misura, i codici che ne costituiscono il risultato, ma come previsto, non ci sarà mai molto codice.

```
@RooService(domainTypes = { whiteship.domain.Book.class })
public interface BookService {
}
public class BookServiceImpl implements BookService {
}
```

Con quest'ultimo codice la realizzazione dell'entità, la repository e il servizio sono conclusi.

Attualmente, una colonna web per Roo può essere implementata utilizzando tre tecnologie. I codici possono essere generati con GWT, Spring Web Flow e Spring MVC. In quest'esempio si è scelto di generare i codici con Spring MVC, per questo eseguiamo il seguente comando nella shell di Roo.

```
web mvc setup
```

In questo modo si aggiunge la dipendenza necessaria per Spring MVC, si aggiungere la configurazione web di Spring e si creano le varie risorse necessarie per la view di base. Molte altre caratteristiche sono aggiunte in questo momento. È possibile controllare il file webmvc-config.xml per vedere quali caratteristiche sono state aggiunte. Ora, cerchiamo di generare il codice del controller, eseguiamo quindi il seguente comando nella shell di Roo.

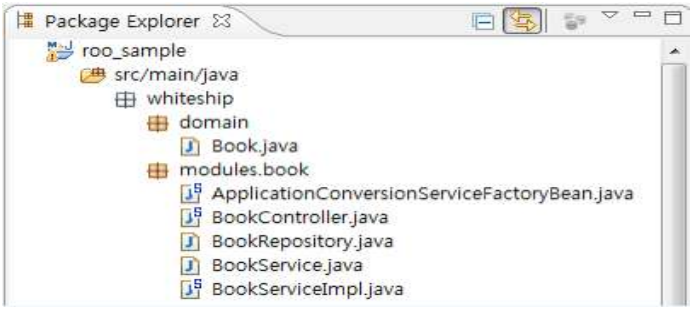
```
web mvc scaffold--class~.modules.book.BookController--backingType~.domain.Book--path/bo
```

La riga di comando è lunga, ma se si utilizza *"Ctrl + Space"*, non si dovrebbe avere alcun problema ad immettere l'intero comando. Una volta eseguito si è pronti per lanciare l'applicazione web. Ci sono diversi modi per eseguire un'applicazione web da STS. Si può, infatti, vedere che nel file *pom.xml* sono stati installati i plugin Tomcat e Jetty, ma è anche possibile eseguire l'applicazione utilizzando il plugin Maven.

Una volta avviato il server, basterà inserire l'indirizzo http://localhost:8080 nel browser, e si vedrà il link per collegamento all' applicazione di esempio. Naturalmente, è possibile

anche inserire l'intero indirizzo: http://localhost:8080/project_name per lanciare direttamente l'applicazione.

Ecco i codici Java generati finora.



In questi pochi file ci sono poche righe di codice, ci si chiede, quindi, come sia possibile sia le funzioni CRUD di base possano lavorare con così poche annotazioni. Se si dispone di esperienza nella produzione di strutture che offrono queste opzioni, si dovrebbe sapere che queste funzioni non sono poi così complicate. In realtà è possibile anche utilizzare un diverso framework che fornisce le funzioni CRUD di base usando GenericDao, GenericService, GenericController, e dichiarando esclusivamente la classe e l'interfaccia. Esso può, sicuramente presentare il vantaggio di creare le funzioni CRUD velocemente, ma il grande svantaggio lo si potrebbe avere per la presenza dell' eccessiva "ereditarietà", in quando la descrizione delle classi e delle interfacce si potrebbe complicare troppo e la leggibilità del codice potrebbe drasticamente diminuire. Spring Roo risolve questo problema molto egregiamente. In AspectJ, c'è una funzionalità denominata "Dichiarazione Inter-Type (ITD)," nota anche come "Mixin". Questa caratteristica non è un concetto esistente solo in AspectJ, ma un concetto che è comunemente utilizzato in un linguaggio dinamico. Come Java, supporta solo l'ereditarietà singolare, non può ereditare più caratteristiche comuni. In Mixin, d'altra parte, le caratteristiche sono create per essere utilizzate da più oggetti. Queste funzionalità possono essere aggiunte inserendole direttamente negli oggetti stessi. Spring Roo utilizza la funzionalità ITD. Utilizzando ITD non è più richiesta l'ereditarietà. Dal momento che i codici generati dall'utente e codici forniti da Spring Roo possono essere totalmente separati, è ancora possibile generare codici con Spring Roo e dopo la generazione modificarli. È possibile nascondere i codici

di base che sono generati con ITD di Spring Roo, e lasciare solo i codici relativi alla logica di business.

Infine, anche i file di AspectJ sono creati anche con Spring Roo, e tali codici sono fondamentalmente nascosti. Questo per permettere di concentrarsi solo sui codici relativi alla logica di business, ma è possibile anche modificare le funzioni CRUD di base se è necessario. I file AspectJ possono essere trovati semplicemente nel menù del Package Explorer.

4.3.5 Conclusioni

Spring Roo è uno strumento molto importante per tutti coloro che utilizzano il framework Spring per implementare i propri progetti Java EE. Il motivo principale per utilizzare Roo è il concetto di Convention over Configuration: tramite Roo si può scrivere un server Java EE basato su Spring seguendo lo "standard di Spring" in maniera rigida. Il vantaggio di tutto ciò è avere un progetto maggiormente manutenibile e comprensibile da qualsiasi sviluppatore Spring. Il secondo motivo per usare Spring Roo è la velocità con cui è possibile mettere in piedi da zero un progetto Java EE. Spring Roo permette di costruire velocemente lo scheletro di un progetto web risolvendo i tipici problemi da affrontare. Se si utilizza il plugin per le JSP, per esempio, Roo configura una servlet login in https, realizzando automaticamente tutto il codice che serve per la parte di autenticazione e autorizzazione. Infine, Roo non rallenta ne’spreca memoria usata dal programma a run time, semplicemente perche’ gli aspetti vengono risolti a tempo di compilazione da AspectJ, e non a run time.

4.4 SkyGen

SkyGen è una soluzione integrata per la generazione di applicazioni J2EE, ed è basato sul principio del Model Driven Development (MDD).

4.4.1 Il Model Driven Development

Spesso, l'inizio dello sviluppo di una nuova soluzione software è un compito gravoso. In genere, dopo una serie di approvazioni da parte della gestione, alcuni requisiti di alto livello sono consegnati ad un'organizzazione di sviluppo per definire e creare la soluzione. A questo punto del ciclo di vita del prodotto, le aziende, spesso, decidono per uno dei due seguenti approcci:

- spendere il tempo per disegnare la soluzione, senza creare gli artefatti che entreranno a far parte della soluzione
- iniziare la codifica dei requisiti funzionali, senza definire formalmente come sarà strutturata la soluzione.

Ogni approccio contiene i propri rischi e i propri benefici. Perdendo troppo tempo, inizialmente, ad analizzare la soluzione, si provoca la perdita di profitti ed eventualmente l'annullamento dello sforzo. Vi è, inoltre, il rischio che i problemi connessi con la realizzazione dell'architettura definita potrebbero richiedere che la soluzione prenda un percorso diverso, rendendo l'esercizio fatto una perdita di tempo.

Ci sono anche molti rischi nel dare troppa poca considerazione alla struttura complessiva della soluzione, in particolare, a come il prodotto potrebbe essere esteso e mantenuto anche in futuro. Qualsiasi soluzione che abbia dimensioni e complessità ragionevoli richiede, di solito, prima una fase di analisi e di progettazione per comprendere come la soluzione deve essere realizzata. Senza alcuna linea guida iniziale, una soluzione, infatti, potrebbe richiedere una profonda “ristrutturazione” nelle versioni successive. Inoltre, una soluzione tecnicamente complessa potrebbe comportare una base di codice ingestibile, che sarà difficile da mantenere, nel momento in cui quanto il progetto si evolverà.

Infine, anche per lo sviluppo di Application Services, si è di fronte ad innumerevoli problemi. Si parla, infatti, in tal contesto di mancanza di soluzioni standardizzate, che comporta maggiori costi di manutenzione e di supporto per le applicazioni realizzate, di scarsa agilità di sviluppo con conseguente perdita di business e ricavi, di poco riutilizzo

con conseguente aumento dei costi di consegna e delle scadenze, di interruzioni dovute alla scarsa qualità del software e conseguenti penalità delle prestazioni.

Negli ultimi anni, con l'evoluzione degli strumenti e delle tecnologie, si è evoluta anche una terza opzione, la definizione l'architettura di un software soluzione, il **Model Driven Development (MDD)**. MDD offre agli architetti la capacità di definire e comunicare una soluzione durante la creazione di artefatti che diventeranno parte della soluzione globale.

"Model Driven Development = standards, models, iterative development, transformations, patterns, tools, best-practices/guidance and reuse"

Model- driven development (MDD) è, dunque, un nuovo paradigma di sviluppo software supportato e guidato dalla metodologia Modeldriven Architecture (MDA), approccio di progettazione software rilasciato da Object Management Group (OMG). MDA fornisce una serie di linee guida per le specifiche di strutturazione espresse come modelli, a partire da un modello indipendente dalla piattaforma (PIM), per poi passare alla scelta di un linguaggio specifico del dominio e concludersi con la trasformazione in uno o più modelli specifici della piattaforma (PSM) per la piattaforma di implementazione. Questo vale per un qualsiasi piattaforma come, ad esempio: Java ™ 2 Platform, Enterprise Edition (J2EE ™), CORBA, o .Net, che sono implementate con l'utilizzo di un linguaggio di programmazione generale, come Java ™, C# e Python.

MDA viene normalmente eseguito con strumenti automatizzati. MDD, invece, guidato da MDA, si concentra di più sulla trasformazione del modello e sulla generazione di codice.

Una delle caratteristiche principali del MDD è usare i modelli come chiave degli artifacts. Un modello è la descrizione di un sistema da una particolare prospettiva, omettendo i dettagli irrilevanti in modo tale che le caratteristiche di interesse siano viste più chiaramente. Nel MDD, il modello deve essere leggibile da una macchina, in modo che sia possibile accedere al contenuto del modello in modo automatico.

I modelli sono in genere realizzati mediante diagrammi UML. Essi, infatti, nascondono i dettagli tecnici di implementazione, in modo che un sistema possa essere progettato utilizzando solamente i concetti del dominio dell' applicazione. È importante sottolinea,

però, che l'utilizzo di modelli UML per la progettazione di software era una prassi consolidata anche prima dell'introduzione del MDD.

Nella maggior parte dei casi, i modelli erano utilizzati in due modi:

- come schizzi, che informalmente descrivevano alcuni aspetti di un sistema
- come modelli, che descrivevano un progetto dettagliato da implementare manualmente.

Nel Model Driven Development i modelli sono utilizzati non solo come schizzi, ma, piuttosto, come artefatti primari da cui sono generate implementazioni efficienti applicando delle specifiche trasformazioni. Da ciò si comprende come i modelli applicativi di dominio descritti in modo preciso sono l'obiettivo primario nello sviluppo di nuovi componenti software.

Il codice è generato utilizzando le trasformazioni progettate con il controllo da parte di esperti sia di modellazione, che del dominio di destinazione.

La modellazione UML è una tecnica valida di per sé, ma la sincronizzazione tra modelli e codice è soggetta ad errori e richiede molto tempo. La trasformazione è la caratteristica principale che differenzia il Model Driven Development dagli altri approcci che utilizzano la modellazione. Inoltre, esso si occupa di automatizzare il processo di sviluppo in modo da generare qualsiasi codice, che può essere derivato dalle informazioni presenti in un modello. Oltre al codice e agli artefatti, dalla generazione si otterà anche:

- *la documentazione*. Nelle organizzazioni che seguono un approccio di sviluppo formale, produrre la documentazione richiede una notevole quantità di sforzo, così come il mantenere la documentazione in linea con lo sviluppo. Quando si utilizza MDD, invece, i documenti vengono generati dai modelli stessi, ciò garantisce la coerenza e rende le informazioni disponibili all'interno dei modelli, piuttosto che in documenti difficili da navigare.
- *altri modelli*: un sistema coinvolge molti modelli interdipendenti a diversi livelli di astrazione (analisi, progettazione, implementazione), che rappresentano le diverse parti del sistema (utente, database, logica aziendale, amministrazione di sistema),

diverse preoccupazioni (sicurezza, prestazioni e resilienza), o diversi compiti (test, modellazione e distribuzione). In molti casi, è possibile generare un modello parzialmente da un altro (per esempio, passando da un modello di analisi di un modello di disegno, o da un modello di applicazione di un modello di prova).

Un' altra importante caratteristica del MDD è quella di acquisire competenza.

I modelli catturano le soluzioni pratiche migliori ai problemi ricorrenti e possono ritrovarsi a tutti i livelli di astrazione della modellazione. Per esempio, si hanno:

- modelli di architettura
- modelli di progettazione
- modelli di implementazione

Dunque, i modelli possono essere composti per risolvere problemi più grandi ed avere le "best- practices" per un' intera zona del dominio applicativo. Essi, inoltre, specificano sia le caratteristiche degli elementi del modello, che le relazioni tra di essi. È possibile automatizzare i modelli in modo che i nuovi elementi creati, e gli elementi esistenti modificati siano conformi al modello, anche quando il modello viene applicato ad un ulterire modello.

Dunque, i patterns migliorano la consistenza e la qualità delle soluzioni. Piuttosto che ripetutamente e manualmente, l'attuazione delle competenze tecniche per la costruzione della soluzione è codificata direttamente nelle trasformazioni. Questo ha il doppio vantaggio di coerenza e di manutenibilità.

La *Figura 23* mostra come un problema di business si traduce con l'utilizzo di MDD in una soluzione software.

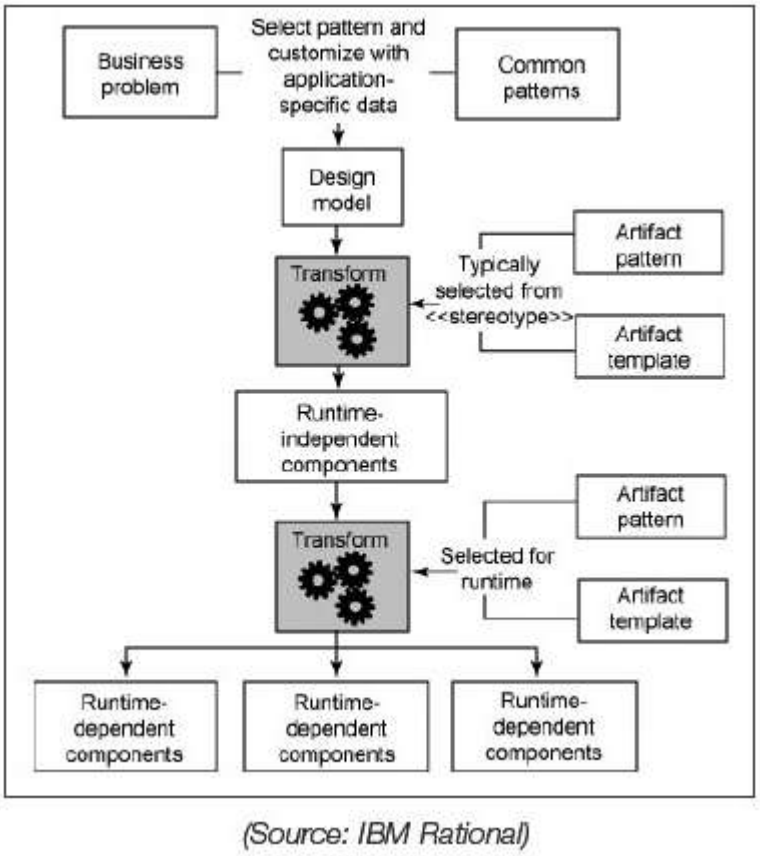


Figura 23. Trasformazione con MDD di un problema di business

Per comprendere meglio l'utilizzo di questo approccio di sviluppo, analizziamo i tre principali aspetti architettonici che devono essere definite per i requisiti funzionali, prima dello sviluppo, ovvero:

1. Confini
2. Struttura
3. Modello di dominio

Prima di creare codice, l'architetto deve capire l'intento della soluzione, e se la soluzione proposta si può inserire nel contesto del problema di business che si sta risolvendo. In particolare, l'architetto dovrebbe avere un'idea di quali oggetti agiscono come input per una soluzione, e quali sono necessari per la soluzione, per soddisfare i requisiti di business. L'architetto deve, anche, capire il flusso di base delle operazioni di business, come se le operazioni dovessero essere eseguite manualmente. La comprensione dei

confini della soluzione dà all'architetto un obiettivo da raggiungere attraverso la progettazione, essa consente anche di capire la portata della soluzione. Tutto ciò è il primo passo per la definizione della struttura della soluzione.

Come sarà strutturata la soluzione è una delle decisioni più critiche che l'architetto deve fare. Senza definire la struttura di una soluzione e le politiche che delimitano tale struttura, la soluzione può diventare ingombrante. La soluzione può soddisfare i requisiti aziendali senza la necessaria coerenza che permetterà, in futuro, un facile ampliamento o una semplice manutenzione. Uno dei primi artefatti da creare è lo schema dell'architettura concettuale. Essa definisce come saranno soddisfatti i requisiti funzionali all'interno della struttura definita per la soluzione. La definizione dell'architettura concettuale dovrebbe comprendere come la soluzione è esposta ai processi esterni, come gli utenti interagiscono con la soluzione, come le operazioni saranno mantenute durante tutta la soluzione, come sarà accessibile e manipolato il dominio della soluzione, e come la soluzione gestisce gli eventi internamente ed esternamente.

Utilizzando uno strumento di modellazione che consente all'architetto di definire l'architettura concettuale, e la costruzione di un thread di lavoro per la realizzazione della soluzione può ridurre al minimo qualsiasi ambiguità per l'attuazione dell'architettura, e può esporre eventuali problemi tecnici che potrebbero sorgere.

Come detto, l'architettura concettuale è un primo importante passo nella realizzazione di una soluzione, ma è solo un primo passo. Considerando l'architettura concettuale al punto in cui la soluzione può essere sviluppata con un risultato estensibile e gestibile, richiede la definizione nel dettaglio del dominio della soluzione. Il modello di dominio contiene oggetti che sono essenziali per il soddisfacimento dei requisiti funzionali della soluzione e definisce, anche, i rapporti tra gli oggetti. Questa parte della definizione della soluzione richiede che sia controllato l'accesso agli oggetti di dominio, e siano coordinate le transazioni attraverso il dominio. Il risultato della modellazione di dominio può includere: quadri, modelli di oggetti e definizioni di servizi.

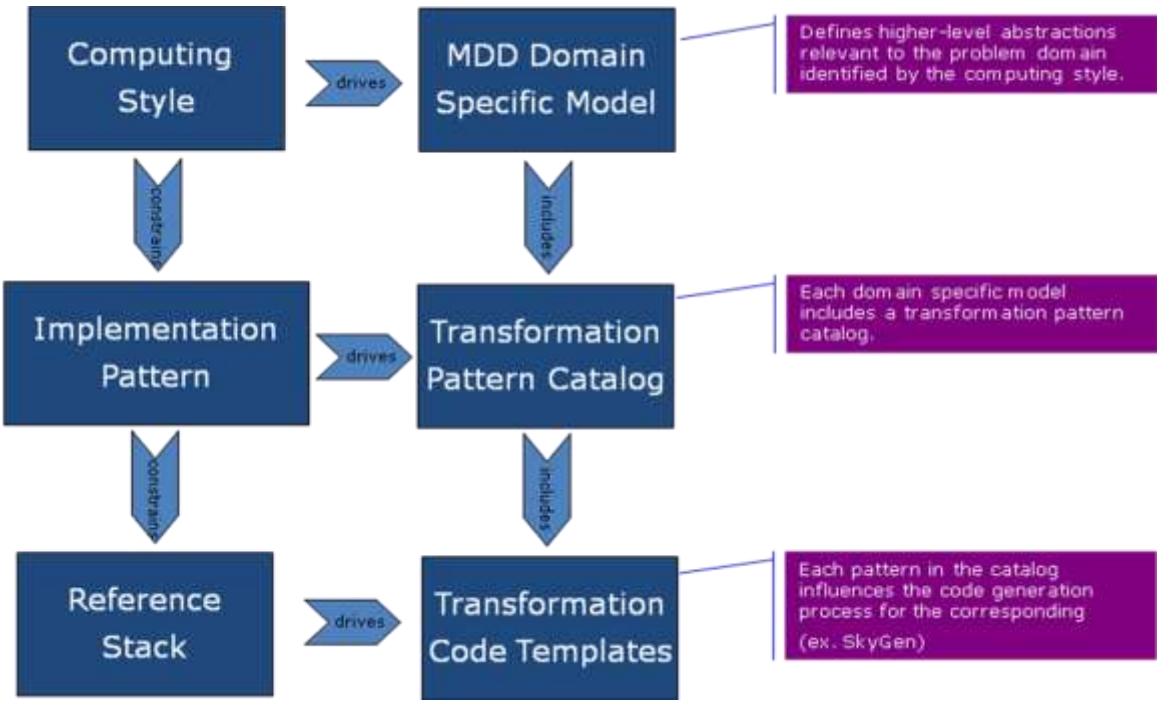


Figura 24. Architettura MDD

L'industria dello sviluppo di software ha riscontrato numerosi benefici adottando il MDD. Fra i più significativi vi sono:

- *aumento della produttività*: MDD riduce i costi di sviluppo del software per la generazione di codice, e ciò aumenta la produttività degli sviluppatori.
- *aumento del riuso*: MDD è particolarmente potente se applicato ad un livello del programma o dell' organizzazione. Questo perché il ritorno sugli investimenti, nello sviluppare le trasformazioni, aumenta ogni volta che vengono riutilizzate. L'uso di trasformazioni collaudate aumenta anche la prevedibilità di sviluppare nuove funzioni, e riduce i rischi del progetto.
- *migliore manutenibilità*: MDD porta alla realizzazione di un' architettura maggiormente manutenibile, consentendo, anche, di migrare componenti più efficienti su nuove tecnologie.
- *maggiore flessibilità*: i modelli di alto livello sono caratterizzati da un numero irrilevante di dettagli di implementazione. Mantenere i modelli privi di dettagli di

implementazione rende più facile la gestione dei cambiamenti per quanto riguarda la piattaforma sottostante e l' architettura.

- *maggiore consistenza*: effettuare manualmente la codifica ed attuare le decisioni architetturali sono attività soggette ad errori, MDD assicura che gli artefatti siano generati in modo coerente.
- *migliore comunicazione*: nei modelli sono omessi i dettagli di implementazione, in quanto essi non sono rilevanti per comprendere il comportamento logico del sistema. I modelli sono, quindi, molto più vicini al dominio del problema, riducendo il gap semantico tra i concetti che sono capiti dalle parti interessate e la lingua in cui si esprime la soluzione. I modelli facilitano anche la comprensione e il ragionamento sui sistemi al livello della progettazione. Questo porta ad una migliore comprensione su un sistema.
- *conservare la competenza*: i progetti dipendono spesso da esperti che ripetutamente prendono le decisioni migliori. Essendo la loro competenza catturata nei modelli e nelle trasformazioni, non si ha più il bisogno che essi siano fisicamente presenti.
- *minor numero di rischi*: quando si utilizza un approccio MDD, lo sviluppo delle applicazioni è focalizzato sull' attività di modellizzazione. Ciò significa che è possibile ritardare alcune scelte, come ad esempio la scelta di una specifica piattaforma, riducendo così eventuali rischi.

MDD è un' ottima metodologia di sviluppo, tuttavia, le risposte del settore in cui è utilizzato indicano che non tutti i progetti che utilizzano MDD hanno avuto successo. Questo è generalmente dovuto a diversi fattori:

- scarsa selezione di progetto,
- pianificazione insufficiente,
- mancanza di necessarie esperienze e competenze.

L'approccio basato sulla generazione di codice utilizzato da MDD ha, dunque, il suo lato negativo, a causa, ad esempio, dei vincoli nel codice generato, di sviluppatori non sufficientemente qualificati, e un accoppiamento troppo stretto al modello. Quando le

aziende si impegnano al 100 % per codificarela generazione, c'è poco spazio per la messa a punto, soprattutto quando gli sviluppatori devono sempre passare attraverso il modello.

Infine, per ottenere i maggiori benefici dal MDD bisogna:

- *Selezionare i progetti candidati giusti*. MDD è particolarmente potente se applicato ad determinato un livello di programma, perché il ritorno sull'investimento (ROI) dello sviluppo o acquisizione delle trasformazioni aumenta ogni volta che essi vengono riutilizzati. Un buon progetto candidato deve, inoltre, avere un'architettura ben definita e modelli di comportamento ripetibili, cosicche tali modelli possono essere astratti e possono essere usati nelle trasformazioni.
- *Considerare il costo aggiuntivo*: potrebbe essere necessario prendere in considerazione i costi di sviluppo o l'acquisto di trasformazioni, ma un'attenta pianificazione dei costi farà in modo che vi sia una riduzione complessiva dei costi nel lungo periodo.
- *Piccole vittorie*: in molte organizzazioni, un nuovo approccio viene accolto con un certo scetticismo finché non viene dimostrato che effettivamente funziona. Se si è davanti al primo progetto che utilizza l'approccio MDD, è opportuno suddividere lo sviluppo in un determinato numero di iterazioni. Le iterazioni successive si basano sulle esperienze, ciò consente di supportare una gamma sempre più ampia di scenari per la realizzazione di applicazioni aziendali.
- *Sviluppare le competenze giuste*. MDD richiede che gli architetti e gli sviluppatori abbiano familiarità con l'UML e l'IDE di MDD. Costruire queste competenze all'interno del team può richiedere un certo tempo all'inizio, che porta ad un relativo ritardo sul tempo totale impiegato per lo sviluppo dell'intero progetto. Tuttavia, con lo stesso team, i progetti successivi saranno molto più agevoli.

4.4.2 SkyGen: Application Framework & Code Generation

Da quanto descritto precedentemente, al giorno d'oggi il mercato dell' *Information Technology* impone la realizzazione, in tempi estremamente brevi, di soluzioni efficaci, complesse e funzionali, il tutto opportunamente documentato.

L'azienda SkyIT proprone una suite di prodotti, Sky*, che nasce proprio con l'intento di soddisfare queste esigenze. Tale suite è composta da:

- *SkyGen: Application Framework & Code Generation.* E' il componente più importante dell'intera suite di sviluppo. Esso è basato sul principio del Model Driven Development (MDD) e consente, pertanto, di trasformare modelli UML in artefatti;
- *SkySec: Enterprise Security Sub-System.* Integrato, oppure centralizzato, pone in sicurezza le applicazioni J2EE. È realizzato sulla potente infrastruttura di Spring Security ed offre i due principali componenti di Authentication & Authorization, per una gestione completa della sicurezza;
- *SkyDoc: Document life cycle management.* E' un'applicazione per la stesura della documentazione del progetto, come ad esempio: Specifiche dei requisiti, Specifiche funzionali, Casi di test. È integrata con SkyGen e permette, dunque la generazione automatica di documentazione e artefatti;
- *SkyJob: Batch processing and ETL*. Permette la generazione automatica di procedure batch dai modelli UML di riferimento, grazie a SkyGen. È un supporto semplificato e strutturato per la personalizzazione di tali procedure, grazie ad una versione arricchita di SpringBatch.

Sky* è una suite per lo sviluppo di soluzioni enterprise, basata sul principio del Model Driven Development. Inoltre, grazie a questa soluzione l'azienda riesce a registrare produttività elevatissime nello sviluppo di soluzioni software su tecnologia J2EE ad alto contenuto tecnologico e funzionale.

Soffermiamoci, nello specifico, sul componente SkyGen. Esso, come detto già in precedenza è una soluzione integrata per la generazione di applicazioni JEE, ed è basato

sul principio del Model Driven Development (MDD). In particolare, SkyIT adotta il terzo livello di astrazione del Model Driven Development.

SkyGen è, un framework per la generazione di applicazioni basate su di un modello strutturato. L'idea che sta alla base di questo strumento è quella di programmare partendo dalla modellazione. La funzionalità principale di SkyGen, infatti, è quella di accettare in ingresso un modello, ovvero un digramma UML e di fornire in uscita una serie di classi completamente implementate, la realizzazioni dei vincoli, le relazioni e le associazioni, che sono descritte nel modello di partenza. Il programmatore dovrà, quindi, creare il modello e dopo aver generato, tramite appunto SkyGen, customizzare tutti quegli aspetti che il modello non è in grado di rappresentare.

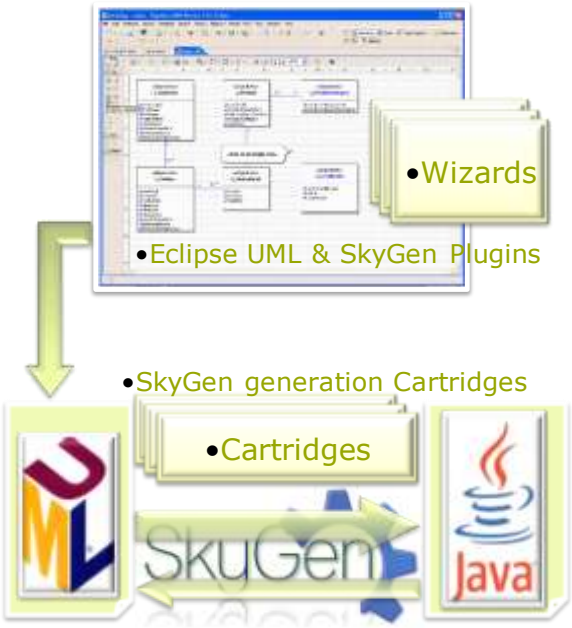


Figura 25. Schematizzazione funzionamento SkyGen

Le caratteristiche principali di SkyGen sono:

1. *Lazy-Instantiation.* È la tattica di istanziare un oggetto, di inizializzare una variabile, di creare una risorsa, di effettuare un calcolo o di eseguire un processo solo nel momento in cui tale operazione effettivamente richiesta, in questo modo si libera spazio prezioso in memoria. La Lazy-Insantiation è, dunque, la tecnica che dà

possibilità di navigare il modello delle entità in modo trasparente e "lazy", a qualsiasi livello e fuori da ogni contesto di persistenza e anche a distanza.

2. *Complesso Meccanimo di Persistenza.* Come il meccanismo di persistenza a cascata di Hibernate, tale tecnica dà la possibilità di rendere persistente la struttura di un'entità complessa, anche se non si conosce la relativa sessione di Hibernate, tutto ciò fornisce una spinta alle performance in termini di accesso al database.
3. *Servizio astratto di hot-plug per il livello di accesso (POJO, EJB, WebServices, AJAX).* Grazie a Spring si può facilmente switchare tra l'implementazione di POJO e il livello di accesso EJB 2.1. La stessa implementazione del servizio si può avere attraverso una chiamata al servizio web e attraverso una chiamata AJAX. Tutto ciò può essere semplicemente riassunto in "Scrivere una volta, usare ovunque!"
4. *MVC Pull vs MVC Push.* Come già detto il componente View del pattern MVC non fa altro che delegare al Controller i processi conseguenti l'input dell'utente. Ha, inoltre, la responsabilità della presentazione dei dati (GUI), in modo che i dati siano sempre aggiornati. Ciò può essere eseguito dalla View seguendo due approcci: push e pull. Con la tecnica push è il model che notifica al view sia il cambiamento di stato dei dati che le informazioni aggiornate; con la tecnica pull, invece, il view riceve dal model la notifica del cambiamento di stato e poi accede al model per ottenere le informazioni aggiornate. Esportando il modello direttamente nella presentazione (attraverso il modulo di azione), e grazie alla lazy instantiation, è possibile estendere il form della pagina alle entità padre e figlio semplicemente aggiungendo i controlli di modulo. I vantaggi di questa funzionalità non sono solo per la pagina di sola lettura, ma anche per submission/processing.
5. *Motore di ricerca avanzato.* Per estendere ai padri delle entità la funzione di ricerca con criteri, basta aggiungere i controlli del form alla pagina e ADFGen creerà automaticamente i joins per soddisfare la richiesta.
6. *Sotto sistema di sicurezza basato su ACEGI (Method authorization access level and ACL).* È realizzato un sotto-sistema di sicurezza utilizzando un ADFGen e

potenziando l'ACEGI security framework. ADFGen aggiunge la capacità di applicare facilmente le regole di autorizzazione per ogni transazione utente e definire facilmente i ruoli ACL, utilizzando fornite funzioni di sicurezza on-line.

7. *Costruzione di Menu.* È facile costruire una o più menu ad albero, su cui è possibile applicare il filtro per disattivare o eliminare gli elementi non autorizzati, in base all'utente attualmente connesso. ADFGen fornisce due layout del menu di default: orizzontale e verticale. È possibile utilizzarne uno, entrambi o anche nessuno. È possibile, infatti, configurare menù diverso per ogni diverso punto di vista, in quanto si può collegare ogni diversa autorizzazione con una diversa autenticazione. La configurazione predefinita del menu è memorizzata in un file xml di Spring.
8. *Scegliere i Tag.* Un componente riutilizzabile per consentire all'utente di selezionare righe di riferimento. Come un potente controllo dell' elenco a discesa che permette di filtrare, ordinare ed effetture l'impaginazione dei risultati. Si ha uno scambio automatico tra la versione popup e le altre versioni della pagina, a seconda della funzione che consente lo JavaScript del browser.
9. *Strict XHTML 1.0, CSS 2.* Le pagine generate rispettano pienamente tali standard. Infatti, le pagine che sono scritte contengono solo informazioni strutturali, mentre la disposizione è contenuta solo nei CSS. Provando a disattivare i CSS si può vedere come il risultato sia ancora una pagina che può essere tranquillamente utilizzata e visualizzata.
10. *Lavorare senza JavaScript.* Provando a disabilitare gli javascript e utilizzando l'applicazione si nota che:
    - il tag di selezione mostrerà una nuova pagina, ma tornando indietro i dati si perderanno;
    - la validazione dei dati in ingresso verrà eseguita sul lato server;
    - alcuni pulsanti di submit continueranno a lavorare grazie ad un filtro della servlet;
    - altri JavaScript presenti saranno rimossi automaticamente dalla pagina;

- la struttura dei menu potrà essere navigata utilizzando un browser con una dinamica struttura del menu;

11. *No-Table layout.* I tags <table> sono utilizzati solo per rendere vera un'informazione tabellare e non per effettuare dei controlli della pagina.
12. *Accessibilità.* Gli standard utilizzati sono:
    - W3C ;
    - Livello WCAG AA (il livello AAA non si ottiene solo per evitare il pre-riempimento di tutti i controlli dei moduli di ingresso);
    - US 508;
    - Legge Stanca.
13. *Integrazione di Sistema.* Grazie ai file di configurazione XML di Spring si può facilmente integrare più applicazioni insieme e vedere la demo in tempo reale.
14. *Event driven programming.* Indotta dallo spring container. La programmazione a eventi è un paradigma di programmazione. Mentre in un programma tradizionale l'esecuzione delle istruzioni segue percorsi fissi, che si ramificano soltanto in punti ben determinati predefiniti dal programmatore, nei programmi scritti utilizzando la tecnica a eventi il flusso del programma è largamente determinato dal verificarsi di eventi esterni. Event driven programming è alla base della programmazione delle GUI, l'utente, infatti, genera tramite la GUI una serie di eventi a cui il controllore deve prontamente reagire in maniera opportuna, è per questo che si parla in tal contesto di "Handling events" ("processare gli eventi"). Il Controller che, invece, processa gli eventi è chiamato event handler o event listener, mentre le informazioni sull'evento in Java sono memorizzate in opportuni oggetti detti EventObject. La computazione è guidata completamente dalla serie di eventi generati dall'utente, naturalmente gli eventi devono essere generabili in maniera coerente da parte dell'utente. Il programma processa tali eventi come input e aggiorna il proprio modello interno e lo visualizza tramite la View. Il Controller, invece, ha il compito di gestire il flusso di eventi e dati dalla vista al modello e

quindi nuovamente verso la vista, è anche possibile che più controllori, viste e modelli possono coesistere per formare un unico programma.

15. *AOP.* La programmazione orientata agli aspetti è un paradigma di programmazione basato sulla creazione di entità software - denominate *aspetti* - che sovrintendono alle interazioni fra oggetti finalizzate ad eseguire un compito comune. Il vantaggio rispetto alla tradizionale Programmazione orientata agli oggetti consiste nel non dover implementare separatamente in ciascuna classe il codice necessario ad eseguire questo compito comune. Supponendo di avere un'applicazione che deve effettuare il logging di alcune transazioni che richiedano l'interazione tra più oggetti: ognuno degli oggetti coinvolti deve, nei propri metodi, contenere codice in grado di gestire il suddetto problema; si viene così a creare, nella stesura del codice, una ridondanza non necessaria. Inoltre gli oggetti modificati a tale scopo diventano più complessi e quindi diminuisce la manutenibilità del programma. L'aspect oriented programming (AOP) è nato con lo scopo di risolvere problemi di questo tipo. Gli aspetti modellano cioè le problematiche trasversali agli oggetti stessi, ossia compiti (quali ad esempio l'accounting) che nell'OOP tradizionale sono difficilmente modellabili.

16. *Inversion of Control e gestione avanzata di complesse strutture di bean.* Indotta dallo Spring container. Grazie a Spring si può costruire una struttura complessa dei bean, e poi iniettare tale struttura in qualsiasi target, è possibile iniettare tutto ciò che si vuole, non solo le risorse JEE.
17. *Pattern di presentazione Master/Detail.* Ultimo ma non meno importante, un nuovo e potente Pattern Master/Detail per la gestione delle entità, esso è guidato dalla modellazione. Una presentazione Master/Detail consente all'utente di effettuare una selezione da una collezione di oggetti (master) e visualizzare i dettagli delle singole voci all'interno della raccolta (detail) all'interno della stessa schermata. Presentare entrambe le voci (principale e dettaglio) sulla stessa schermata porta due principali vantaggi: fornisce il contesto del set di dati e allo

stesso tempo permette di focalizzarsi su un solo elemento, e consente agli utenti di navigare velocemente e in modo efficiente nei diversi elementi del set di dati. La schermata principale di un modello Master/Detail può sembrare simile alla navigazione verticale e a quella di ricerca per filtri, ma c'è una distinzione importante. Nel modello Master/Detail, gli oggetti del Master sono raggruppamenti in base alla loro tipologia, viceversa, nel caso di filtri di ricerca e di navigazione verticale, queste categorie possono contenere più insiemi eterogenei, e, in particolare nel caso di ricerca con filtri, l'elenco degli elementi può essere non-deterministico. Il rapporto Master/Detail è una relazione uno-a-molti (una raccolta master , molti articoli della collezione). Il design dei componenti principali e di dettaglio può assumere molte forme come ad esempio lista e albero.

Bisogna utilizzare il modello Master/Detail:

- Quando l'utente desidera visualizzare o agire su un singolo elemento all'interno di un set di dati, mantenendo il più ampio possibile il contesto del set di dati.
- Quando un utente naviga un insieme di dati, e deve ispezionare i dettagli degli elementi del set al fine di determinare quale scegliere. Il Master/Detail rende questo scenario più veloce, perché l'utente non ha bisogno di passare a delle pagine separate per trovare le informazioni necessarie per effettuare la sua selezione.

Non bisogna, invece, utilizzare il Master/Detail:

- Quando la quantità di contenuto supera la larghezza della pagina sia nell'area di master sia in quella di dettaglio. Si tratta di una buona pratica per evitare lo scorrimento orizzontale.
- Quando il flusso di lavoro nell'area dettagli necessita di una estesa attività intorno ad un unico oggetto. La natura della zona dettaglio è transitoria e ciò potrebbe portare a problemi di capacità e di flessibilità pur di soddisfare le interazioni necessarie.

La prima cosa da dover fare per utilizzare il generatore di codice, SkyGen, è la realizzazione di un modello. Quando si deve realizzare un progetto, nel caso più specifico, un'applicazione web, bisogna anzitutto analizzare il problema ed individuare quali siano le entità (o classi)di cui si avrà bisogno; tali entità saranno poi trasformate nelle tabelle che andranno a popolare il database dell'applicazione. Dunque, si dovrà disegnare il class diagram che descriverà l'applicazione da realizzare. Per fare ciò, verrà, semplicemente, utilizzato un tool di Eclipse. Tale tool permetterà di disegnare il diagramma UML, in quanto I costrutti UML da esso supportati sono molti, e dovrebbero essere sufficienti per la maggior parte dei modelli: classi, classi astratte, attributi, operazioni, riferimenti (one-way, bidirezionali, contenimento, con tutte le molteplicità possibili), ereditarietà, enumerazioni e data types. Per ogni entità disegnata saranno quindi indicati i suoi attributi, le sue relazioni ed associazioni, come mostrato in figura:

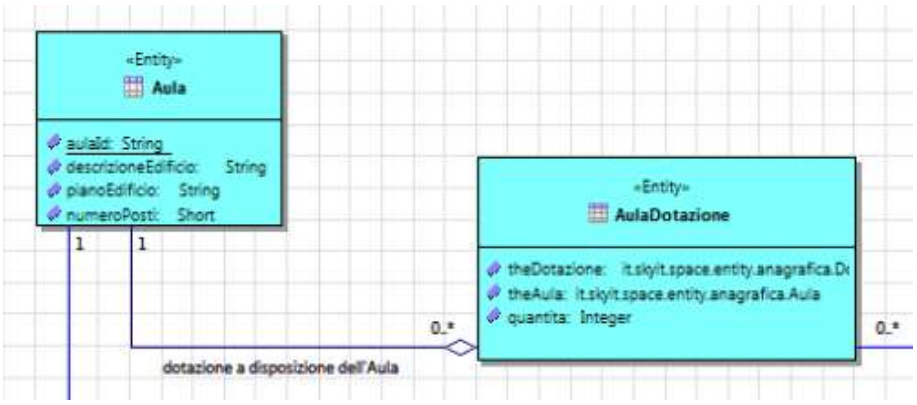


Figura 26. Diagramma UML

Una volta completata la fase di modellazione, tale diagramma delle classi sarà dato come input a SkyGen. L'output del generatore saranno quattro progetti:

- Core: conterrà i package e quindi i file .java relativi alle entity, ai dao, alle factory, ai service, alle utility. In particolare, i file .java delle entity conteranno tutte le dichiarazioni degli attributi delle entità e le implemetazioni dei metodi set e get di tali attributi.
- Ear: utile quando dovrà essere generato un file jar dell'applicazione

- Impl: conterrà i package e quindi i file .java relativi alle implementazioni dei dao, dei service e della security.
- Root: consiste invece nella radice dell'intero progetto e da un punto di vista della customizzazione o delle modifiche difficilmente sarà modificato. In esso sono presenti tutti i file di configurazioni relativi all'applicazione e al database.
- Web: conterrà i package e quindi i file relativi ai controller, ai decorator,a i form, e alla security; inoltre è qui contenuto tutto ciò che ha a che vedere con la visualizzazione web dell'applicazione, dunque quindi vi sono tutte le jsp relative ai button, ai common, ai layout, alle pages e ai popups, e tutti i file xml dei tiles.

L'applicazione avrà questa struttura:

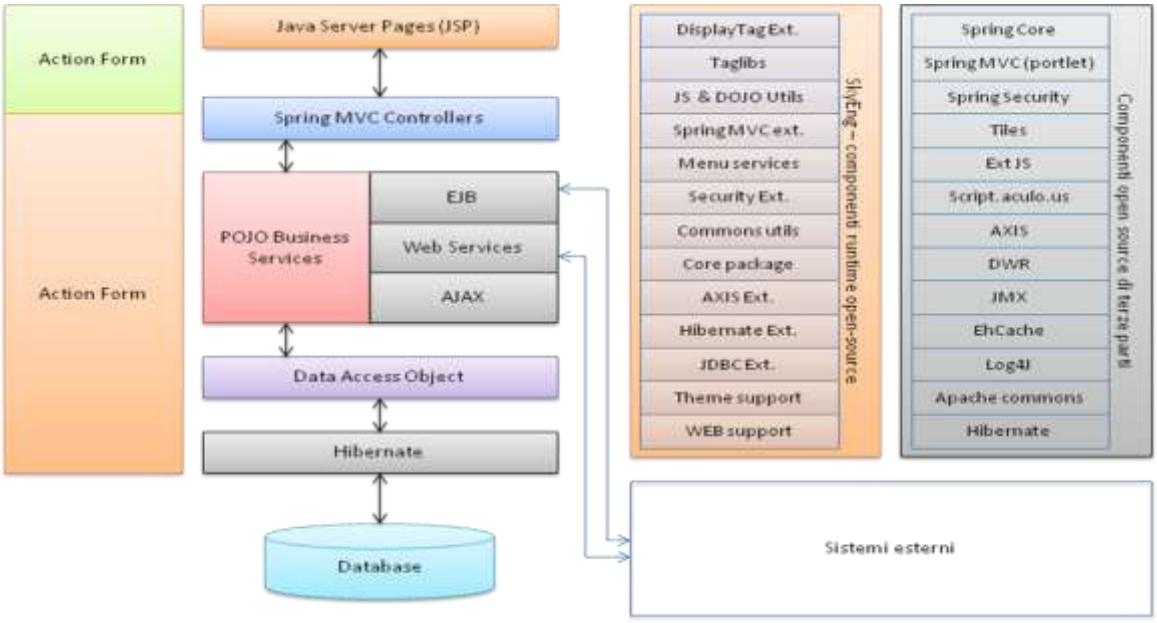


Figura 27. Struttura di un'applicazione web generata con SkyGen

Tutti i file generati sono completi, vengono infatti anche implementate le operazioni *"CRUD"* per ogni entità descritta nel modello. Dunque, non si può parlare solo di "scheletri", pronti per l'implementazione da parte del programmatore, ma di un'applicazione che è già possibile lanciare. Naturalmente il comportamento dell'applicazione sarà, in questo momento, del tutto standardizzato, per darle, infatti, un

comportamento sempre più compatibile con le aspettative e le richieste del cliente, il codice generato dovrà essere customizzato. È proprio questo il compito del programmatore.

SkyGen, inoltre, genera anche tre file di fondamentale importanza:

- *Oracle-DDL.ddl*: è il file contenente la creazione delle tabelle del database dell'applicazione, naturalmente scritto in linguaggio sql. Tutte le entità del modello, disegnato inizialmente, sono qui trasformate effettivamente in tabelle
- *SkyDoc_SQLData.sql* e *SQL_Data. sql*: sono i file contenente la creazione e implementazione, lato database, degli utenti, dei profili, della profilazione degli utenti, delle operazoni elementari e le relative profilazioni.

Questi file saranno direttamente eseguiti come script, all'interno della database creato appositamente per l'applicazione che si sta realizzando. In questo modo, il database è stato popolato dalle tabelle relative alle entità e alla sicurezza del sistema. Il programma utilizzato per la creazione, del databese è Toad for Oracle.

I professionisti che si occupano di database Oracle devono svolgere numerose attività, come l'esecuzione di query, la creazione di report, la scrittura e il debug di codice, l'amministrazione e l'ottimizzazione. Per gestire questo carico di lavoro, occorre una soluzione in grado di automatizzare le attività quotidiane e di fornire un flusso di lavoro flessibile per passare rapidamente da un'attività all'altra. Toad for Oracle rappresenta una nuova frontiera in termini di produttività per DBA, sviluppatori e analisti. Grazie al suo livello superiore di facilità d'uso, automazione, documentazione e flessibilità, consente sviluppo e amministrazione efficienti e accurati dei database Oracle. Toad for Oracle, negli anni, si è evoluto fino a diventare la più avanzata soluzione del proprio genere. Rappresenta lo strumento ideale per gli utenti Oracle con qualsiasi livello di esperienza, assicurando:

- maggiore efficienza nell'esecuzione delle attività quotidiane.
- automazione per ottenere più risultati in meno tempo e con meno fatica.
- risorse di formazione integrate, incluso Toad World.

- eccezionale facilità d'uso per utilizzare meglio le best practice e ridurre al minimo gli errori di codifica.
- maggiore precisione durante la scrittura, il debug e l'ottimizzazione del codice.
- flessibilità per eseguire attività di sviluppo e amministrazione da un singolo strumento.
- un flusso di lavoro semplificato per passare facilmente da un'attività all'altra.
- avanzate capacità di reporting per produrre dati quantificabili e documentazione

Una volta generate le implementazioni delle classi, ci si potrebbe limitare ad utilizzare ciò che SkyGen ha preparato. Nella maggior parte dei casi, tuttavia, il modello sarà abbastanza complicato da richiedere un po' di customizzazione e di modifiche.

Aggiungere al codice generato un metodo ex-novo non comporta alcun problema, in tal caso, infatti, non ci sarà bisogno di rigenerare. Invece, se venissero apportate modifiche alle entità, come l'aggiunta ad una di esse di un attributo, o la modifica del tipo di uno o più attributi, bisognerebbe rigenerare l'intero modello, in quanto non è supportata la rigenerazione parziale. In questa particolare situazione, si potrebbe pensare di perdere le custumizzazioni e le modifiche fatte nel precedente modello, in realtà non è così. Ogni volta che un modello è generato, il risultato della generazione sarà salvato in una repository, di conseguenza, ogni volta che il codice è rigenerato la nuova generazione, è sincronizzata con la precedente. Sarà data la possibilità al programmatore, quindi, di visualizzare quali file sono stati modificati dalla nuova generazione e quali sono i conflitti che si sono creati con la vecchia generazione. In tal modo sarà il programmatore a decidere quale delle due versioni dovrà essere mantenuta. In particolare, il programmatore visualizzerà ogni singola riga di codice che genera conflitto tra le due versioni e potrà scegliere quale mantenere nella repository. Sono in questo modo conservate customizzazioni e modifiche, anche dopo un certo numero di rigenerazioni.

Da ciò, si comprende facilmente, che la stesura di un modello è forse la parte più importante per la realizzazione dell'applicazione, in quanto se le idee sono chiare fin

dall'inizio non ci sarà più bisogno di rigenerare, ma dovranno essere apportate solo modifiche e customizzazioni.

SkyGen è lo strumento che permette alla società SkyIT di generarefacilemente codice, che è leggibile e di qualità eccezionale. Inoltre, permette di realizzare, in tempi estremamente brevi, di soluzioni efficaci, complesse e funzionali, il tutto opportunamente documentato. Tuttavia la potenza di sviluppo del codice offerta dalla suite Sky* non è tutto in un processo evoluto di produzione di sistemi software su larga scala. Ecco perché si sono adoperati per l'adozione di un processo di sviluppo finalizzato all'automazione dei processi di build e controllo (particolarmente utili per progetti di sviluppo distribuiti) quali:

- Agile Project Planning & Tracking con metodologia SCRUM;
- Issue tracking & control;
- Continuous Integration process (nightly build, source xref, javadoc, artifact deployment, site generation).
- Collaboration & Wiki systems;
- Metric collection, analisys & reporting;
- Automatic build notifiers.

SkyGen, EMF e Spring Roo, come ben si può capire da quanto detto, sono strumenti di fondamentale importanza, in quanto permettono di alleggerire il compito del programmatore e di ridurre notevolmente i tempi di sviluppo, ma, naturalmente ognuno ha i propri difetti e i propri pregi.

Spring Roo è uno strumento molto importante per tutti coloro che utilizzano il framework Spring, in quanto permette l'implementazione dei propri progetti Java EE, e di utilizzare il concetto di Convention over Configuration. Attraverso Roo, infatti, si può scrivere un server Java EE basato su Spring seguendo lo "standard di Spring" in maniera rigida. Il vantaggio di tutto ciò è avere un progetto maggiormente manutenibile e comprensibile da qualsiasi sviluppatore Spring. Sostanzialmente però, Spring Roo non è semplice da utilizzare come EMF o SkyGen, in quanto a differenza di questi ultimi, il modello da cui

generare non è disegnato, ma deve essere introdotto da shell, e quindi scritto a linea di comando. EMF e SkyGen, hanno, invece, il vantaggio di poter utilizzare tools per la realizzazione di diagrammi UML che costituiranno l'input del generatore stesso. EMF, inoltre, permette l'importazione di un modello come input. Allo stato attuale sono quattro i formati che si possono utilizzare: XMI, XMLSchema, UML e Java annotated code.

SkyGen, invece, prevede come suo input solo un diagramma delle classi. Un ulteriore componente che viene generato da EMF, e che lo differenzia dagli altri,è un insieme di metodi di test. Anche in questo caso, non essendo il comportamento modellizzato, si tratta di scheletri, che tuttavia facilitano il compito per il programmatore. Spring Roo, invece, nella sua installazione predefinita facilita la creazione di applicazioni anche grazie all'utilizzo della tecnologia JUNIT, che genera automaticamente test per i progetti realizzati dall'utente. Per quanto concerne invece SkyGen, non vi è alcun suo componente che possa generare, oltre al codice sorgente, anche i test. Essi quindi dovranno essere effettuati sfruttando l'ambiente di sviluppo Spring ToolsSource in cui si lavora, e quindi la libreria Junit, integrata con Spring.

4.5 Vantaggi e Svantaggi dei generatori di codice

Il riutilizzo basato su generatori ha avuto particolare successo per i sistemi applicativi aziendali, e ci sono molti generatori che possono produrre applicazioni complete o automatizzare parzialmente la creazione dell'applicazione e lasciare al programmatore i dettagli specifici. Quest'approccio è utilizzato anche in altri ambiti, tra cui:

1. Generatori di parser per l'elaborazione del linguaggio. L'ingresso del generatore è una grammatica che descrive il linguaggio da analizzare, e l'uscita è un convertitore del linguaggio. Quest'approccio si concretizza in sistemi come lex e yacc per C e JavaCC, un compilatore per Java.
2. Generatori di codice in strumenti CASE. L'ingresso di questi generatori è un progetto software e l'uscita è un programma che implementa il sistema progettato. Questi possono essere basati su modelli UML e, a seconda delle informazioni nei

modelli UML, generare sia un programma completo o un componente, o un skeleton di codice. Lo sviluppatore software aggiunge dettaglio per completare il codice.

Il riutilizzo basato su generatori approfitta della struttura comune di applicazioni in questi settori. La tecnica è stata utilizzata anche in domini applicativi più specifici come i sistemi di comando e controllo (O'Connor, et al., 1994) e la strumentazione scientifica (Butler, 1994) dove sono state sviluppate librerie di componenti. Gli esperti di settore quindi utilizzano il linguaggio specifico del dominio per unire questi componenti e creare nuove applicazioni. Tuttavia, vi è un costo iniziale più elevato per la definizione e implementazione dei concetti del dominio e del linguaggio di composizione. Questo ha fatto sì che molte aziende fossero riluttanti ad assumere i rischi di questo approccio.

Il riutilizzo basato su generatori è efficace in termini di costi per applicazioni come l'elaborazione dei dati aziendali. È molto più facile per gli utenti finali di sviluppare programmi utilizzando generatori rispetto ad altre soluzioni al riutilizzo basate su componenti. Inevitabilmente, tuttavia, ci sono inefficienze nei programmi generati. Ciò significa che potrebbe non essere possibile utilizzare questo approccio in sistemi con alti requisiti di prestazioni o di rendimento.

La programmazione generativa è una componente chiave di tecniche emergenti di sviluppo software che combinano la generazione di programmi con sviluppo basato su componenti.

Il più sviluppato di questi approcci è lo sviluppo di software orientato agli aspetti (AOSD) (Elrad, et aI., 2001). Lo sviluppo di software orientato agli aspetti affronta uno dei principali problemi nella progettazione di software, il problema della separazione dei concern, principio basilare nella progettazione; si deve progettare il software in modo che ogni unità o ogni componente faccia una cosa sola.

In molte situazioni, tuttavia, i concern non sono associati a funzioni applicative chiaramente definite, ma sono trasversali, tanto da influenzare tutti i componenti del sistema. Per esempio, se si vuole seguire l’utilizzo di ogni modulo del sistema per ciascun

utente, si deve avere un concern di monitoraggio associato a tutti i componenti, che non può essere implementato semplicemente come un oggetto di riferimento di tali componenti. Il monitoraggio specifico che è eseguito, richiede informazioni ambientali alle funzioni del sistema monitorato. Nella programmazione orientata agli aspetti, tali concern trasversali sono implementati come aspetti e, all'interno del programma, si definisce, dove un aspetto deve essere inserito. Questi sono chiamati i punti d'unione. Gli aspetti sono sviluppati separatamente, poi, in una fase di precompilazione chiamata tessitura degli aspetti, sono collegati ai punti di unione. La tessitura degli aspetti e una forma di generazione di programma: l'uscita dal tessitore è un programma in cui è stato integrato il codice dell'aspetto.

Di seguito saranno presentati alcuni esempi di generati di codice integrati con i principali strumenti IDE.

Capitolo 5: SPACE-CLASSROOM

L'applicazione web ***Space Classroom***, essendo un'applicazione gestionale, ha come scopo la gestione degli spazi e delle aule del DIETI (Dipartimento di ingegneria elettrica e delle tecnologie dell'informazione dell'Università Federico II di Napoli). L'obiettivo del progetto è quello di rappresentare il dipartimento, attraverso l'utilizzo delle carte topografiche degli edifici, con le relative informazioni riguardanti le singole stanze e i loro utilizzatori.

L'applicazione è rivolta a tutti coloro che vogliono avere informazioni sui locali del dipartimento. Nello specifco, le informazioni riguardanti i locali possono essere visualizzate da tutti gli utenti, mentre, operazioni come la prenotazione dell'aula possono essere effettuate solo da parte di utenti autorizzati.

L'applicazione è composta dalle seguenti Macrofunzioni:

- Gestione Prenotazioni
- Gestione Aule
- Gestione Dotazioni
- Gestione Personale
- Gestione Tipo Prenotazione

5.1 Contesto dell'intervento del progetto

Nel paragrafo si descriverà il contesto nel quale l'utente opera, inteso come insieme di procedure amministrative e di attività, di vincoli organizzativi o di altro tipo, caratteristici della realtà in cui si deve intervenire.

Gli aspetti considerati saranno legati, ad esempio, a particolari schemi di organizzazione del lavoro, a flussi di comunicazione da garantire o ad un eventuale distribuzione delle postazioni di lavoro. L'individuazione di questi elementi consentirà di definire, nella loro complessità, le funzioni, i dati coinvolti ed i requisiti funzionali, nonché di stabilire vincoli e requisiti non funzionali del sistema che si va a realizzare.

5.1.1 Contesto attuale

Volendo analizzare il contesto dipartimentale attuale, si può facilmente notare come visitando il sito dell'università sia possibile accedere alle carte topografiche degli edifici, in modo da individuare la locazione delle singole aule. Tuttavia, non è possibile avere informazioni più specifiche sull'aula, quali ad esempio la capienza, l'arredo, il numero di pc presenti, i software installati sui pc, si ha inoltre difficoltà nell'individuazione di quali siano le aule libere in un determinato arco temporale. È facile dedurre, quindi, che ciò che manca è proprio un sistema di gestione e prenotazione delle aule automatizzato, attualmente, infatti, è possibile la prenotazione da parte degli utenti autorizzati solo in via cartacea.

5.1.2 Contesto previsto ed ipotesi di soluzione

Dalla descrizione condotta in precedenza sono emerse, dunque, nuove esigenze nell'ambito dipartimentale, che richiedono l'introduzione di un sistema di gestione informatizzato che svolga le seguenti Macrofunzioni:

- **Gestione Prenotazioni.** Tale funzione permetterà, ad un utente autorizzato, di poter prenotare l'aula in base alle proprie esigenze, in quanto, al momento della prenotazione potranno essere visualizzate le informazioni relative all'aula che si vuole prenotare. La prenotazione potrà essere anche cancellata e/o modificata. Inoltre, saranno richieste, al momento della prenotazione, data e ora della prenotazione in modo tale da non avere eventuali conflitti con le prenotazioni già effettuate.

- **Gestione Aule.** Questa funzione permetterà a utenti autorizzati di inserire tutte le informazioni relative alle aule, ovvero: l'ubicazione (in termini di edificio e piano), capienza, arredo (numero di computer presenti, proiettori, cattedre, lavagne, sedute), e prenotazioni effettuate (in termini di data, orario e motivazione). Tali informazioni saranno utili, in particolar modo, agli utenti autorizzati che vorranno effettuare una prenotazione, in quanto riusciranno a soddisfare tutte le loro esigenze.
- **Gestione Dotazioni.** Questa funzione permetterà di arricchire le informazioni riguardanti un'aula, in quanto con essa sarà possibile visualizzare, inserire, modificare, eliminare i diversi tipi di arredo che compongono un' aula. Le dotazioni inserite potranno, così, essere associate ad ogni aula presente nella rispettiva anagrafica. Tutte queste operazioni elementari potranno essere svolte solo da un utente registrato come amministratore.
- **Gestione Personale.** Solo l'utente amministratore potrà usufruire di tale macrofunzione. Essa permetterà di inserire, nell'anagrafica del personale, nome, cognome e titolo di ogni professore del DIETI. Sarà, inoltre, possibile per l'amministratore visualizzare, modificare ed eliminare tali informazioni. Con la realizzazione dell'anagrafica relativa al personale si potrà inserire, al momento della prenotazione di un'aula, il nominativo del docente responsabile della prenotazione.
- **Gestione Tipo Prenotazione.** Tale macrofunzione permetterà di inserire, modificare, visualizzare, eliminare i possibili motivi delle prenotazioni effettuate. Le operazioni elementari che compongono tale macrofunzione possono essere eseguite da un amministratore o da un utente registrato come docente.
- **Gestione Sicurezza.** L'unico utente che può accedere alle operazioni elementari di questa macrofunzione è l'amministratore. Egli avrà, infatti, la possibilità di inserire, modificare, visualizzare, eliminare:
    - le applicazioni web associate a questo sottosistema di sicurezza;

- i profili di utenza, ai quali associare le operazioni elementari concesse;
- gli utenti, indicando username e password;
- le operazione di log;

Infine, sarà possibile pubblicare le modifiche.

Con tal sistema, sarà resa molto più semplice la comunicazione tra il dipartimento e gli studenti. Questi ultimi potranno, infatti, visualizzare, in modo molto semplice, quale sia l'aula prenotata per una lezione e/o per un esame e chi sia il docente responsabile della prenotazione. Per i docenti, invece, sarà facilmente concessa l'individuazione di un'aula libera e adatta per la prenotazione di un corso o di un esame.

5.2 Utenti dell' applicazione

L'applicazione SPACE CLASSROOM potrà essere acceduta da tutti gli utenti che desiderano avere informazioni circa i locali del dipartimento, anche se con granularità differente. È, infatti, possibile, distinguere, tre tipologie di utente:

- Ospite: è un semplice utente non registrato. Egli avrà accesso esclusivamente alle informazioni pubbliche che riguardano le aule. L'ospite, infatti, potrà utilizzate solo le operazioni elementari di ricerca e visualizzazione dei dettagli della macrofunzione Gestione Aule.
- Utente registrato: è uno studente o un professore. Avrà accesso alle informazioni pubbliche e a informazioni più specifiche riguardanti le aule, quindi non solo la loro ubicazione (edificio e piano), ma anche informazioni su disponibilità di prenotazione, capienza e arredo. Gli sarà, inoltre, possibile effettuare la prenotazione di un'aula, sia essa una semplice aula o un laboratorio o una sala riunioni, indicando la data, la fascia oraria in cui si vuole tener impegnato il locale e il motivo della prenotazione. Quindi, l'utente registrato potrà accedere a tutte le operazioni elementari delle macrofunzioni: Gestione Prenotazioni, Gestione Aule, Gestione Tipo Prenotazione.

- Amministratore: è utente registrato con particolari privilegi. Egli, oltre a poter visualizzare tutte le informazioni riguardanti gli spazi e ai locali del dipartimento, potrà effettivamente gestire tali informazioni, inserendole, modificandole o eliminandole. Infine, potrà registrare un nuovo utente e assegnare ad ognuno determinati privilegi. Quindi, l'amministrare gestirà tutte le macrofunzioni, ovvero Gestione Prenotazioni, Gestione Aule, Gestione Dotazione, Gestione Personale, Gestione Tipo Prenotazione e Gestione Sicurezza.

5.3 Requisiti Funzionali

L'applicazione Space ClassRoom è composta dalle due Macrofunzioni a cui si è accennato precedentemente, Gestione Aule e Gestione Prenotazione. Ognuna di esse è a sua volta costituita da sottofunzioni, esaminiamole nello specifico:

- La macrofunzione Gestione Aule è costituta dalle seguenti operazioni elementari:
    - Ricerca Aule: permette di ricercare le aule già presenti nel database. La ricerca può avvenire inserendo uno o più parametri di ricerca, come ad esempio il nome dell'aula, il nome dell'edificio in cui è ubicata l'aula oppure il numero dei posti disponibili nell'aula. Inoltre, è possibile effettuare la ricerca anche non inserendo alcun parametro, in questo caso si otterrà come risultato la lista di tutte le aule memorizzate.
    - Visualizza Aule: permette di visualizzare le informazioni relative all'aula selezionata. Nello specifico, le informazioni che saranno rese disponibili tramite questa funzione saranno l'ubicazione dell'aula, la capienza, e l'arredo.
    - Inserisci Aula: permette di inserire informazioni relative ad una determinata aula. Tale funzione potrà essere adoperata esclusivamente da un amministratore.
    - Salva Nuova Aula: permette di memorizzare un'aula nel database. Tale funzione potrà essere adoperata esclusivamente da un amministratore.

- Modifica Aula: permette di modificare le informazioni relative ad una determinata aula. Tale funzione potrà essere adoperata esclusivamente da un amministratore.
- Elimina Aula: permette di eliminare le informazioni relative ad una determinata aula. Tale funzione potrà essere adoperata esclusivamente da un amministratore.
- Stampa Aule: permette di stampare tutte le aule presenti nell'anagrafica.

- La macrofunzione Gestione Prenotazione, invece, è costituta dalle seguenti operazioni elementari:
  - Ricerca Prenotazione: permette di effettuare la ricerca di una prenotazione. La ricerca può avvenire attraverso l'inserimento di alcuni parametri, quali il nome dell'aula prenotata, il nome dell' edificio, il motivo della prenotazione, la data di inizio, l'ora di inizio, la data di fine e l'ora di fine, oppure senza l'inserimento di alcun parametro. In quest'ultimo caso ciò che si ottiene è la lista completa di tutte le prenotazioni effettuate, dove per ogni prenotazione è indicato il numero della prenotazione, il nome dell'aula prenotata, il motivo della prenotazione, il responsabile della prenotazione, la data di inizio, l'ora di inizio, la data di fine e l'ora di fine. Tale funzione potrà essere effettuata da un utente registrato.
  - Visualizza Prenotazione: permette di visualizzare le informazioni dettagliate relative alla prenotazione selezionata.
  - Inserisci Nuova Prenotazione: permette di effettuare la prenotazione di una determinata aula, indicando il nome dell'aula, il motivo della prenotazione, le date e le ore in cui si vuole occupare l'aula. Tale operazione potrà essere effettuata da un utente registrato.
  - Salva Nuova Prenotazione: quest' operazione permette di memorizzare la nuova prenotazione nel database.

- Modifica Prenotazione: permette di modificare tutti i campi relativi ad una prenotazione effettuata; è possibile, infatti, modificare data, orario, aula, motivazione, ed inoltre, è possibile aggiungere/modificare personale responsabile della prenotazione.
- Elimina Prenotazione: permette di eliminare la prenotazione di una determinata aula. Tale funzione potrà essere effettuata da un utente registrato.
- Stampa Prenotazioni: permette di stampare tutte le prenotazioni effettuate in un determinato giorno oppure in un determinato arco temporale (una settimana, un mese...).

- La macrofunzione Gestione Dotazione è costituta dalle seguenti operazioni elementari:
  - Ricerca Dotazione: permette di ricercare tra le dotazioni presenti nel database. La ricerca può avvenire inserendo il nome della dotazione che si sta ricercando oppure non inserendo alcun parametro, si otterrà la lista completa di tutte le dotazioni.
  - Visualizza Dotazione: permette di visualizzare le informazioni relative alla dotazione selezionata.
  - Inserisci Dotazione: permette di inserire una nuova dotazione. Tale funzione potrà essere adoperata da un amministratore.
  - Salva Nuova Dotazione: permette di salvare una nuova dotazione nel database.
  - Modifica Dotazione: permette di modificare le informazioni relative ad una determinata dotazione. Tale funzione potrà essere adoperata da un amministratore.
  - Elimina Dotazione: permette di eliminare la dotazione dal database. Tale funzione potrà essere adoperata da un amministratore.
  - Stampa Dotazioni: permette di stampare tutte le dotazioni memorizzate.

- La macrofunzione Gestione Personale è costituta dalle seguenti operazioni elementari:
    - Ricerca Personale: questa operazione permette di effettuare una ricerca tra il personale già inserito nel database. La ricerca potrà essere puntuale inserendo alcuni parametri di ricerca, come il nome, il cognome o il titolo del docente che si sta ricercando, oppure non inserendo alcun parametro il risultato della ricerca restituirà la lista di tutti i docenti memorizzati.
    - Visualizza Personale: permette di visualizzare le informazioni relative al docente selezionato. Nello specifico, le informazioni che verranno rese disponibili saranno il codice identificativo di ogni docente, il nome, il cognome e il titolo.
    - Inserisci Personale: permette di inserire un nuovo elemento nell'anagrafica relativa al personale. Tale funzione potrà essere adoperata da un amministratore.
    - Salva Nuovo Personale: permetterà di salvare il nuovo elemento inserito nel database.
    - Modifica Personale: permette di modificare le informazioni relative ad un docente selezionato. Tale funzione potrà essere adoperata da un amministratore.
    - Elimina Personale: permette di eliminare il personale selezionato. Tale funzione potrà essere adoperata da un amministratore.
    - Stampa Personale: questa operazione permetterà di stampare tutti gli elementi presenti nell'anagrafica del personale.

- La macrofunzione Gestione Tipo Prenotazione è costituta dalle seguenti operazioni elementari:
    - Ricerca Tipo Prenotazione: permette di ricercare tra i tipi di prenotazione presenti nel database. La ricerca può avvenire inserendo il motivo (o tipo)

della prenotazione che si sta ricercando, oppure non inserendo alcun parametro, si otterrà la lista completa di tutte i tipi di prenotazione.

- Visualizza Tipo Prenotazione: permette di visualizzare le informazioni relative al tipo di prenotazione selezionata.
- Inserisci Tipo Prenotazione: permette di inserire un nuovo tipo di prenotazione. Tale funzione potrà essere adoperata da un amministratore, ma anche da un utente registrato come docente.
- Salva Nuovo Tipo Prenotazione: permette di salvare un nuovo tipo di prenotazione nel database.
- Modifica Tipo Prenotazione: permette di modificare le informazioni relative ad un determinato tipo di prenotazione. . Tale funzione potrà essere adoperata da un amministratore, ma anche da un utente registrato come docente.
- Elimina Tipo Prenotazione: permette di eliminare le informazioni relative ad un determinato tipo di prenotazione. Tale funzione potrà essere adoperata da un amministratore, ma anche da un utente registrato come docente.
- Stampa Tipo Prenotazione: permette di stampare tutti i tipi di prenotazione memorizzati.

Si consideri, infine, la macrofunzione Gestione Sicurezza. Il sotto-sistema di sicurezza high-end offre una versatilità e completezza sia sotto il profilo dell'autenticazione che sotto il profilo di autorizzazione degli accessi. Consente di definire in maniera flessibile le regole di autenticazione e autorizzazione. La soluzione consente di modificare le regole di sicurezza online, utilizzando apposite funzioni web di profilazione:

- Autenticazione sicura su protocollo https con cifratura della password a una via, oppure integrazione con tutti gli authentication managers conosciuti;
- Possibilità di definire regole di accesso a livello di singola operazione elementare eseguibile a sistema;

- Possibilità di definire regole dinamiche di profilazione utenti in funzione delle regole di business definite;
- Possibilità di monitorare tutte le operazioni sensibili e scegliere la modalità di invio dei risultati: file di log, database, email, ecc.
- Definire in maniera flessibile gli attori del sistema.

Le possibili funzioni presenti in tal sottosistema sono:

- Gestione Applicazioni. Attraverso le funzionalità utente è possibile definire quali sono gli applicativi soggetti alle regole del sottosistema di sicurezza. Il modulo Sicurezza è organizzato per gestire la sicurezza anche di altre applicazioni esterne a SPACE. Attraverso le schermate di gestione l'utente può:
  - Ricercare e visualizzare i moduli installati e gestiti dal sottosistema della sicurezza;
  - Esportare tali liste sia in formato PDF sia in formato Excel;
  - Inserire un nuovo Modulo applicativo da sottoporre a Sicurezza;
  - Modificare un Modulo Applicativo esistente in archivio;
  - Eliminare un Modulo Applicativo esistente dall'archivio (saranno eliminati a cascata i profili, le operazioni elementari associate al modulo).

  Un modulo soggetto a sicurezza è identificato attraverso un Identificativo (di solito un acronimo che identifichi l'applicazione) e una descrizione.
- Gestione Operazioni. Le operazioni elementari rappresentano tutte le operazioni di un'applicazione che è possibile sottoporre a controllo d'autorizzazione. L'utente attraverso le funzionalità di Gestione "Operazioni" può manutenere tutte le operazioni elementari di tutti i moduli definiti nel sistema. Attraverso le schermate di gestione l'utente può:
  - Ricercare e visualizzare le operazioni elementari;
  - Esportare tali liste sia in formato PDF che in formato Excel;
  - Inserire una nuova Operazione Elementare;
  - Modificare un'Operazione esistente in archivio;

- Eliminare un'Operazione esistente dall'archivio.

Tutte le operazioni elementari non censite sono considerate come "sempre autorizzate".

- Gestione Profili. Censito il Modulo Applicativo il passo successivo è quello di censire i profili gestiti dal modulo. L'utente attraverso le funzionalità di Gestione dei "Profili" può:
  - Ricercare e visualizzare la lista di Profili censite per i moduli registrati;
  - Esportare tali liste sia in formato PDF sia in formato Excel;
  - Inserire un nuovo Profilo;
  - Modificare un Profilo esistente in archivio;
  - Eliminare un elemento Profilo esistente dall'archivio.

  Inoltre, dalla funzionalità di visualizzazione, dettaglio e modifica, è possibile visualizzare e modificare il contenuto delle Operazioni elementari associate al Profilo.
- Gestione Utenti. Attraverso tale funzionalità è possibile:
  - Ricercare e visualizzare liste degli Utenti;
  - Esportare tali liste sia in formato PDF che in formato Excel;
  - Inserire un nuovo elemento Utente;
  - Modificare un Utente esistente in archivio;
  - Eliminare un Utente esistente dall'archivio.

  E' possibile temporaneamente abilitare o disabilitare un utente all'accesso al sistema.
- Gestione Pubblica Modifiche. La funzionalità viene utilizzata per rendere persistenti le modifiche al sotto-sistema di sicurezza circa:
  - Inserimento di nuove funzionalità;
  - Modifica di funzionalità esistenti;
  - Modifica dei profili utente.

Gli utenti del sistema beneficiano di tali modifiche solo a seguito dell'attivazione di tale funzionalità.

5.4 Requisiti Non Funzionali

L'individuazione, in questa fase, di tutti i requisiti non funzionali del progetto è di fondamentale per il corretto disegno della soluzione che si dovrà adottare, in quanto si eviterà, in tal modo, di incorrere in errori difficilmente rimovibili dopo la realizzazione.

I requisiti non funzionali, molto spesso, non sono manifestati in modo consapevole dall'utente, né sono individuabili in modo evidente, è per questo fondamentale un'attenta opera di deduzione da parte del responsabile del progetto. Quindi, per ogni requisito è opportuno evidenziare con la dicitura "Esplicito" o "Implicito"(se espresso dall'utente o dedotto dal Capo Progetto) al fine di evidenziare all'utente le conseguenze che altrimenti potrebbero essere sottovalutate.

Per applicazioni Web ritenute "critiche" (dopo la compilazione del questionario presente nel documento di "Linee Guida per la definizione della criticità di applicazioni Web e per lo sviluppo di applicazioni "critiche"), devono essere individuati e codificati, secondo il formalismo del presente standard, i requisiti non funzionali relativi a specifici aspetti di sicurezza da garantire, recependo le direttive fornite e descritte dettagliatamente, dalla struttura di Sicurezza, nel documento "Requisiti di Sicurezza".

Di seguito sono elencati i requisiti che non comportano l'implementazione di funzionalità.

5.4.1 Requisito Non Funzionale : Gestione Log

Il sistema è predisposto per effettuare il log di tutte le operazioni critiche (es. accessi al db, transaction performance reporting). Tale feature si rileva di estrema importanza nel caso di problemi riscontrati in esercizio, dal momento che, abilitandolo attraverso un file di configurazione, viene prodotto un accurato trace delle operazioni effettuate rendendo il processo di identificazione di eventuali "bug" relativamente semplice.

L'adozione di Log4J rende la soluzione proposta altamente configurabile. Si ricorda che **Log4J** è una libreria Java sviluppata dalla Apache Software Foundation. Essa permette di mettere a punto un ottimo sistema di logging per tenere sotto controllo il comportamento di una applicazione, sia in fase di sviluppo che in fase di test e messa in opera del prodotto finale. Il modo migliore per configurare la libreria, ed utilizzarla in un'applicazione, è scrivere un file di properties. Il file può anche essere scritto in formato xml.

5.4.2 Requisito Non Funzionale: Accesso e Profilazione

La gestione dell'autenticazione all'accesso degli utenti dovrà essere demandata all'applicazione. A tal fine l'organizzazione delle autorizzazioni di accesso alle funzionalità del sistema dovrà essere organizzata per:

- Profili di utenza;
- Funzioni elementari del sistema;
- Associazione delle funzioni ai profili.

Inoltre il sistema avrà una gestione integrata dell'auditing delle operazioni eseguite nel sistema, realizzata monitorando in maniera generalizzata, l'esecuzione di tutte le funzioni elementari. L'audit registrerà utente, data, ora e tipo operazione eseguita con gli estremi dell'oggetto sul quale è stata eseguita e provvederà periodicamente ed automaticamente, sulla base di parametri impostati in un file di configurazione editabile, a mantenere costante la dimensione massima della tabella di Log. L'adozione di Log4J rende la soluzione proposta altamente configurabile.

5.4.3 Requisito Non Funzionale: Elenchi

Nel sistema si avranno due modalità di selezione delle entità relazionate a quella in oggetto:

- Selezione da Combo-Box – adottata nel caso di tabelle con poche occorrenze e a basso incremento e dove non si richiedono;

- Selezione con finestra di popup - adottata nel caso di tabelle con molte occorrenze o ad alto incremento o dove si richiedono filtri.

5.5 Descrizione dei Dati

Si effettua, ora, una descrizione concettuale dei dati dell' applicazione:

Entità: definizione dell'entità del modello	
Entità Nome	**Entità Definizione**
Aula	- Nome dell'aula, inteso come identificativo univoco - Edificio di ubicazione - Piano dell'edificio - Capienza in termini di posti a sedere
Dotazione	- Identificativo univoco per ogni dotazione (computer, proiettore, lavagna…) - Descrizione della dotazione
Prenotazione	- Identificativo univoco per ogni prenotazione - Riferimento all'aula soggetta a prenotazione - Riferimento al tipo di prenotazione effettuata - Data di inizio e fine della prenotazione - Orario di inizio e fine per la prenotazione (Dalle ore...Alle ore...) - Utente che ha effettuato la prenotazione
Tipo Prenotazione	- Identificativo univoco per ogni tipo di prenotazione - Descrizione del motivo della prenotazione
Personale	- Identificativo univoco dell'utente - Nome - Cognome

Entità: definizione dell'entità del modello	
Entità Nome	**Entità Definizione**
	- Titolo posseduto dall'utente nell'ambito dipartimentale
Aula Dotazione	Entità che ha il compito di collegare l'entità Aula con l'entità Dotazione che è quindi da considerare come una tipologica, ed aggiunge l'attributo quantità.
Prenotazione Personale	Entità che ha il compito di collegare l'entità Prenotazione con l'entità Personale, ed aggiunge l'attributo ruolo, che ha il compito di descrivere il ruolo svolto da un utente nell'effettuare una prenotazione.

5.6 Specifica dei Requisiti

La specifica è una descrizione completa del comportamento di un sistema da sviluppare. Essa rappresenta un sotto-campo dell'ingegneria del software che si occupa del concepimento, dell'analisi, della specifica, e della validazione dei requisiti per il software.
Il documento di Specifica dei Requisiti elenca, come si è visto, tutti i requisiti necessari per lo sviluppo del progetto, ma per ottenere i requisiti di cui si ha bisogno, è necessaria una comprensione chiara e completa dei prodotti da sviluppare, per questo saranno riportati di seguito i diversi diagrammi dei casi d'uso.

5.6.1 Diagramma generale dei casi d'uso relativi all'applicazione Space ClassRoom

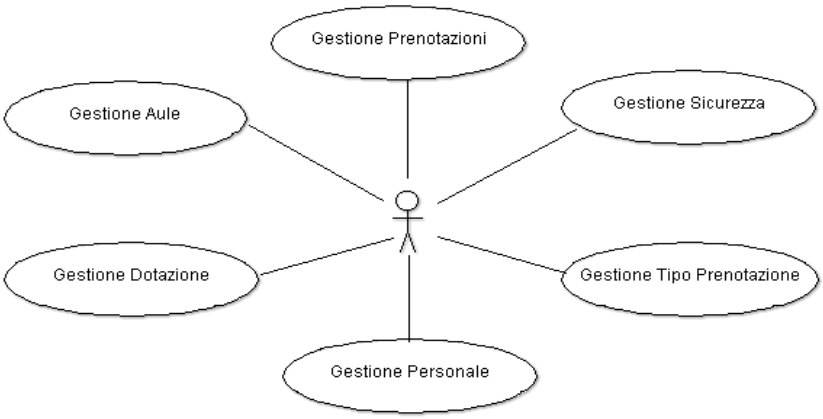


Figura 28. Macrofunzioni dell'applicazione

Di seguito sono riportati i casi d'uso relativi alle tipologie di utenti che utilizzano l'applicazione Space ClassRoom, e alle macrofunzioni.

5.6.2 Casi d'uso relativi alla tipologia di utenti Amministratore

Figura 29. Casi d'uso dell'Ammistratore

L'amministratore è una particolare tipologia di utente, in quanto ha il compito di gestire tutte le operazioni elementari delle macrofunzioni:

- GESTIONE PRENOTAZIONE
- GESTIONE AULE
- GESTIONE DOTAZIONE
- GESTIONE PERSONALE
- GESTIONE TIPO PRENOTAZIONE
- GESTIONE SICUREZZA

L'amministratore è una particolare tipologia di utente, in quanto ha il compito di gestire le informazioni riguardanti le aule, tramite la macrofunzione GESTIONE AULE. Egli potrà, infatti, visualizzare, inserire, modificare ed eliminare le informazioni relative a tutte le aule del DIETI. Le informazioni caratteristiche delle aule sono proprie della tabella AULE e, dunque, sono: nome identificativo dell'aula, capienza, e ubicazione (edificio e piano). L'amministratore accede anche a tutte le operazioni elementari delle macrofunzioni: GESTIONE PRENOTAZIONI. Egli potrà, dunque, controllare le informazioni circa le prenotazioni, potrà modificare o eliminare una prenotazione che risiede già nel database, inoltre accedendo ai dettagli di una prenotazione, oltre a visualizzare le informazioni ad essa correlate, si potrà modificare/aggiungere personale impegnato nella prenotazione. Inoltre, l'amministratore ha il compito di gestire completamente l'anagrafica delle dotazioni, del personale e del tipo di prenotazione, in quanto può usufruire delle macrofunzioni: GESTIONE DOTAZIONE, GESTIONE PERSONALE e GESTIONE TIPO PRENOTAZIONE. Infine, l'amministratore ha il compito di gestire la sicurezza dell'applicazione attraverso l'utilizzo della macrofunzione GESTIONE SICUREZZA. Egli avrà il compito di gestire non solo gli utenti, ma anche i profili di utenza, che saranno assegnati ad ogni utente in modo tale da differenziare l'utilizzo e la gestione dell'applicazione a seconda di chi vi accede. L'amministratore potrà, dunque, inserire un nuovo utente nel database assegnandoli un profilo, modificare le informazioni di un utente già presente nel database ed, infine, eliminare un utente già registrato.

5.6.3 Casi d'uso relativi alla tipologia di utenti Utente registrato

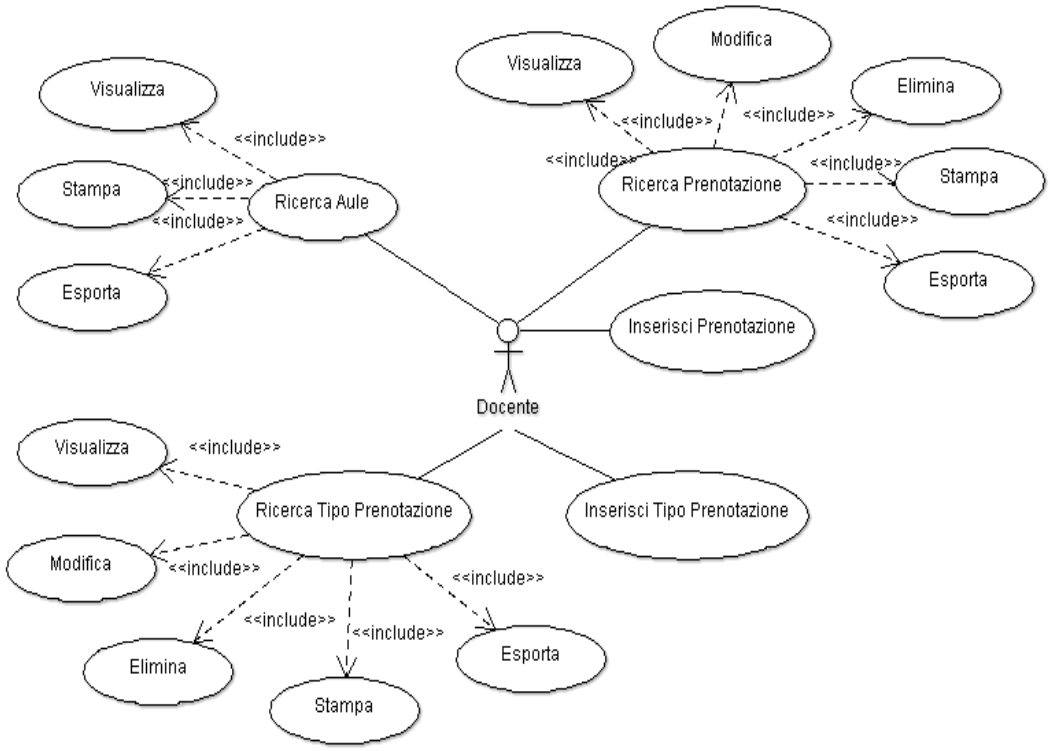


Figura 30. Casi d'uso dell'utente registrato come Docente

L'utente registrato può essere un professore o uno studente. Egli, a seconda dei privilegi, potrà utilizzare la macrofunzione GESTIONE PRENOTAZIONI. Tale macrofunzione, infatti, dà la possibilità all'utente registrato di poter visualizzare le prenotazioni già effettuate per una determinata aula selezionata o per tutte le aule del DIETI, di poter effettuare una prenotazione indicando aula, orario e motivazione della prenotazione, ed, infine, di poter modificare e cancellare una prenotazione effettuata. L'utente registrato ha, inoltre, la possibilità di poter visualizzare tutte le informazioni delle aule del dipartimento (GESTIONE AULE), ma a differenza dell'amministratore non potrà in alcun modo modificare le informazioni relative alle aule, a meno che non venga autorizzato nel farlo da un amministratore. La differenza tra un docente e uno studente sta nel numero di operazioni elementari che entrambi possono svolgere; infatti, è vero che entrambi fanno uso della macrofunzione GESTIONE PRENOTAZIONI, ma se il docente può accedere a tutte le

operazioni elementari della macrofunzione, lo studente non può. Lo studente, infatti, potrà effettuare la ricerca delle prenotazioni e analizzare i dettagli di ognuna, ma non potrà inserire, modificare o eliminare.

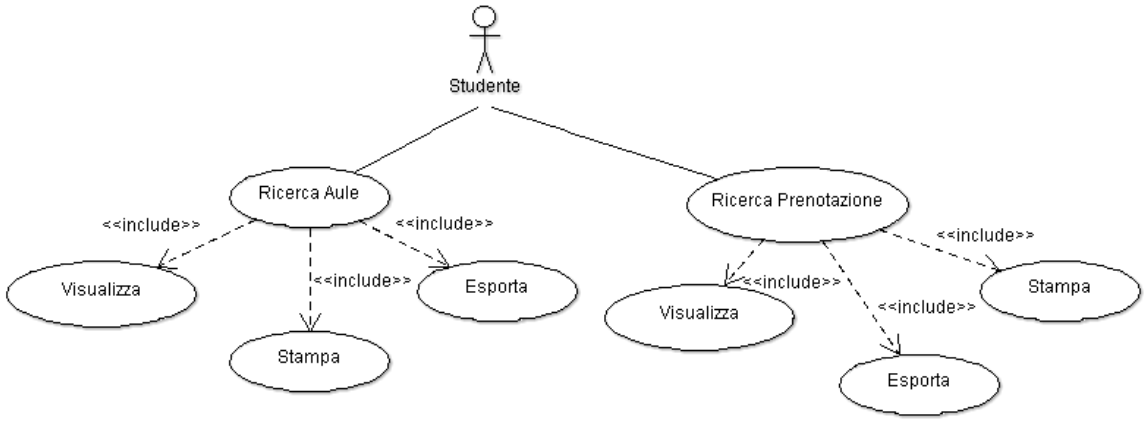


Figura 31. Casi d'uso dell'utente registrato come Studente

5.6.4 Casi d'uso relativi alla tipologia di utenti Ospite

L'utente ospite può accedere alla sola macrofunzione GESTIONE AULE, ma non potrà utilizzare tutte le operazioni elementari che la compongono, infatti, potrà solo effettuare la ricerca delle aula e visualizzare i dettagli dell'aula selezionata (edificio, piano e numero posti).

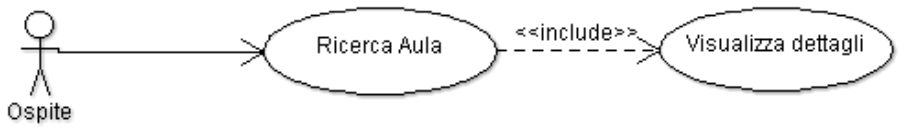


Figura 32. Casi d'uso dell'utente ospite

5.6.5 Casi d'uso relativi alla macrofunzione Gestione Prenotazioni

La macrofunzione GESTIONE PRENOTAZIONE può essere utilizzata solo da un utente con appropriati privilegi, in quanto essa permette di poter visualizzare le prenotazioni già effettuate per una determinata aula selezionata o per tutte le aule del DIETI, di poter effettuare una prenotazione indicando aula, orario o motivazione della prenotazione, ed, infine, di poter modificare e cancellare una prenotazione effettuata. È possibile, inoltre, avere un elenco completo di tutte le prenotazioni effettuate, esse possono essere stampata o salvate in un file in formato RTF o PDF. Verranno di seguito analizzate alcuni scenari dell'utilizzo della suddetta macrofunzione:

- Inserimento di una nuova Prenotazione
- Ricerca delle Prenotazioni effettuate
- Visualizzazione, Modifica, Eliminazione delle Prenotazioni

Inserimento di una nuova Prenotazione

SCENARIO PRINCIPALE

L'utente registrato come amministratore, oppure come docente potrà effettuare una nuova prenotazione inserendo tutti i relativi parametri.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Prenotazione	Il sistema mostra la pagina di Ricerca Prenotazione
2	L'utente richiede l'inserimento di un elemento Prenotazione, premendo il pulsante "Nuovo".	Il sistema mostra la pagina di dettaglio dei dati della prenotazione

3	L'utente inserisce i dati della Prenotazione, quindi preme il pulsante “Salva”.	Il sistema salva i dati della Prenotazione, confermando la corretta esecuzione dell'operazione d’inserimento attraverso apposito messaggio.

VARIANTE: ANNULLA OPERAZIONE DI INSERIMENTO PRENOTAZIONE

L’annullamento di un'operazione d’inserimento dei dati di una Prenotazione, è effettuata dall'utente attraverso l’apposito pulsante “Annulla”.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Prenotazione.	Il sistema mostra la pagina di Ricerca Prenotazione.
2	L'utente richiede l'inserimento di un elemento Prenotazione, premendo il pulsante “Nuovo”.	Il sistema mostra la pagina di dettaglio dei dati della prenotazione.
3	L'utente inserisce o meno i dati della Prenotazione, ma preme il pulsante “Annulla”	Il sistema annulla l'operazione di inserimento e ritorna alla schermata precedente.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Nome Aula	Indica il nome univoco dell'aula	Obbligatorio	20	String	I
Motivo Prenotazione	Indica il motivo per cui è effettuata una prenotazione	Obbligatorio	-	theTipoPrenotazione	I
Data Inizio	Indica la data di inizio della Prenotazione	Obbligatorio	-	Date	I
Ora Inizio	Indica l'ora di inizio della Prenotazione	Obbligatorio	9	String	I
Data Fine	Indica la data di fine della Prenotazione	Obbligatorio	-	Date	I
Ora Fine	Indica l'ora di inizio della Prenotazione	Obbligatorio	9	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante "Salva"	Effettua l'inserimento dell'elemento in archivio
Pulsante "Annulla"	Annulla l'operazione di inserimento e ritorna alla pagina precedente

I controlli da eseguire sono:

Controllo	Descrizione
1	In caso di inserimento di una nuova Prenotazione il sistema controlla la presenza dell'elemento in archivio. Se è già presente, impedisce l'operazione e notifica l'errore attraverso apposito messaggio.

Ricerca Delle Prenotazioni Effettuate

Il sistema, attraverso le funzionalità di Ricerca Prenotazione, consente di specificare diversi criteri di ricerca. Il risultato della ricerca è riportato in un elenco paginato ed ordinabile.

Attraverso le schermate di ricerca l'utente può effettuare:

1. Ricerca Generica;
2. Ricerca per Nome Aula;
3. Ricerca per nome Edificio;
4. Ricerca per Motivo della Prenotazione;
5. Ricerca per Data Inizio e/o Data Fine;
6. Ricerca per Ora Inizio e/o Ora Fine

DESCRIZIONE DELL'IMPATTO

La funzionalità di Ricerca Generica di una Prenotazione potrà essere adoperata da un utente autorizzato e permetterà di ottenere la lista di tutte le prenotazioni effettuate.

A partire da tale tabella completa, l'utente potrà decidere di visualizzare l'output della ricerca nei vari formati previsti dal sistema (excel e pdf).

SCENARIO PRINCIPALE

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Almeno un elemento Prenotazione deve essere presente in archivio.

3. Sessione aperta sull'applicazione, utente posizionato sulla Home page

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Prenotazione	Il sistema mostra la pagina di Ricerca Prenotazione.
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante “Ricerca”.	Il sistema mostra la pagina di visualizzazione elenco Prenotazioni riportando una riga per ogni elemento qualificato.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Order	Ordine di visualizzazione della colonna	F	1	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante “Excel”	Esporta la tabella, ottenuta come risultato della ricerca, in formato Excel.
Pulsante “Pdf”	Esporta la tabella, ottenuta come risultato della ricerca, in formato PDF.
Pulsante “Ricerca”	Effettua l’operazione di ricerca
Pulsante “Nuovo”	Effettua l'inserimento dell'elemento in archivio

Visualizzazione, Modifica, Eliminazione delle Prenotazioni

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Prenotazione.	Il sistema mostra la pagina di Ricerca Prenotazione.
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante “Ricerca”.	Il sistema mostra la pagina di “visualizzazione elenco Prenotazioni”, riportando una riga per ogni elemento qualificato.
3	L'utente seleziona una delle Prenotazioni presenti nella tabella, ovvero sceglie una riga della tabella attraverso il codice “Prenotazione Id”.	Il sistema mostra la pagina di Dettaglio Prenotazione, in cui è possibile visualizzare tutti i dati della Prenotazione.
4	L’utente può modificare/eliminare i dati della Prenotazione inseriti premendo il pulsante “Modifica”.	Il sistema permetterà all’utente di modificare non solo i parametri propri della Prenotazione, ma anche di aggiungere/modificare il Personale relativo alla Prenotazione selezionata.

VARIANTE: ELIMINA PRENOTAZIONE EFFETTUATA

L’ eliminazione di una Prenotazione effettuata da un utente può avvenire attraverso l’apposito pulsante “Elimina”.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla pagina “Ricerca Prenotazione”.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la riga della tabella, contenente tutte le prenotazioni, che rappresenta la prenotazione che si vuole eliminare.	Il sistema mostra la pagina di Dettaglio della Prenotazione.
2	L'utente richiede l’eliminazione della Prenotazione, premendo il pulsante “Elimina”.	Il sistema mostra il messaggio di avvenuta eliminazione e torna alla pagina precedente.

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante “Indietro”	Ritorna alla pagina precedente
Pulsante “Modifica”	Permette di modificare i parametri della Prenotazione
Pulsante “Nuovo”	Permette di inserire i dati per una nuova Prenotazione
Pulsante “Stampa PDF”	Permette di stampare in formato PDF la Prenotazione
Pulsante “Produci RTF”	Permette di riprodurre in formato RTF la Prenotazione
Pulsante “Produci PDF”	Permette di riprodurre in formato PDF la Prenotazione
Pulsante “Eimina”	Permette di eliminare l’eliminazione della Prenotazione selezionata

5.6.6 Casi d'uso relativi alla macrofunzione Gestione Aule

La funzionalità in oggetto potrà essere utilizzata dagli utenti registrati come docenti, studenti e amministratori. In particolare sia i docenti che gli amministratori potranno accedere a tutte le operazioni elementari della macrofunzione, Essa permetterà, infatti, di visualizzare, inserire, modificare ed eliminare le informazioni relative a tutte le aule del DIETI. Verranno di seguito analizzate alcuni scenari dell'utilizzo della suddetta macrofunzione:

- Inserimento di una nuova Aula
- Ricerca delle Aule presenti in archivio
- Visualizzazione, Modifica, Eliminazione delle Aule

Inserimento di una nuova Aula

SCENARIO PRINCIPALE

L'utente registrato come amministratore potrà inserire una nuova aula nell'archivio, includendo anche tutti i relativi dati.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Aule.	Il sistema mostra la pagina di Ricerca Aule.
2	L'utente richiede l'inserimento di un elemento Aula, premendo il pulsante "Nuovo".	Il sistema mostra la pagina di dettaglio dei dati dell' aula.
3	L'utente inserisce i dati relativi	Il sistema salva i dati dell'Aula

	all'aula, quindi preme il pulsante "Salva".	confermando la corretta esecuzione dell'operazione di inserimento attraverso apposito messaggio.

VARIANTE: ANNULLA OPERAZIONE DI INSERIMENTO AULA

L'annullamento di un'operazione di inserimento dei dati relativi ad un'aula, è effettuata dall'utente attraverso l'apposito pulsante "Annulla".

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Aule.	Il sistema mostra la pagina di Ricerca Aule.
2	L'utente richiede l'inserimento di un elemento Aula, premendo il pulsante "Nuovo".	Il sistema mostra la pagina di dettaglio dei dati dell'Aula.
3	L'utente inserisce o meno i dati relativi all'Aula, ma preme il pulsante "Annulla".	Il sistema annulla l'operazione di inserimento e ritorna alla schermata precedente.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Nome Aula	Indica il nome univoco dell'aula	Obbligatorio	20	String	I
Edificio	Indica il nome	Obbligatorio	40	String	I

	dell'edificio				
Piano dell'Edificio	Indica il piano, dove è situata l'aula	Obbligatorio	10	String	I
Numero Posti	Indica il numero di posti a sedere dell'aula.	Facoltativo	4	Integer	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante "Salva"	Effettua l'inserimento dell'elemento in archivio
Pulsante "Annulla"	Annulla l'operazione di inserimento e ritorna alla pagina precedente

I controlli da eseguire sono:

Controllo	Descrizione
1	In caso di inserimento di una nuova Aula il sistema controlla la presenza dell'elemento in archivio. Se è già presente, impedisce l'operazione e notifica l'errore attraverso apposito messaggio.

Ricerca delle Aula memorizzate

Il sistema, attraverso le funzionalità di Ricerca Aule, consente di specificare diversi criteri di ricerca. Il risultato della ricerca viene riportato in un elenco paginato ed ordinabile.

Attraverso le schermate di ricerca l'utente può effettuare:

1. Ricerca Generica;
2. Ricerca per Nome Aula;
3. Ricerca per nome Edificio;
4. Ricerca per Numero Posti (da…a);

Esaminiamo nel dettaglio la funzionalità di Ricerca Generica.

DESCRIZIONE DELL'IMPATTO

La funzionalità di Ricerca Generica di un' Aula potrà essere adoperata da un utente autorizzato e permetterà di ottenere la lista di tutte le aule presenti in archivio.

A partire da tale tabella completa, l'utente potrà decidere di visualizzare l'output della ricerca nei vari formati previsti dal sistema (excel e pdf).

SCENARIO PRINCIPALE

PRECONDIZIONI

1) Utente abilitato all'esecuzione della funzione elementare.

2) Almeno un elemento Aula deve essere presente in archivio.

3) Sessione aperta sull'applicazione, utente posizionato sulla Home page

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Aule.	Il sistema mostra la pagina di Ricerca Aule
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante "Ricerca".	Il sistema mostra la pagina di "visualizzazione elenco Aule" riportando una riga per ogni elemento qualificato.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Order	Ordine di visualizzazione della colonna	Facoltativo	1	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante “Excel”	Esporta la tabella, ottenuta come risultato della ricerca, in formato Excel.
Pulsante “Pdf”	Esporta la tabella, ottenuta come risultato della ricerca, in formato PDF.
Pulsante “Ricerca”	Effettua l’operazione di ricerca
Pulsante “Nuovo”	Effettua l'inserimento dell'elemento in archivio

Visualizzazione, Modifica, Eliminazione di un elemento Aula

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Aule.	Il sistema mostra la pagina di Ricerca Aule.
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante “Ricerca”.	Il sistema mostra la pagina di “visualizzazione elenco Aule” riportando una riga per ogni elemento qualificato.
3	L'utente seleziona una delle aule presenti nella tabella, ovvero sceglie una riga della tabella attraverso il codice “Aula Id”.	Il sistema mostra la pagina di Dettaglio Aula, in cui è possibile visualizzare tutti i dati dell’aula.
4	L’utente può modificare/eliminare i	Il sistema permetterà all’utente di

	dati dell'aula inseriti premendo il pulsante "Modifica".	modificare i parametri propri dell'elemento Aula.

VARIANTE: ELIMINA AULA MEMORIZZATA

L'eliminazione di un'Aula memorizzata da un utente può avvenire attraverso l'apposito pulsante "Elimina".

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla pagina "Ricerca Aule"

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la riga della tabella, contenente tutte le aule memorizzate, che rappresenta l'aula che si vuole eliminare.	Il sistema mostra la pagina di Dettaglio dell'Aula.
2	L'utente richiede l'eliminazione dell'Aula, premendo il pulsante "Elimina".	Il sistema mostra il messaggio di avvenuta eliminazione e torna alla pagina precedente.

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante "Indietro"	Ritorna alla pagina precedente
Pulsante "Modifica"	Permette di modificare i parametri dell'Aula.
Pulsante "Nuovo"	Permette di inserire i dati per una nuova Aula.
Pulsante "Stampa PDF"	Permette di stampare in formato PDF l'Aula.
Pulsante "Produci RTF"	Permette di riprodurre in formato RTF l'Aula.
Pulsante "Produci PDF"	Permette di riprodurre in formato PDF l'Aula.
Pulsante "Eimina"	Permette di eliminare l'eliminazione dell'Aula selezionata.

5.6.7 Casi d'uso relativi alla macrofunzione Gestione Dotazioni

La funzionalità in oggetto potrà essere utilizzata dagli utenti registrati come amministratori. Questi ultimi potranno accedere a tutte le operazioni elementari della macrofunzione, Essa permetterà, infatti, di visualizzare, inserire, modificare ed eliminare le informazioni relative alle dotazioni delle aule del DIETI. Verranno di seguito analizzate alcuni scenari dell'utilizzo della suddetta macrofunzione:

- Inserimento di una nuova Dotazione
- Ricerca delle Dotazione presenti in archivio
- Visualizzazione, Modifica, Eliminazione delle Dotazioni

Inserimento di una nuova Dotazione

SCENARIO PRINCIPALE

L'utente registrato come amministratore potrà inserire una nuova dotazione nell'archivio, includendo anche tutti i relativi dati.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Dotazione.	Il sistema mostra la pagina di Ricerca Dotazione.
2	L'utente richiede l'inserimento di un elemento Dotazione, premendo il pulsante "Nuovo".	Il sistema mostra la pagina di dettaglio dei dati della Dotazione.
3	L'utente inserisce i dati della Dotazione, quindi preme il pulsante "Salva".	Il sistema salva i dati della Dotazione confermando la corretta esecuzione dell'operazione di inserimento attraverso apposito messaggio.

VARIANTE: ANNULLA OPERAZIONE DI INSERIMENTO DOTAZIONE

L'annullamento di un'operazione di inserimento dei dati di una Dotazione, è effettuata dall'utente attraverso l'apposito pulsante "Annulla".

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Dotazione.	Il sistema mostra la pagina di Ricerca Dotazione.
2	L'utente richiede l'inserimento di un elemento Dotazione, premendo il pulsante "Nuovo".	Il sistema mostra la pagina di dettaglio dei dati della Dotazione.
3	L'utente inserisce o meno i dati della Dotazione, ma preme il pulsante "Annulla".	Il sistema annulla l'operazione di inserimento e ritorna alla schermata precedente.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Dotazione Id	Indica l'identificativo univoco della dotazione.	Obbligatorio	9	Integer	I
Descrizione	Indica il nome della dotazione	Obbligatorio	40	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante "Salva"	Effettua l'inserimento dell'elemento in archivio.
Pulsante "Annulla"	Annulla l'operazione di inserimento e ritorna alla pagina precedente.

I controlli da eseguire sono:

Controllo	Descrizione
1	In caso di inserimento di una nuova Dotazione il sistema controlla la presenza dell'elemento in archivio. Se è già presente, impedisce l'operazione e notifica l'errore attraverso apposito messaggio.

Ricerca delle Dotazioni memorizzate

Il sistema, attraverso le funzionalità di Ricerca Dotazione, consente di specificare diversi criteri di ricerca. Il risultato della ricerca viene riportato in un elenco paginato ed ordinabile.

Attraverso le schermate di ricerca l'utente può effettuare:

1. Ricerca Generica;
2. Ricerca per Nome Dotazione;

DESCRIZIONE DELL'IMPATTO

La funzionalità di Ricerca Generica di una Dotazione potrà essere adoperata da un utente autorizzato e permetterà di ottenere la lista di tutte le dotazioni presenti in archivio. A partire da tale tabella completa, l'utente potrà decidere di visualizzare l'output della ricerca nei vari formati previsti dal sistema (excel e pdf).

SCENARIO PRINCIPALE

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Almeno un elemento Dotazione deve essere presente in archivio.
3. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Dotazione.	Il sistema mostra la pagina di Ricerca Dotazione.
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante “Ricerca”.	Il sistema mostra la pagina di “visualizzazione elenco Dotazioni” riportando una riga per ogni elemento qualificato.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Order	Ordine di visualizzazione della colonna	F	1	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante “Excel”	Esporta la tabella, ottenuta come risultato della ricerca, in formato Excel.
Pulsante “Pdf”	Esporta la tabella, ottenuta come risultato della ricerca, in formato PDF.

Pulsante “Ricerca”	Effettua l’operazione di ricerca.
Pulsante “Nuovo”	Fa l'inserimento dell'elemento in archivio.

Visualizzazione, Modifica, Eliminazione di un elemento Dotazione.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Dotazione.	Il sistema mostra la pagina di Ricerca Dotazione.
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante “Ricerca”.	Il sistema mostra la pagina di “visualizzazione elenco Dotazione” riportando una riga per ogni elemento qualificato.
3	L'utente seleziona una delle Dotazioni presenti nella tabella, ovvero sceglie una riga della tabella attraverso il codice “Dotazione Id”.	Il sistema mostra la pagina di Dettaglio Dotazione, in cui è possibile visualizzare tutti i dati della dotazione.
4	L’utente può modificare/eliminare i dati della Dotazione inseriti premendo il pulsante “Modifica”.	Il sistema permetterà all’utente di modificare i parametri propri dell’elemento Dotazione.

VARIANTE: ELIMINA DOTAZIONE MEMORIZZATA

L’eliminazione di una Dotazione memorizzata da un utente può avvenire attraverso l’apposito pulsante “Elimina”.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla pagina “Ricerca Dotazione”.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la riga della tabella, contenente tutte le Dotazioni memorizzate, che rappresentano la dotazione che si vuole eliminare.	Il sistema mostra la pagina di Dettaglio Dotazione.
2	L'utente richiede l’eliminazione della Dotazione, premendo il pulsante “Elimina”.	Il sistema mostra il messaggio di avvenuta eliminazione e torna alla pagina precedente.

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante “Indietro”	Ritorna alla pagina precedente
Pulsante “Modifica”	Permette di modificare i parametri della Dotazione.
Pulsante “Nuovo”	Permette di inserire i dati per una nuova Dotazione.
Pulsante “Stampa PDF”	Permette di stampare in formato PDF della Dotazione.
Pulsante “Produci RTF”	Permette di riprodurre in formato RTF della Dotazione.
Pulsante “Produci PDF”	Permette di riprodurre in formato PDF della Dotazione.
Pulsante “Eimina”	Permette di eliminare l’eliminazione della Dotazione.

5.6.8 Casi d'uso relativi alla macrofunzione Gestione Personale

La funzionalità in oggetto potrà essere utilizzata dagli utenti registrati come amministratori. Questi ultimi potranno accedere a tutte le operazioni elementari della macrofunzione, Essa permetterà, infatti, di visualizzare, inserire, modificare ed eliminare le informazioni relative al personale del DIETI. Verranno di seguito analizzate alcuni scenari dell'utilizzo della suddetta macrofunzione:

- Inserimento di un nuovo elemento Personale
- Ricerca del Personale presente in archivio
- Visualizzazione, Modifica, Eliminazione del Personale

Inserimento di un nuovo elemento Personale

SCENARIO PRINCIPALE

L'utente registrato come amministratore potrà inserire un nuove elemento Personale nell'archivio, includendo anche tutti i relativi dati.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Personale.	Il sistema mostra la pagina di Ricerca Personale.
2	L'utente richiede l'inserimento di un elemento Personale, premendo il pulsante "Nuovo".	Il sistema mostra la pagina di dettaglio dei dati del Personale.
3	L'utente inserisce i dati dell'elemento Personale, e quindi preme il pulsante "Salva".	Il sistema salva i dati relativi al Personale aggiunto confermando la corretta esecuzione dell'operazione di inserimento attraverso apposito messaggio.

VARIANTE: ANNULLA OPERAZIONE DI INSERIMENTO PERSONALE

L'annullamento di un'operazione di inserimento dei dati di un elemento Personale, è effettuata dall'utente attraverso l'apposito pulsante "Annulla".

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Personale.	Il sistema mostra la pagina di Ricerca Personale.
2	L'utente richiede l'inserimento di un elemento Personale, premendo il pulsante "Nuovo".	Il sistema mostra la pagina di dettaglio dei dati del Personale.
3	L'utente inserisce o no i dati dell'elemento Personale che si sta aggiungendo, ma preme il pulsante "Annulla".	Il sistema annulla l'operazione di inserimento e ritorna alla schermata precedente.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Nome	Indica il nome del Personale	Obbligatorio	30	String	I
Cognome	Indica il cognome del Personale	Obbligatorio	30	String	I
Titolo	Indica il titolo del Personale	Obbligatorio	30	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante "Salva"	Effettua l'inserimento dell'elemento in archivio.
Pulsante "Annulla"	Annulla l'operazione di inserimento e ritorna alla pagina precedente.

I controlli da eseguire sono:

Controllo	Descrizione
1	In caso di inserimento di un nuovo elemento Personale, il sistema controlla la presenza dell'elemento in archivio. Se è già presente, impedisce l'operazione e notifica l'errore attraverso apposito messaggio.

Ricerca del Personale presente in archivio

Il sistema, attraverso le funzionalità di Ricerca Personale, consente di specificare diversi criteri di ricerca. Il risultato della ricerca è riportato in un elenco paginato ed ordinabile.
Attraverso le schermate di ricerca l'utente può effettuare:

1. Ricerca Generica;
2. Ricerca per Nome;
3. Ricerca per Cognome;
4. Ricerca per Titolo;

DESCRIZIONE DELL'IMPATTO

La funzionalità di Ricerca Generica di un elemento Personale potrà essere adoperata da un utente autorizzato e permetterà di ottenere la lista di tutte le dotazioni presenti in archivio. A partire da tale tabella completa, l'utente potrà decidere di visualizzare l'output della ricerca nei vari formati previsti dal sistema (excel e pdf).

SCENARIO PRINCIPALE

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Almeno un elemento Personale deve essere presente in archivio.
3. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Personale.	Il sistema mostra la pagina di Ricerca Personale.
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante "Ricerca".	Il sistema mostra la pagina di "visualizzazione elenco Personale" riportando una riga per ogni elemento qualificato.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Order	Ordine di visualizzazione della colonna	F	1	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante "Excel"	Esporta la tabella, ottenuta come risultato della ricerca, in formato Excel.
Pulsante "Pdf"	Esporta la tabella, ottenuta come risultato della ricerca, in formato PDF.
Pulsante "Ricerca"	Effettua l'operazione di ricerca

Pulsante “Nuovo”	Effettua l'inserimento dell'elemento in archivio.

Visualizzazione, Modifica, Eliminazione di un elemento Personale.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Personale.	Il sistema mostra la pagina di Ricerca Personale.
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante “Ricerca”.	Il sistema mostra la pagina di “visualizzazione elenco Personale” riportando una riga per ogni elemento qualificato.
3	L'utente seleziona uno degli elementi del Personale presenti nella tabella, ovvero sceglie una riga della tabella attraverso il codice “Personale Id”.	Il sistema mostra la pagina di Dettaglio Personale, in cui è possibile visualizzare tutti i dati dell’elemento selezionato.
4	L’utente può modificare/eliminare i dati dell’elemento inserito premendo il pulsante “Modifica”.	Il sistema permetterà all’utente di modificare i parametri propri dell’elemento Personale.

VARIANTE: ELIMINA PERSONALE PRESENTO IN ARCHIVIO

L'eliminazione di un elemento Personale memorizzato da un utente può avvenire attraverso l'apposito pulsante "Elimina".

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla pagina "Ricerca Personale".

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la riga della tabella, contenente tutto il personale memorizzato, che rappresenta la dotazione che si vuole eliminare.	Il sistema mostra la pagina di Dettaglio Personale.
2	L'utente richiede l'eliminazione del Personale, premendo il pulsante "Elimina".	Il sistema mostra il messaggio di avvenuta eliminazione e torna alla pagina precedente.

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante "Indietro"	Ritorna alla pagina precedente
Pulsante "Modifica"	Permette di modificare i parametri dell'elemento Personale.
Pulsante "Nuovo"	Permette di inserire i dati per un nuovo elemento Personale.
Pulsante "Stampa PDF"	Permette di stampare in formato PDF dell'elemento Personale.

Pulsante “Produci RTF”	Permette di riprodurre in formato RTF dell’elemento Personale.
Pulsante “Produci PDF”	Permette di riprodurre in formato PDF dell’elemento Personale.
Pulsante “Eimina”	Permette di eliminare l’eliminazione dell’elemento Personale.

5.6.9 Casi d’uso relativi alla macrofunzione Gestione Tipo Prenotazioni

La funzionalità in oggetto potrà essere utilizzata dagli utenti registrati come docenti, e amministratori. In particolare sia i docenti sia gli amministratori potranno accedere a tutte le operazioni elementari della macrofunzione, Essa permetterà, infatti, di visualizzare, inserire, modificare ed eliminare le informazioni relative ai tipi di prenotazione tutte. Verranno di seguito analizzate alcuni scenari dell’utilizzo della suddetta macrofunzione:

- Inserimento di un nuovo Tipo di Prenotazione
- Ricerca del Tipo di Prenotazione presente in archivio.
- Visualizzazione, Modifica, Eliminazione del Tipo di Prenotazione.

Inserimento di una nuova Aula

SCENARIO PRINCIPALE

L’utente, sia esso amministratore sia docente, potrà inserire un nuovo Tipo di Prenotazione nell’archivio, includendo anche tutti i relativi dati.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla	Il sistema mostra la pagina di Ricerca Tipo Prenotazione.

	funzione: Gestione Tipo Prenotazione.	
2	L'utente richiede l'inserimento di un elemento Tipo Prenotazione, premendo il pulsante “Nuovo”.	Il sistema mostra la pagina di dettaglio del Tipo Prenotazione.
3	L'utente inserisce i dati relativi al Tipo Prenotazione, e quindi preme il pulsante “Salva”.	Il sistema salva i dati del Tipo Prenotazione confermando la corretta esecuzione dell'operazione di inserimento attraverso apposito messaggio.

VARIANTE: ANNULLA OPERAZIONE DI INSERIMENTO TIPO PRENOTAZIONE

L’annullamento di un'operazione di inserimento dei dati relativi ad un Tipo Prenotazione, è effettuata dall'utente attraverso l’apposito pulsante “Annulla”.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Tipo Prenotazione.	Il sistema mostra la pagina di Ricerca Tipo Prenotazione.
2	L'utente richiede l'inserimento di un elemento Tipo Prenotazione, premendo il pulsante “Nuovo”.	Il sistema mostra la pagina di dettaglio dei dati del Tipo Prenotazione.
3	L'utente inserisce o meno i dati relativi al Tipo Prenotazione, ma preme il pulsante “Annulla”.	Il sistema annulla l'operazione di inserimento e ritorna alla schermata precedente.

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Tipo Prenotazione Id	Indica l'identificativo univoco del Tipo di Prenotazione.	Obbligatorio	9	Integer	I
Descrizione	Indica il nome del Tipo di Prenotazione	Obbligatorio	40	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante "Salva"	Effettua l'inserimento dell'elemento in archivio.
Pulsante "Annulla"	Annulla l'operazione di inserimento e ritorna alla pagina precedente.

I controlli da eseguire sono:

Controllo	Descrizione
1	In caso di inserimento di un nuovo Tipo Prenotazione il sistema controlla la presenza dell'elemento in archivio. Se è già presente, impedisce l'operazione e notifica l'errore attraverso apposito messaggio.

Ricerca del Tipo Prenotazione memorizzato

Il sistema, attraverso le funzionalità di Ricerca Tipo Prenotazione, consente di specificare diversi criteri di ricerca. Il risultato della ricerca viene riportato in un elenco paginato ed ordinabile.

Attraverso le schermate di ricerca l'utente può effettuare:

1. Ricerca Generica;
2. Ricerca per Motivo della Prenotazione;

DESCRIZIONE DELL'IMPATTO

La funzionalità di Ricerca Generica di un Tipo Prenotazione potrà essere adoperata da un utente autorizzato e permetterà di ottenere la lista di tutti i tipi di prenotazione presenti in archivio. A partire da tale tabella completa, l'utente potrà decidere di visualizzare l'output della ricerca nei vari formati previsti dal sistema (excel e pdf).

SCENARIO PRINCIPALE

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Almeno un elemento Tipo Prenotazione deve essere presente in archivio.
3. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Tipo Prenotazione.	Il sistema mostra la pagina di Ricerca Tipo Prenotazione.
2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio,	Il sistema mostra la pagina di "visualizzazione elenco Tipo Prenotazione" riportando una riga per ogni elemento qualificato.

	oppure non inserisce alcun parametro, e preme il pulsante “Ricerca”.	

DATI DELLA FUNZIONE

I campi della funzione sono i seguenti:

Campi	Descrizione	Obbl/Fac	Long	Formato	I/O
Order	Ordine di visualizzazione della colonna	F	1	String	I

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	Descrizione
Pulsante “Excel”	Esporta la tabella, ottenuta come risultato della ricerca, in formato Excel.
Pulsante “Pdf”	Esporta la tabella, ottenuta come risultato della ricerca, in formato PDF.
Pulsante “Ricerca”	Effettua l’operazione di ricerca
Pulsante “Nuovo”	Effettua l'inserimento dell'elemento in archivio.

Visualizzazione, Modifica, Eliminazione di un elemento Tipo Prenotazione.

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione.
2. Sessione aperta sull'applicazione, utente posizionato sulla Home page.

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la voce di menu corrispondente alla funzione: Gestione Tipo Prenotazione.	Il sistema mostra la pagina di Ricerca Tipo Prenotazione.

2	L'utente immette parametri di ricerca corrispondenti a elementi presenti in archivio, oppure non inserisce alcun parametro, e preme il pulsante "Ricerca".	Il sistema mostra la pagina di "visualizzazione elenco Tipo Prenotazione" riportando una riga per ogni elemento qualificato.
3	L'utente seleziona un dei tipi di prenotazione presenti nella tabella, ovvero sceglie una riga della tabella attraverso il codice "Tipo Prenotazione Id".	Il sistema mostra la pagina di Dettaglio Tipo Prenotazione, in cui è possibile visualizzare tutti i dati del Tipo Prenotazione.
4	L'utente può modificare/eliminare i dati dell'aula inseriti premendo il pulsante "Modifica".	Il sistema permetterà all'utente di modificare i parametri propri dell'elemento Tipo Prenotazione.

VARIANTE: ELIMINA TIPO PRENOTAZIONE MEMORIZZATO

L'eliminazione di un Tipo Prenotazione memorizzato da un utente può avvenire attraverso l'apposito pulsante "Elimina".

PRECONDIZIONI

1. Utente abilitato all'esecuzione della funzione elementare.
2. Sessione aperta sull'applicazione, utente posto sulla pagina "Ricerca Tipo Prenotazione".

Passo	Azione utente	Risposta del sistema
1	L'utente seleziona la riga della tabella, contenente tutti tipi di prenotazione memorizzati, che rappresenta il Tipo Prenotazione che si vuole eliminare.	Il sistema mostra la pagina di Dettaglio del Tipo Prenotazione.

2	L'utente richiede l'eliminazione del Tipo Prenotazione premendo il pulsante "Elimina".	Il sistema mostra il messaggio di avvenuta eliminazione e torna alla pagina precedente.

Le funzionalità presenti in mappa sono le seguenti:

Pulsanti/Link	**Descrizione**
Pulsante "Indietro"	Ritorna alla pagina precedente
Pulsante "Modifica"	Permette di modificare i parametri del Tipo Prenotazione.
Pulsante "Nuovo"	Permette di inserire i dati per un nuovo Tipo Prenotazione.
Pulsante "Stampa PDF"	Permette di stampare in formato PDF del Tipo Prenotazione.
Pulsante "Produci RTF"	Permette di riprodurre in formato RTF del Tipo Prenotazione.
Pulsante "Produci PDF"	Permette di riprodurre in formato PDF del Tipo Prenotazione.
Pulsante "Eimina"	Permette di eliminare l'eliminazione del Tipo Prenotazione selezionato.

5.7 Sottosistema di Sicurezza

L'infrastruttura di sicurezza offerta per l'identificazione ed autenticazione degli utenti consente l'accesso al sistema informativo solo a chi ne ha diritto.

I servizi si basano sul principio che ogni utente che accede alle risorse del sistema deve essere univocamente identificato attraverso un codice (userid), unico nel sistema e associato strettamente ad una persona fisica, che ne è responsabile dell'uso.

In particolare l'infrastruttura garantisce:

- controllo dell'accesso a dati e risorse su base personale;

- tracciamento delle operazioni effettuate dagli utenti per successive operazioni di auditing;
- attivazione dei meccanismi di protezione delle risorse a livello di singolo utente;
- attivazione dei meccanismi di auditing per il controllo del sistema e l'individuazione dei tentativi di attacco.

Ciascun utente avrà una propria userid personale e sarà responsabile dell'utilizzo che ne viene fatto. In tal modo ciascun accesso è tracciabile ed univoco. Le procedure di login degli utenti sono differenziate sulla base delle effettive esigenze operative. Inoltre saranno utilizzate tecnologie per le quali sia possibile adottare almeno un insieme minimo di contromisure quali:

- blocco automatico della sessione dopo un periodo di inattività (automatic lock-out);
- restrizione dell'accesso ai sistemi ed agli applicativi software solo in intervalli temporali predefinibili (business hour), e definizione dei permessi su base individuale;
- assegnazione dei privilegi d'accesso su base individuale o di gruppo;
- generazione, controllo puntuale e limitazione degli accessi agli archivi di log.

L'infrastruttura software di *security*, quindi, eroga i seguenti servizi:

- l'autenticazione all'intero sistema supportata dalle più moderne tecnologie di autenticazione digitale (Authentication Module);
- l'adeguata profilazione degli utenti (Authorization Module);
- la definizione delle politiche di autorizzazione per l'accesso ai servizi di competenza (Authorization Module);
- le regole di accesso ai solo dati di competenza (Authorization Module & Access Control Language).

Il sistema proposto consente, inoltre, di organizzare, in maniera del tutto flessibile, la struttura gerarchica di qualsiasi organizzazione nonché di associare persone fisiche ad una o più strutture della organizzazione definita.

I Moduli di Sicurezza proposti rappresentano uno strato tecnologico indipendente dallo strato applicativo relativo ai servizi applicativi ma fortemente integrati con essi. In pratica, è uno strato applicativo basato su tecnologie all'avanguardia in ambito J2EE. La soluzione proposta adotta e potenzia la tecnologia Spring Security Framework. Oltre all'aspetto tecnologico la soluzione si basa su un modello concettuale dei dati estremamente flessibile e configurabile i cui dettagli sono riportati di seguito.

Il modulo di sicurezza è da intendersi, quindi, come l'insieme di tecnologie, strutture dati e servizi di configurazione asserviti allo strato applicativo al fine di garantire le esigenze di:

- Autenticazione & Identificazione
- Autorizzazione
- Profilazione
- Sicurezza di Accesso ai Dati
- Auditing

Si riporta di seguito il modello logico offerto dal sistema di sicurezza:

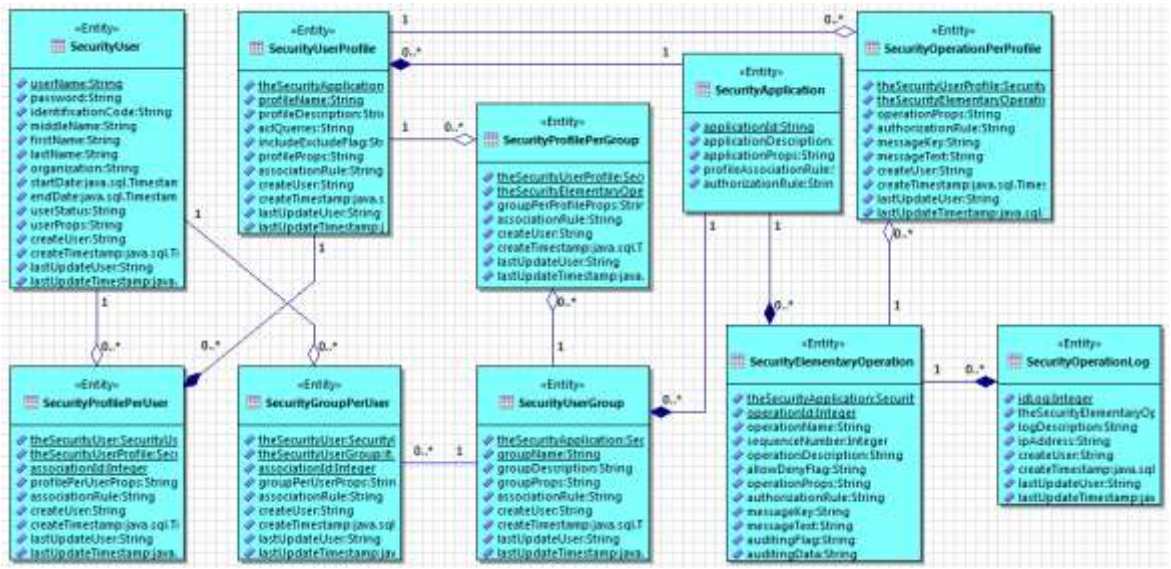


Figura 33. Modello logico del sotto-sistema di sicurezza

Il modello proposto comprende le seguenti entità chiave:

- **SecurityUser** – Anagrafica degli utenti;
- **SecurityApplication** – Il sotto-sistema di sicurezza proposto consente di gestire anche più di un'applicazione;
- **SecurityUserProfile** – Profili di sicurezza definiti nell'ambito di un'applicazione;
- **SecurityUserGroup** – Gruppi di utenti definiti nell'ambito di un'applicazione;
- **SecurityElementaryOperation** – Archivio di tutte le operazioni elementari definite per un'applicazione e sottoposte a controllo degli accessi;
- **SecurityProfilePerUser** – Associazione profili ad utenti
- **SecurityGroupPerUser** – Associazione utenti a gruppi
- **SecurityProfilePerGroup** – Associazione profili a gruppi
- **SecurityOperationPerProfile** – Associazione operazioni elementari ai profili
- **SecurityOperationLog** – Archivio di default per la memorizzazione dei log di accesso al sistema.

Di seguito il dettaglio dei servizi offerti da ogni componente del modulo.

Servizio di Autenticazione

Spring Security, pur fornendo un proprio sistema di autenticazione basato sull'immissione di utenza e password (anche con sistemi di cifratura), garantisce anche la completa integrazione con tutti i sistemi esterni di autenticazione, attraverso la semplice modifica di un file di configurazione XML.

Servizio di Profilazione

Il modello proposto consente di definire tutti i profili e tutti i gruppi coinvolti nell'ambito dell'organizzazione. Ogni utente censito può quindi essere associato ad uno o più profili e/o ad uno o più gruppi. Il profilo, oltre a completare le caratteristiche di un utente nell'ambito della organizzazione, rappresenta l'aspetto chiave per definire le autorizzazioni per l'accesso ai servizi.

Autorizzazione

Il sottosistema di sicurezza è concepito per censire/acquisire tutte le operazioni elementari definite per il sistema applicativo. Per Operazioni elementari si intendono tutte le azioni tipicamente rappresentate negli Use Case Diagram. Le operazioni elementari possono essere associate indifferentemente ad uno o più profili. Questa associazione definisce le operazioni abilitate ad un profilo. Gli utenti ereditano, tramite i loro profili diretti, o indirettamente, tramite i profili associati ai gruppi a cui appartengono, l'autorizzazione all'esecuzione di operazioni elementari/servizi messe a disposizione dalla piattaforma software. La struttura applicativa del sistema di sicurezza garantisce la completa flessibilità e dinamicità nella definizione di ***"chi può fare cosa"***. Si aggiunge, inoltre, che lo strato di sicurezza consente di definire regole dinamiche di autorizzazione all'accesso. Il sistema proposto prevede infatti la possibilità di abilitare l'accesso ai servizi anche su base temporale (accesso per fascia oraria) ereditando anche abilitazioni da profili diversi da quello di appartenenza. E' possibile anche definire regole di accesso basate sui dati scambiati nella transazione.

Controllo selettivo di accesso ai dati

Oltre alla possibilità di definire regole di accesso ai dati (attraverso il sistema di **A**ccess **C**ontrol **L**anguage), è anche possibile definire sotto-insiemi di informazioni sottoposte a controllo di accesso; a titolo di esempio è possibile:

- nascondere singole colonne di elenchi dati a chi non è autorizzato;
- nascondere o proteggere insieme di dati di una pagina di dettaglio a chi non è autorizzato.

Ancora, i link o pulsanti di selezione righe di un elenco possono essere nascosti o disabilitati se l'utente non è autorizzato a compiere l'operazione associata.

Sicurezza di Accesso ai Dati

Un'ulteriore aspetto migliorativo incluso nel modulo di authorization riguarda la possibilità di definire, con ampi margini di flessibilità, l'organizzazione gerarchica delle strutture organizzative coinvolte.

Si allega a scopo esplicativo il modello logico di riferimento:

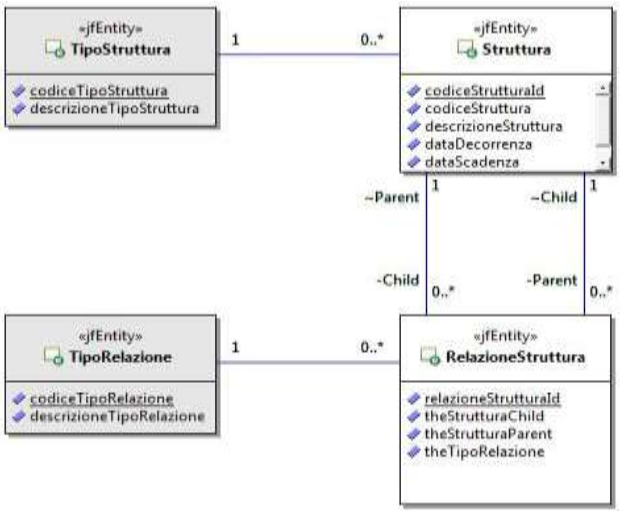


Figura 34. Modello logico delle Strutture Organizzative

Attraverso il modello logico riportato è possibile definire le strutture organizzative di interesse nell'ambito dell'organizzazione. E' possibile, quindi, definire le relazioni gerarchiche esistenti tra le strutture nonché associare gli utenti censiti alle strutture definite. Attraverso questa tipologia di configurazione è possibile definire il dominio dei dati abilitati ad una struttura organizzativa, attraverso le relazioni gerarchiche definite.

L' utente, associato ad una struttura organizzativa, eredita l'accesso al dominio dei dati della struttura, fermo restando i vincoli di accesso dettati dalle autorizzazioni relative al profilo di appartenenza dell'utente. L'associazione degli utenti alle strutture organizzative consente di gestire con estrema flessibilità eventuali subentri di personale, consentendo quindi ai nuovi utenti di ereditare le competenze del personale in uscita.

Sistema di Auditing

Il modulo prevede inoltre la possibilità di monitorate le operazioni sensibili e tracciare gli stessi attraverso un meccanismo di **logging** e rendere i dati presistenti in base dati. E' possibile definire modalità di tracciatura delle operazioni sensibili e configurare l'output di tracciatura a livello di singola operazione (operazione A tracciata su log, operazione B su base dati, etc…).

Servizi a supporto del Modulo di Autorizzazione

Il modulo è configurabile attraverso apposite interfacce web il cui accesso può essere messo a disposizione di personale addetto alla configurazione del sistema.

Tali interfacce consentono di censire:

- **Utenti** (con relativa funzionalità di associazione a gruppi e profili esistenti);
- **Profili** (con relativa funzionalità di associazione a utenti, gruppi e operazioni elementari esistenti);
- **Gruppi** (con relativa funzionalità di associazione a utenti e profili esistenti);
- **Operazioni Elementari** (con relativa funzionalità di associazione a profili esistenti);

Per una dimostrazione del sotto-sistema di sicurezza proposto è possibile consultare le funzionalità di Space alla voce di menu "Sicurezza".

5.8 Disegno e Realizzazione

I linguaggi per la modellazione e il design di applicazioni software object-oriented esistono ormai da molti anni, eppure ancora oggi è facile rendersi conto che molti programmatori, anziché usare un modello come base per l'implementazione delle loro applicazioni, si limitano a documentare il loro codice con UML a posteriori, rinunciando ai benefici che un design accurato a priori può portare, quali maggiore chiarezza, più facilità nell'identificare codice riutilizzabile, minor necessità di modificare l'architettura in corso d'opera se inizialmente ben progettata e così via. Nonostante questi aspetti positivi, infatti, la modellazione è spesso vista come un'attività, rispetto alla stesura di codice, vera e propria: molti ad esempio ritengono che un modello non avrà mai lo stesso potere espressivo di un linguaggio di programmazione, dato che altrimenti si tratterebbe solo di un diverso linguaggio di programmazione; e quindi, se il modello è meno espressivo del linguaggio, realizzare sistemi molto complessi basandosi su un "disegno" ad alcuni può sembrare un'utopia. Recentemente sono stati sviluppati vari sistemi per ridurre questo gap

tra modellazione e programmazione, in sostanza creando modelli in grado di "generare codice" e automatizzare le operazioni più noiose e ripetitive che la creazione di un'applicazione richiede, cercando quindi di far comprendere anche ai programmatori più incalliti i vantaggi dell'astrazione e della modellazione a priori.

È proprio questa la filosofia adottata per la realizzazione dell'applicazione Space ClassRoom. Dopo la fase di analisi e specifica dei requisiti si è passati alla fase di modellazione attraverso il disegno del diagramma delle classi, che rappresenterà il design dell'applicazione.

5.8.1 Strumenti e tecnologie utilizzate

1. **Hibernate**.

Hibernate è, come già detto nei capitoli in precedenza, una piattaforma middleware *open source* per lo sviluppo di applicazioni Java, attraverso l'appoggio al relativo framework, che fornisce un servizio di Object-relational mapping (ORM) ovvero gestisce la persistenza dei dati sul database attraverso la rappresentazione e il mantenimento su database relazionale di un sistema di oggetti Java. Nell'ambito dello sviluppo di applicazioni web, tale strato software si frappone tra il *livello logico di business* o di elaborazione e quello di persistenza dei dati sul database (*Data Access Layer*). Lo scopo principale di Hibernate è di fornire un *mapping* delle classi Java in tabelle di un database relazionale; sulla base di questo *mapping* Hibernate gestisce il salvataggio degli oggetti di tali classi su database. Si occupa, inoltre, al rovescio del reperimento degli oggetti dal database, producendo ed eseguendo automaticamente le query SQL necessarie al recupero delle informazioni e la successiva reistanziazione dell'oggetto precedentemente "ibernato", ovvero mappato su database. L'obiettivo di Hibernate è di esonerare lo sviluppatore dall'intero lavoro relativo alla persistenza dei dati. Hibernate si adatta al processo di sviluppo del programmatore, sia se si parte da zero sia se da un database già esistente. Hibernate genera le chiamate SQL e solleva lo sviluppatore dal lavoro di recupero manuale dei dati e dalla loro conversione,

mantenendo l'applicazione portabile in tutti i database SQL. Hibernate fornisce una persistenza trasparente per *Plain Old Java Object* (*POJO*); l'unica grossa richiesta per la persistenza di una classe è la presenza di un costruttore senza argomenti. In alcuni casi si richiede un'attenzione speciale per i metodi *equals()* e *hashcode().* Hibernate è tipicamente usato sia in applicazioni Swing che Java EE facenti uso di servlet o EJB di tipo *session beans.*

2. **HQL.**

Un ORM implementa solitamente il pattern Repository, il cui scopo è schermare l'utilizzatore dal reale supporto fisico delle istanze. Un repository è percepito dall'esterno come il *gestore della memorizzazione degli oggetti* e non è assolutamente necessario preoccuparsi di come questa memorizzazione sia realmente ottenuta.

Le operazioni base di un repository dal punto di vista logico sono le solite CRUD e tra queste l'operazione di Read è indubbiamente la più impegnativa, perché si ha la necessità di fornire all'utilizzatore una sintassi con cui specificare condizioni complesse sugli oggetti da recuperare. Mentre nel linguaggio SQL la sintassi per le query prevede solo un formato testuale, un repository permette di specificare condizioni direttamente sulle proprietà degli oggetti, solitamente tramite interfacce differenti. Nel caso di Hibernate ad esempio possiamo utilizzare il linguaggio *HQL (Hibernate Query Language).* Esso è un linguaggio di query object-oriented, simile a SQL. La differenza sta nel fatto che, invece di operare su tabelle e colonne, HQL opera su oggetti (persistenti) e le loro proprietà. Hibernate si occuperà di tradurre le query HQL in SQL e gestirà lui le azioni effettuate sul database. Con Hibernate è possibile, anche, effettuare query in SQL nativo, ma usare HQL è la scelta raccomandata così da poter sfruttare al meglio i meccanismi di hibernate di generazione sql e caching. In HQL la sintassi è simile a SQL in termini di parole chiave, che però non sono case-sensitive. Sono case-sensitive invece i nomi delle tabelle e delle colonne. Il principale vantaggio nell'utilizzo di HQL si realizza,

appunto, nel fatto che la query è espressa sul Domain Model e non sul modello fisico di memorizzazione.

Figura 35. Propagazione dell'interrogazione

Grazie a HQL si possono esprimere query come: *"Seleziona tutti i clienti che hanno la proprietà* Nome *pari a* 'Maria'», in questo modo si crea un livello di astrazione tra la logica applicativa ed il supporto di memorizzazione. Questa separazione è resa possibile dall'ORM, che si occupa di tradurre le richieste in un formato compatibile con il supporto di memorizzazione dei dati, trasformando poi i risultati dell'interrogazione in oggetti.

3. **SQL.**

SQL (Structured Query Language) è un linguaggio standardizzato per database basati sul modello relazionale (RDBMS) progettato per:

- creare e modificare schemi di database (DDL- *Data Definition Language*);
- inserire, modificare e gestire dati memorizzati (DML- *Data Manipulation Language*);
- interrogare i dati memorizzati (DQL- *Data Query Language*);
- creare e gestire strumenti di controllo ed accesso ai dati (DCL- *Data Control Language*).

Nonostante il nome, non si tratta, dunque, solo di un semplice linguaggio di interrogazione, ma alcuni suoi sottoinsiemi si occupano della creazione, della gestione e dell'amministrazione del database. SQL è un linguaggio per interrogare e gestire basi di dati mediante l'utilizzo di query. Con SQL si leggono, modificano, cancellano dati e si esercitano funzioni gestionali ed amministrative

sul sistema dei database. La maggior parte delle implementazioni dispongono di interfaccia alla riga di comando per l'esecuzione diretta di comandi, in alternativa alla sola interfaccia grafica GUI. Originariamente progettato come linguaggio di tipo dichiarativo, si è successivamente evoluto con l'introduzione di costrutti procedurali, istruzioni per il controllo di flusso, tipi di dati definiti dall'utente e varie altre estensioni del linguaggio. Alcune delle critiche più frequenti rivolte a SQL riguardano la mancanza di portabilità del codice fra vendor diversi, il modo inappropriato con cui sono trattati i dati mancanti (Null), e la semantica a volte inutilmente complicata. Essendo un linguaggio dichiarativo, SQL non richiede la stesura di sequenze di operazioni (come ad es. i Linguaggi imperativi), piuttosto di specificare le proprietà logiche delle informazioni ricercate.

4. **Data Access Object**.

   Il ***DAO*** (*Data Access Object*) è un pattern architetturale per la gestione della persistenza. È fondamentalmente una classe che rappresenta un'entità tabellare di un database. Principalmente usato in applicazioni J2EE come quelle EJB, serve a stratificare e isolare l'accesso ad una tabella dalla parte di *business logic* creando un maggiore livello di astrazione. Ciò si ottiene spostando la logica di accesso ai dati dai componenti di business ad una classe DAO rendendo gli EJB indipendenti dalla natura del dispositivo di persistenza. Quest' 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, quindi, 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

sulla base dati, nascondendo le problematiche JDBC. Il vantaggio relativo all'uso del DAO è il mantenimento di una rigida separazione tra le componenti di un'applicazione, le quali potrebbero essere il "Modello" e il "Controllo" in un'applicazione basata sul paradigma MVC.

5. **POJO.**

   I POJO (Plain Old Java Object) sono semplici oggetti Java, ossia dei JavaBeans, che incapsulano le informazioni in unità ben distinte e facilmente identificabili dal punto di vista funzionale. Non implementano alcuna interfaccia né hanno bisogno di un Application Server per poter funzionare, sono quindi semplici classi Java con le regole seguite dai JavaBean. Un EJB, utilizzato per la persistenza, portava con sé la problematica di essere un oggetto remoto e quindi la necessità di esportare non l'oggetto in sé, ma il proxy, che attraverso RMI (Remote Method Invocation) avrebbe effettuato le operazioni di lettura/scrittura sul database. In particolare, quest'ultima situazione ha visto il moltiplicarsi dell'uso di oggetti POJO incapsulati in un EJB, in modo da poter essere facilmente rappresentati e serializzati in fase di visualizzazione. Come per i Javabean, si creerà una classe contenente le variabili di istanza rappresentanti le entità in questione, i costruttori dei metodi getter e setter, ed eventualmente, saranno anche inserti altri metodi così come in una normale classe Java. È ovvio che bisognerà annotare la classe in modo che chi ne gestirà le operazioni di persistenza sappia come gestirla; la classe, infatti, dovrà essere annotata come Entity e già di per sé questo sarebbe sufficiente, in quanto, per default, tutte le variabili sono considerate come persistenti e quindi gestiti come tali. Da ciò si nota che non si è parlato della persistenza, essa, infatti, verrà automaticamente gestita dal layer della persistenza. In altri contesti, non è consentita questa funzionalità, che è, invece, una peculiarità di fondamentale importanza, in quanto permette a diversi team di lavorare in modo separato e di poter effettuare ognuno le proprie prove.

6. **EJB.**

La tecnologia J2EE mette a disposizione un ambiente adatto alla produzione di software server side "per componenti". Questi componenti non sono altro che gli EJB, Enterprise Java Bean. Un EJB è una classe Java che segue determinate regole imposte dalla specifica e, una volta programmato, viene posto in un application server e da esso verrà gestito secondo le regole definite. Si tratta dei componenti utilizzati per la logica di business di un'applicazione. In un contesto Java puro, generalmente, i client di questi oggetti sono le pagine JSP e le servlet, che si poggiano agli EJB per definire il contenuto dinamico da mostrare all'utente. In realtà, l'idea di questi componenti è nata per permettere a diverse tecnologie di usare lo stesso strato di logica, residente in un contesto distribuito. Quindi, saranno client degli EJB anche altre classi Java, altri EJB, un client scritto in un altro linguaggio di programmazione, una applet, o un sistema esterno. Questo modello di sviluppo consente a chi produce il componente server side di disinteressarsi di chi e di come accederà al servizio, permettendogli, quindi, uno sviluppo ed una manutenzione disaccoppiata da ogni logica di utilizzo esterna.

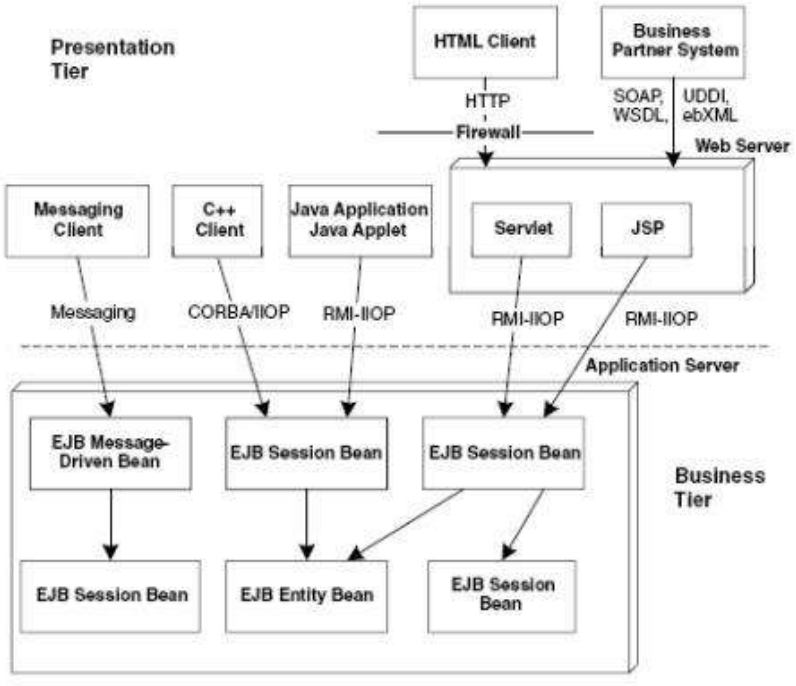


Figura 36. Uso degli EJB

Le diverse esigenze applicative sono state divise in tre aree principali:

- Session Bean: sono EJB dedicati alla logica di business, cioè oggetti adibiti ad effettuare operazioni logiche, ovvero tutte quelle operazioni che rientrano nella definizione di logica di business.
- Entity Bean: sono EJB dedicati alla rappresentazione dei dati. Attraverso questi oggetti è possibile leggere e scrivere dati da sorgenti persistenti.
- Message-Driven Bean: sono EJB dedicati allo sviluppo di applicazioni asincrone, cioè tutte quelle applicazioni che non necessitano una risposta immediata da parte del sistema a cui ci si connette. Si tratta di applicazioni che fanno uso di scambio messaggi, come i sistemi di invio mail o sms, e non è inusuale trovare sistemi di billing e reporting che ne fanno uso.

Dunque, tutte queste tecnologie hanno il principale obiettivo di svincolare totalmente le operazioni sui dati da quelle di logica applicativa, come pure le operazioni asincrone, basate sullo scambio di messaggi.

7. **Web Service.**

   Un Web Service, secondo la definizione data dal World Wide Web Consortium (W3C), è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una medesima rete ovvero in un contesto distribuito; tale caratteristica si ottiene associando all'applicazione un'interfaccia software, che espone all'esterno il servizio/i associato/i e utilizzando la quale altri sistemi possono interagire con l'applicazione stessa attivando le operazioni descritte nell'interfaccia tramite appositi "messaggi" di richiesta. Tali messaggi di richiesta sono inclusi in una "busta" (la più famosa è SOAP), formattati secondo lo standard XML, incapsulati e trasportati tramite i protocolli del Web (solitamente HTTP), da cui appunto il nome *web service*. Proprio grazie all'utilizzo di standard basati su XML, tramite un'architettura basata sui Web Service, chiamata *Service oriented Architecture- SOA*, applicazioni

software scritte in diversi linguaggi di programmazione e implementate su diverse piattaforme hardware possono quindi essere utilizzate, tramite le interfacce che queste "espongono" pubblicamente. Tali applicazioni possono essere utilizzate sia su reti aziendali sia su Internet. Alcuni dei vantaggi che è possibile ottenere con l'utilizzo dei Web Service sono i seguenti:

- permettono l'interoperabilità tra diverse applicazioni software su diverse piattaforme hardware;
- utilizzano standard e protocolli "open"; i protocolli ed il formato dei dati è, ove possibile, in formato testuale, cosa che li rende di più facile comprensione ed utilizzo da parte degli sviluppatori;
- mediante l'uso di HTTP per il trasporto dei messaggi i Web Service non necessitano, normalmente, che vengano effettuate modifiche alle regole di sicurezza utilizzate come filtro sui firewall;
- possono essere facilmente utilizzati, in combinazione l'uno con l'altro, indipendentemente da chi li fornisce e da dove vengono resi disponibili, per formare servizi "integrati" e complessi;
- consentono il riutilizzo di infrastrutture ed applicazioni già sviluppate e sono (relativamente) indipendenti da eventuali modifiche delle stesse.

Di contro vi sono i seguenti aspetti da considerare:

- attualmente non esistono standard consolidati per applicazioni critiche quali, ad esempio, le transazioni distribuite;
- le *performance* legate all'utilizzo dei Web Service possono essere minori di quelle riscontrabili utilizzando approcci alternativi di distributed computing quali Java RMI, CORBA, o DCOM;
- l'uso dell'HTTP permette ai Web Service di evitare le misure di sicurezza dei firewall.

La ragione principale per la creazione e l'utilizzo di Web Service è il "disaccoppiamento" che l'interfaccia standard esposta dal Web Service rende

possibile fra il sistema utente ed il Web Service stesso: modifiche ad una o all'altra delle applicazioni possono essere attuate in maniera "trasparente" all'interfaccia tra i due sistemi; tale flessibilità consente la creazione di sistemi software complessi costituiti da componenti svincolati l'uno dall'altro e consente una forte riusabilità di codice ed applicazioni già sviluppate. Web service hanno, inoltre, guadagnato consensi visto che, come protocollo di trasporto, possono utilizzare HTTP "over" TCP sulla porta 80; tale porta è, normalmente, una delle poche (se non l'unica) lasciata "aperta" dai sistemi firewall al traffico di entrata ed uscita dall'esterno verso i sistemi aziendali e ciò in quanto su tale porta transita il traffico HTTP dei web browser: ciò consente l'utilizzo dei Web Service senza modifiche sulle configurazioni di sicurezza dell'azienda. Un'ultima ragione che ha favorito l'adozione ed il proliferare dei Web Service è la mancanza, prima dello sviluppo di SOAP, di interfacce realmente funzionali per l'utilizzo di funzionalità distribuite in rete.

La pila protocollare dei Web Service è l'insieme dei protocolli di rete utilizzati per definire, localizzare, realizzare e far interagire tra di loro i Web Service; è principalmente composta di quattro aree:

a) *Trasporto del servizio*: responsabile per il trasporto dei messaggi tra le applicazioni in rete, include protocolli quali HTTP,SMTP, FTP.
b) *XML Messaging*: tutti i dati scambiati sono formattati mediante "tag" XML in modo che gli stessi possano essere utilizzati a entrambi i capi delle connessioni; il messaggio può essere codificato conformemente allo standard SOAP.
c) *Descrizione del servizio*: l'interfaccia pubblica di un Web Service viene descritta tramite WSDL (*Web Services Description Language*) un linguaggio basato su XML usato per la creazione di "documenti" descrittivi delle modalità di interfacciamento ed utilizzo del Web Service.

d) *Elencazione dei servizi*: la centralizzazione della descrizione e della localizzazione dei Web Service in un "registro" comune permette la ricerca ed il reperimento in maniera veloce dei Web Service disponibili in rete; a tale scopo viene attualmente utilizzato il protocollo UDDI.

8. **Ajax.**

AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive (Rich Internet Application). Lo sviluppo di applicazioni HTML con AJAX si basa su uno scambio di dati in background fra web browser e server, che consente l'aggiornamento dinamico di una pagina web senza esplicito ricaricamento da parte dell'utente. AJAX è *asincrono* nel senso che i dati extra sono richiesti al server e caricati in background senza interferire con il comportamento della pagina esistente. Normalmente le funzioni richiamate sono scritte con il linguaggio JavaScript. Tuttavia, e a dispetto del nome, l'uso di JavaScript e di XML non è obbligatorio, come non è necessario che le richieste di caricamento debbano essere necessariamente asincrone. AJAX è una tecnica multi-piattaforma utilizzabile cioè su molti sistemi operativi, architetture informatiche e browser web, ed esistono numerose implementazioni open source di librerie e framework. La tecnica Ajax utilizza una combinazione di:

- HTML (o XHTML) e CSS per il markup e lo stile;
- DOM (Document Object Model) manipolato attraverso un linguaggio ECMAScript come JavaScript o JScript per mostrare le informazioni ed interagirvi;
- l'oggetto XMLHttpRequest per l'interscambio asincrono dei dati tra il browser dell'utente e il web server. In alcuni framework Ajax e in certe situazioni, può essere usato un oggetto Iframe invece di XMLHttpRequest

per scambiare i dati con il server e, in altre implementazioni, tag <script> aggiunti dinamicamente (JSON);

- in genere viene usato XML come formato di scambio dei dati, anche se di fatto qualunque formato può essere utilizzato, incluso testo semplice, HTMLpreformattato, JSON e perfino EBML. Questi file sono solitamente generati dinamicamente da script lato server.

Ajax, dunque, non è una tecnologia individuale, piuttosto è un gruppo di tecnologie utilizzate insieme. Oggigiorno si è soliti identificare AJAX nello specifico oggetto XMLHttpRequest. Quest' oggetto permette di effettuare la richiesta di una risorsa (con HTTP) ad un server web in modo indipendente dal browser. Nella richiesta è possibile inviare informazioni, ove opportuno, sotto forma di variabili di tipo GET o di tipo POST in maniera simile all'invio dati di un form. La richiesta è asincrona, il che significa che non bisogna necessariamente attendere che sia stata ultimata per effettuare altre operazioni, stravolgendo sotto diversi punti di vista il flusso dati tipico di una pagina web. Generalmente, infatti, il flusso è racchiuso in due passaggi alla volta, richiesta dell'utente (link, form o refresh) e risposta da parte del server per poi passare, eventualmente, alla nuova richiesta da parte dell' utente.

Figura 37. Rappresentazione di una richiesta ad un server con latenza e risposta

Il terzo ed inevitabile incomodo di questo ciclo è l'attesa che trascorre tra una richiesta dell' utente e la risposta del server. Con l'aggiunta di AJAX si perde questa linearità e mentre l'utente è all'interno della stessa pagina le richieste sul server possono essere numerose e completamente indipendenti.

Nulla, infatti, vieta, teoricamente, di effettuare decine di richieste simultanee al server per operazioni differenti con o senza controllo da parte del navigatore.

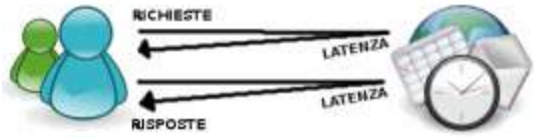


Figura 38. Rappresentazione di una richiesta ad un server con latenza e risposta

Ciò che resta del vecchio flusso, il tempo di attesa, passa spesso in secondo piano ed in molti tipi di interazione è praticamente impercettibile, ma si noti che questo tempo è anche uno dei maggiori problemi dell'utilizzo di AJAX, sia per gli sviluppatori sia per i navigatori. I primi potrebbero trovarsi in difficoltà qualora l'operazione asincrona dovesse attendere assolutamente una risposta al fine di completare una serie di operazioni più o meno sensibili, mentre i secondi potrebbero non avere idea di cosa stia accadendo alla pagina chiudendola ignari di aver richiesto un'informazione.

Infine, il tipo di risposta che l'oggetto si aspetta dopo una chiamata, non deve essere necessariamente di tipo XML, ma che può essere semplicemente testuale, in contro tendenza con l'acronimo stesso, ma non per questo inusuale.

9. Spring MVC Controllers

Spring MVC è un framework che permette la realizzazione di applicazioni web basate sul modello MVC sfruttando i punti di forza offerti dal framework Spring come l'inversion of control e l'aspect oriented programming. Esso si occupa di:

- mappare metodi e classi Java con determinati url;
- di gestire differenti tipologie di "viste" restituite al client;
- di realizzare applicazioni internazionalizzate;
- di gestire i cosiddetti temi, per personalizzare al meglio l'esperienza utente.

I controller rappresentano la vera anima del paradigma MVC. Essi sono il componente che viene invocato direttamente dai client che si occupano delle principali logiche di business e possono esistere anche senza la presenza di Model e di View. Il tipo di bean che si occupa del mapping tra url invocato dall'utente e metodo Java da invocare è l'handlerMapping. L'implementazione di default presente in Spring MVC è la classe RequestMappingHandlerMapping che permette di sfruttare le java annotations per identificare i metodi da mappare.

10. **JSP**.

JavaServer Pages, di solito indicato con la sigla JSP, è una tecnologia di programmazione Web in Java per lo sviluppo della *logica di presentazione* (tipicamente secondo il pattern MVC) di applicazioni Web, fornendo contenuti dinamici in formato HTML o XML. Si basa su un insieme di speciali tag, all'interno di una pagina HTML, con cui possono essere invocate funzioni, predefinite sotto forma di codice Java e funzioni Javascript. Inoltre, permette di creare librerie di nuovi tag che estendono l'insieme dei tag standard (JSP Custom Tag Library). Le librerie di tag JSP si possono considerare estensioni indipendenti dalla piattaforma delle funzionalità di un Web server. Nel contesto della piattaforma Java, la tecnologia JSP è correlata con quella delle servlet: all'atto della prima invocazione, le pagine JSP vengono infatti tradotte automaticamente da un compilatore JSP in servlet. Una pagina JSP può, quindi, essere vista come una rappresentazione ad alto livello di una servlet. Per via di questa dipendenza concettuale, anche l'uso della tecnologia JSP richiede la presenza, sul Web server, di un *servlet container*, oltre che di un server specifico JSP detto *motore JSP*, che include il compilatore JSP. In genere, il servlet container e il motore JSP sono integrati in un unico prodotto, ad esempio Tomcat. JSP è una tecnologia alternativa rispetto a numerosi altri approcci alla generazione di pagine Web dinamiche, come PHP, o ASP o la più tradizionale CGI. Differisce da queste

tecnologie non tanto per il tipo di contenuti dinamici che si possono produrre, quanto per l'architettura interna del software che costituisce l'applicazione Web, e, di conseguenza, sui tempi di sviluppo, la portabilità, la modificabilità, le prestazioni, e altri aspetti di qualità del software. Sun Microsystem raccomanda di utilizzare il pattern Model-View-Controller con le pagine JSP in modo da dividere il livello di presentazione da quello dell' elaborazione della request e dalla memorizzazione dei dati. Le normali servlet o delle pagine JSP dedicate vengono utilizzate per processare i dati. Dopo che l'elaborazione è terminata, il controllo passa ad una pagina JSP che serve solo a visualizzare l'output. Quest'ultima pagina JSP dovrebbe contenere solo HTML, XML e action e tag JSP, la pagina dovrebbe far uso dei JavaBeans per ottenere i dati. In altri termini, nello sviluppo di un'applicazione web la convenzione vuole che nelle JSP ci sia meno codice Java possibile e quello presente vada a richiamare codice Java nativo implementato in classi separate apposite dette JavaBeans. Questa separazione consente, infatti, un facile riuso di codice dei JavaBeans una volta richiamati in un qualsiasi punto dell'applicazione web.

5.8.2 Diagramma delle classi

Uno dei più importanti documenti prodotti nella specifica del software è il diagramma UML delle classi. Nato a metà degli anni ’90 come unificazione di precedenti linguaggi per l’analisi orientata agli oggetti, probabilmente è il più importante linguaggio per la modellazione attualmente usato nell’ingegneria del software. Il diagramma UML delle classi offre la possibilità di modellare molti aspetti che sono potenzialmente di interesse per varie fasi dello sviluppo del software. In questo capitolo ci concentriamo sull’uso di questo tipo di diagramma nell’implementazione. In termini generali, consentono di descrivere *tipi di entità*, con le loro caratteristiche e le eventuali relazioni fra questi tipi. Il principali strumento concettuale utilizzato è quello di classe del paradigma object-oriented. Il Class Diagram del linguaggio UML consiste, quindi, di:

- Classi. Le classi UML rappresentano insiemi di oggetti con aspetti comuni. Una classe viene rappresentata graficamente mediante un rettangolo diviso in tre parti. La parte superiore (l'unica obbligatoria) contiene il nome della classe, che deve essere unico nel diagramma. Le altre parti, non obbligatorie, contengono, rispettivamente gli attributi e le operazioni della classe.
- Attributi. Gli attributi rappresentano proprietà locale degli oggetti. Saranno ignorati i qualificatori degli attributi, quelli che li distinguono fra pubblici, protetti e privati. I nomi degli attributi sono unici all'interno di una classe, ma due classi differenti possono avere attributi con lo stesso nome.
- Tipi degli attributi. Ogni attributo ha un tipo associato ad esso. Un tipo è una collezione di valori, a cui non viene associato un ciclo di vita (come si fa con gli oggetti). Come tipi vengono solitamente utilizzate entità che hanno una diretta implementazione nei linguaggi di programmazione, ad es. intero, reale, stringa.
- Associazioni. Un'associazione è una relazione fra una o più classi ed ha un nome unico nel diagramma. Le associazioni possono avere un'arità arbitraria, purché sia maggiore di uno.
- Generalizzazioni. Una generalizzazione, fra una superclasse e una sottoclasse, esprime che ogni istanza di quest'ultima è anche un'istanza della prima. Di conseguenza, le istanze della sottoclasse ereditano le proprietà della superclasse, e ne possono avere di specifiche.
- Specializzazioni. Nell'ambito delle generalizzazioni, le sottoclassi possono specializzare proprietà ereditate dalla superclasse.
- Vincoli. I vincoli rappresentano ulteriori informazioni che non possono essere rappresentate in maniera diretta mediante la sintassi grafica del diagramma UML delle classi. Tipicamente, per esprimere queste informazioni viene utilizzato il linguaggio OCL, acronimo di Object Constraint Language. Un esempio di vincolo è l'identificatore della classe.

Il primo passo effettuato, per la realizzazione dell'applicazione web, Space Classroom, è stato il disegno di quello che è il diagramma delle classi dell'applicazione. Nell'analisi orientata agli oggetti, la modellazione di dominio ha lo scopo di descrivere le informazioni della realtà d'interesse secondo una rappresentazione concettuale e a oggetti. In questo contesto, la "realtà di interesse" viene chiamata "dominio del problema", dunque, progettazione concettuale e modellazione di dominio hanno scopi simili e sono attività svolte con strumenti e metodi simili. Una possibile osservazione è che la modellazione di dominio è un modo particolare di fare analisi orientata agli oggetti, esistono anche altri modi di fare analisi, ma non tutti sono interessati alle informazioni della realtà d'interesse. Nell'ingegneria del software i diagrammi si chiamano modelli e i formalismi per esprimere i modelli si chiamano linguaggi, il linguaggio UML. Il risultato della modellazione di dominio è un modello di dominio espresso mediante un diagramma delle classi di UML e usato dal punto di vista concettuale.

Tale diagramma si è potuto costruire grazie all'analisi dei requisiti fatta in precedenza e riportata nei paragrafi antecedenti. Per l'attuazione del class diagram è stato, invece, semplicemente utilizzato un tool di Eclipse, si sta parlando, nello specifico dell'utilizzo della View Prospective Modeling.

È stato di fondamentale importanza, in questa fase, avere una giusta e completa visione di quello che si voleva realizzare, in quanto la creazione del database sarebbe avvenuta proprio sulla base del diagramma disegnato. Ciò è vero, poiché, il class diagram costituisce l'input del generatore di codice SkyGen, che a sua volta restituisce come output non solo l'applicazione implementata, ma anche i file di configurazione del database, sia per quanto riguarda l'applicazione stessa che il sotto-sistema di sicurezza. Una modifica relativa al database, successiva alla generazione sarà, comunque, sempre possibile, ma richiederà una modifica al diagramma delle classi con una conseguente rigenerazione. In realtà, la modifica potrebbe anche essere apportata mettendo direttamente mano al codice, ma questo sarebbe un lavoro troppo lungo e oneroso.

Effettuata tale premessa, si prenda visione del diagramma delle classi realizzato:

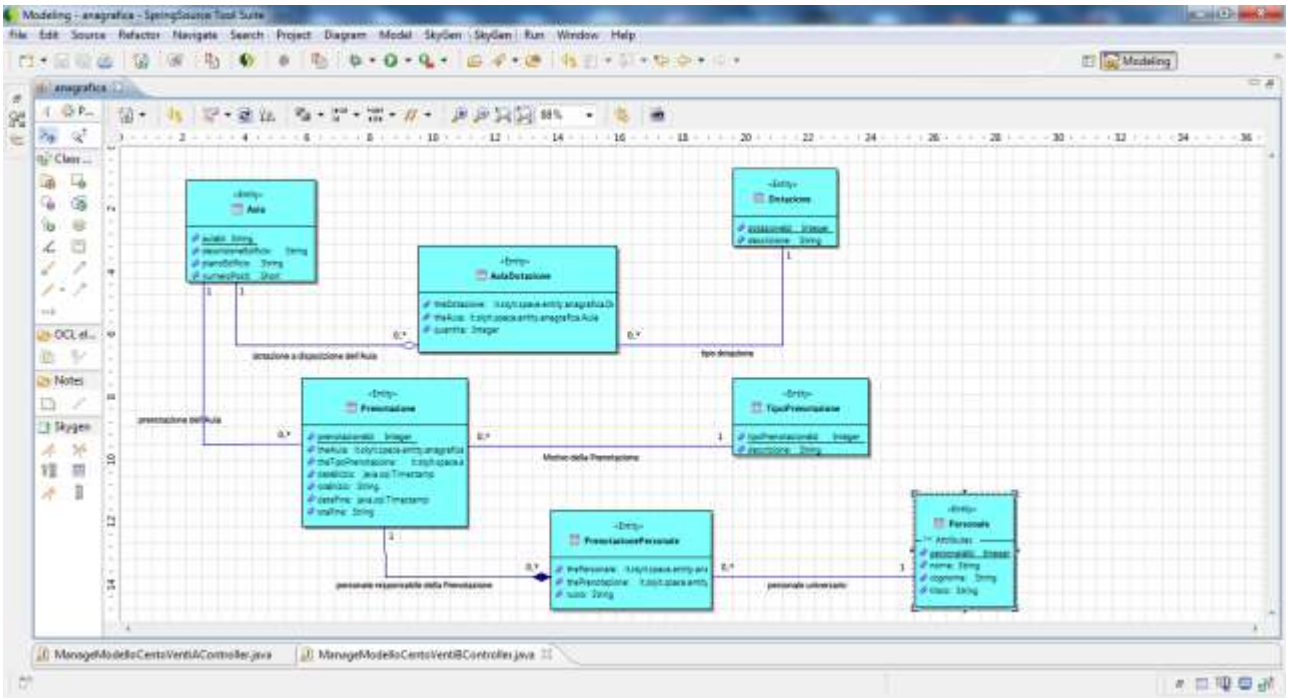


Figura 39. Diagramma UML dell'applicazione Space

Dall'immagine si vede che le entità utilizzate e i loro relativi attributi sono:

- Aula
    - AulaID, DescrizioneEdificio, PianoEdificio, NumeroPosti
- Dotazione
    - DotazioneId, Descrizione
- Prenotazione
    - PrenotazioneId, theAula, theTipoPrenotazione, DateInizio, DateFine, OraInizio, OraFine
- Tipo Prenotazione
    - TipoPrenotazioneId, Descrizione
- Personale
    - PersonaleId, Nome, Cognome, Titolo
- Aula Dotazione
    - theDotazione, theAula, Quantità

- Prenotazione Personale
    - thePersonale, thePrenotazione, Ruolo

È importante notare come tutte le classi, o entità, siano dotate di soli attributi, mancano, infatti, le operazioni. Com'è noto l'unica cosa obbligatoria nella stesura del diagramma delle classi è il nome di ogni classe, gli attributi e le operazioni invece sono solo facoltativi. In realtà, l'aspetto più importante è proprio questo, in quanto una volta effettuata la generazione di codice attraverso lo strumento SkyGen, ogni file .java appartenente al package it.skyit.space.entity conterà i metodi getter e setter per ogni attributo dell'entità considerata.

Un ulteriore aspetto da analizzare del diagramma delle classi è costituito dalle relazioni. Esse possono essere così elencate:

- Associazione Aula – AulaDotazione: è un particolare tipo di associazione chiamato aggregazione, il cui significato è "insieme di". È stato pensato questo tipo di associazione in quanto in un' aula è possibile trovare una o più dotazioni.
- Associazione AulaDotazione – Dotazione: è una associazione molti ad uno (0..* - 1), in quanto essa permette di identificare il tipo delle dotazioni presenti in un' aula. Si ricordi che l'entità Dotazione raccoglie tutti i tipi possibili di dotazione in un'aula.
- Associazione Aula – Prenotazione: è una associazione uno a molti (1 -0..*) . Una stessa aula può essere soggetta ad una o più prenotazioni, naturalmente non sarà possibile la prenotazione di un'aula in un orario in cui essa è già prenotata.
- Associazione Prenotazione – TipoPrenotazione: è una associazione molti ad uno (0..* - 1), in quanto ogni prenotazione può essere di uno e un sol tipo.
- Associazione Prenotazione – PrenotazionePersonale: è un particolare tipo di associazione chiamato composizione, il cui significato è "è composto da". È una relazione più forte dell'aggregazione, l'oggetto parte appartiene ad un solo tutto e le parti hanno lo stesso ciclo di vita dell'insieme. All'atto della distruzione dell'oggetto principale si ha la propagazione della distruzione agli oggetti parte.

- Associazione PrenotazionePersonale – Personale: è una associazione molti ad uno (0..* - 1), in quanto più persone appartenenti al personale possono essere correlate ad una prenotazione.

Una volta creato il diagramma delle classi, esso sarà dato in input al generatore. Quest'ultimo restituirà una serie di classi Java completamente implementate, che realizzano i vincoli, le relazioni e le associazioni descritte nel modello di partenza.

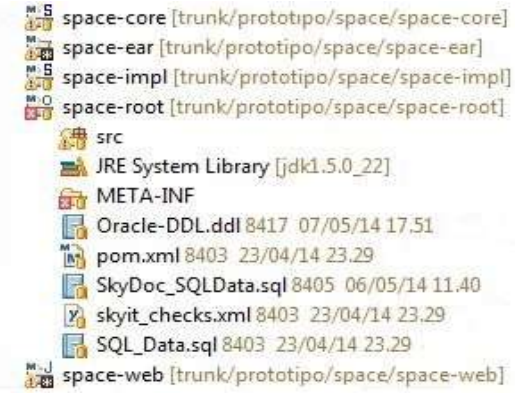
5.8.3 Realizzazione e Analisi del Database

Il diagramma delle classi appena descritto nelle sue parti più tecniche, rappresenta anche, come già accennato il database dell'applicazione, infatti le entità che lo compongono saranno trasformate in tabella grazie al generatore SkyGen.

Dando in input al generatore il class diagram, ciò che verrà dato in output è l'effettiva implementazione dell'applicazione. Essa risulterà essere funzionante fin da subito, basterà, infatti, avviare il server, che nel caso specifico è Tomcat, ed immettere all'apertura del browser il seguente indirizzo: http://localhost:8080/space/home.

Passando ad esaminare il codice, l'output di SkyGen è il seguente.

Sono i quattro progetti che compongono l'applicazione, ognuno di essi risulta ricoprire uno specifico ruolo. In particolare, aprendo space-root si ha:



Tre sono i file che attirano maggiormente l'attenzione, ovvero:

- Oracle-DDL.ddl
- SkyDoc_SQLData.sql
- SQL_Data.sql

Si consideri anzitutto il file Oracle-DDL.ddl, esso è scritto, appunto, utilizzando il linguaggio Data Definition Language (DDL). Questo linguaggio permette di creare, modificare o eliminare gli oggetti in un database ovvero di agire sullo schema del database. Sono i comandi DDL a definire la struttura del database e quindi l'organizzazione logica dei dati in esso contenuti, ma non forniscono gli strumenti per modificare i valori assunti dai dati o per interrogare i dati stessi per i quali, invece, si usano rispettivamente il Data Manipulation Language e il Data Query Language. È utilizzato sia in fase di progettazione, che in fase di ristrutturazione del database. Per agire sulla struttura del database l'utente deve avere i permessi necessari, assegnati tramite il Data Control Language (DCL). Il linguaggio DDL compone una parte del linguaggio SQL. Si considerino adesso alcune delle operazioni possibili e presenti nel file Oracle-DDL.ddl:

- Create Domain. Oltre ai tipi come integer, char, float, è possibile crearne altri. La creazione è simile ad una ridenominazione di un tipo fondamentale visto tra quelli sopra o di un tipo creato ex novo, ereditandone tutte le caratteristiche. La sintassi del comando Create Domain è la seguente:

  ```
  CREATE DOMAIN nome dominio AS tipo [ ValoreImpostato ] [ Vincolo ]
  { DefSchema }
  ```

  Si crea un tipo di nome " nome_dominio " partendo da un precedente " tipo ", impostando un opzionale valore di default ed un insieme di vincoli. Questa operazione permette di definire una ed una sola volta tutte le caratteristiche che possono essere associate ad un attributo, quando questo è utilizzato in più tabelle, evitando così ridondanze.

- Drop Domain. Elimina un dominio definito dall'utente. La sintassi del comando Drop Domain è:

  ```
  DROP DOMAIN nome_dominio [CASCADE|RESTRICT]
  ```

Se si specifica CASCADE, tutte le colonne delle tabelle, che appartengono a tale dominio verranno cancellate con esso. Se si specifica RESTRICT, che è il valore predefinito, questa operazione non verrà eseguita.

- Create database. Il comando create database serve a creare un nuovo database, che potrà contenere tabelle, viste, stored procedure, trigger o altri tipi di oggetti. La sintassi di tal comando è:

```
CREATE {DATABASE | SCHEMA} db name [create specification [,
create specification] ...]
create specification:
[DEFAULT] CHARACTER SET charset name | [DEFAULT] COLLATE
collation name
```

Se nell'esecuzione del comando si specifica [IF NOT EXISTS] si crea il database solo se non esiste, in caso contrario non verrà restituito alcun errore. La stringa create_specification permette di inserire delle opzioni nella creazione del database. Tramite CHARACTER SET si inserisce il set di caratteri supportato nel database. Tramite COLLATE si possono definire i dati di default del database.

- Alter database. Il comando alter database serve a modificare un database esistente. Non esiste nello standard SQL, ma la maggior parte dei Dbms lo implementa. La sintassi del comando è la seguente:

```
ALTER {DATABASE | SCHEMA} [db name]
[DEFAULT] CHARACTER SET charset name
| [DEFAULT] COLLATE collation name
```

- Drop database. Il comando drop database serve a cancellare un database. Tale comando non esiste nello standard SQL, ma tutti i Dbms lo implementano. La Sintassi del comando drop database è :

```
DROP DATABASE nome database
```

- Create table. Il comando create table ha la funzione di creare una nuova tabella (o tavola). Il nome della tabella può essere scritto indifferentemente in maiuscolo o in minuscolo, in ogni caso, però, è necessario che rispetti le seguenti regole:
    - Può essere formato da lettere e numeri, ma il primo carattere deve sempre essere una lettera;
    - Non può superare i 30 caratteri di lunghezza;

- Non può avere lo stesso nome di una tabella o vista già esistente sullo stesso utente di database.

Oltre a definire gli attributi di una tabella è possibile definire dei vincoli. La tabella può essere creata vuota oppure può essere creata e riempita di dati. Nel secondo caso la struttura della tabella è definita implicitamente dal numero di colonne estratte dalla select, dal tipo di dato di ciascuna colonna e dai nomi delle rispettive colonne estratte dalla select. Nel caso in cui la tabella venga popolata in fase di creazione, la transazione viene automaticamente conclusa da un comando di COMMIT. Per implementare i Vincoli di integrità con l'SQL esistono delle parole riservate. Per quanto concerne i Vincoli Intrarelazional, esse sono:

- Not Null: Il vincolo not null indica che il valore nullo non è ammesso come valore dell'attributo. In tal caso l'attributo deve sempre essere specificato tipicamente in fase di inserimento. Se all'attributo è, però, associato un valore di default diverso dal valore nullo, allora diventa possibile effettuare l'inserimento anche senza fornire un valore dell'attributo, in quanto all'attributo viene automaticamente assegnato il valore di default.
- Unique: Un vincolo unique si applica ad un attributo o ad un insieme di attributi di una tabella e impone che i valori dell'attributo (o le ennuple di valori sull'insieme di attributi) siano una (super) chiave, cioè righe differenti della tabella non possano comparire su diverse righe senza violare il vincolo, in quanto si assume che i valori nulli siano tutti diversi tra loro.
- Primary Key: Nella definizione di una tabella è necessario specificare per ogni relazione la chiave primaria, il più importante tra gli identificatori della relazione. SQL permette così di specificare il vincolo primary key una sola volta per ogni tabella. Il vincolo primary key può essere definito direttamente su un singolo attributo, oppure essere definito elencando più attributi che costituiscono l'identificatore.

Per la gestione dei Vincoli Interrelazionali, invece, si utilizza la foreign key. Questo vincolo crea un legame tra i valori di un attributo della tabella corrente e i valori di un attributo di un'altra tabella, che è in relazione alla tabella corrente stessa. Il vincolo impone che per ogni riga della tabella il valore dell'attributo specificato, se diverso dal valore nullo, sia presente nelle righe della tabella esterna tra i valori corrispondenti dell'attributo. La sintassi del comando *create table* è:

1. *Create table semplice:*

```
CREATE TABLE nome_tabella(nome_colonna1 tipo di dato constraint
(opzionale),
nome_colonna2 tipo di dato constraint (opzionale),
nome_colonna3 tipo di dato constraint (opzionale), ...
nome_colonnaN tipo di dato constraint (opzionale));
```

2. *Create table mediante select:*

```
CREATE TABLE nome_tabella AS SELECT...;
```

Di seguito è riportato un esempio di tale operazione in relazione all'applicazione realizzata:

```
CREATE TABLE AULA (
   AULA_ID                                VARCHAR2(20 CHAR) NOT NULL
  ,DESCRIZIONE_EDIFICIO                   VARCHAR2(40 CHAR) NOT NULL
  ,PIANO_EDIFICIO                         VARCHAR2(10 CHAR) NOT NULL
  ,NUMERO_POSTI                           NUMBER(4) NULL
);

CREATE TABLE AULA_DOTAZIONE (
   DOTAZIONE_ID                           INTEGER NOT NULL
  ,AULA_ID                                VARCHAR2(20 CHAR) NOT NULL
  ,QUANTITA                               NUMBER(8) NULL
);
CREATE TABLE DOTAZIONE (
   DOTAZIONE_ID                           INTEGER NOT NULL
  ,DESCRIZIONE                            VARCHAR2(40 CHAR) NOT NULL
);
CREATE TABLE PERSONALE (
   PERSONALE_ID                           INTEGER NOT NULL
  ,NOME                                   VARCHAR2(30 CHAR) NOT NULL
  ,COGNOME                                VARCHAR2(30 CHAR) NOT NULL
  ,TITOLO                                 VARCHAR2(30 CHAR) NOT NULL
);
CREATE TABLE PRENOTAZIONE (
   PRENOTAZIONE_ID                        INTEGER NOT NULL
  ,AULA_ID                                VARCHAR2(20 CHAR) NOT NULL
```

```
  ,TIPO_PRENOTAZIONE_ID                    INTEGER NOT NULL
  ,DATE_INIZIO                             DATE NULL
  ,ORA_INIZIO                              VARCHAR2(9 CHAR) NULL
  ,DATE_FINE                               DATE NULL
  ,ORA_FINE                                VARCHAR2(9 CHAR) NULL
);
CREATE TABLE PRENOTAZIONE_PERSONALE (
   PERSONALE_ID                            INTEGER NOT NULL
  ,PRENOTAZIONE_ID                         INTEGER NOT NULL
  ,RUOLO                                   VARCHAR2(30 CHAR) NOT NULL
);
CREATE TABLE TIPO_PRENOTAZIONE (
   TIPO_PRENOTAZIONE_ID                    INTEGER NOT NULL
  ,DESCRIZIONE                             VARCHAR2(40 CHAR) NOT NULL
);
```

In questo caso, sono state create, dal generatore SkyGen, le tabelle del database rappresentanti le entità Aula, AulaDotazione, Dotazione, Personale, Prenotazione, PrenotazionePersonale, TipoPrenotazione.

- Alter table. Il comando alter table ha la funzione di modificare la struttura della tabella. L'operatore ADD consente di inserire una nuova colonna su una tabella esistente oppure di aggiungere delle constraint alle colonne della tabella. L'operatore MODIFY consente di cambiare il tipo di dato e/o la constraint propri di ogni colonna di una tabella. L'operatore DROP consente di eliminare la constraint dalla colonna. La sintassi del comando alter table è la seguente :
  - *Operatore add*:
    - Inserimento di una nuova colonna:
      ```
      ALTER TABLE nome_tabella
      ADD nome_colonna_nuova tipo_di_dato constraint;
      ```
    - Aggiunta di una chiave primaria (primary key):
      ```
      ALTER TABLE nome_tabella
      ADD CONSTRAINT nome_tabella_pk
      PRIMARY KEY(nome_colonna che funge da chiave primaria);
      ```
    - Aggiunta di un indice:
      ```
      ALTER TABLE nome_tabella
      ADD CONSTRAINT nome_tabella_pk
      ADD INDEX nome_indice (nome_colonna)
      ```
    - Aggiunta di una chiave esterna (foreign key) ereditata da un'altra tabella:
      ```
      ALTER TABLE nome_tabella_figlia
      ADD CONSTRAINT nome_tabella_figlia_fk
      FOREIGN KEY (nome_colonna che funge da chiave esterna
      sulla tabella figlia)
      ```

```
REFERENCES nome_tabella_padre (nome_colonna che funge
da chiave primaria sulla tabella padre);
```

– *Operatore modify*:

o Modifica del tipo di dato di una colonna:

```
ALTER TABLE nome_tabella
MODIFY nome_colonna tipo di dato nuovo; (il tipo di
dato nuovo deve essere compatibile)
```

o Modifica della constraint di una colonna:

```
ALTER TABLE nome_tabella
MODIFY nome_colonna  constraint nuova;
```

– *Operatore drop*:

o Eliminazione della chiave primaria:

```
ALTER TABLE nome_tabella
DROP PRIMARY KEY;
```

o Eliminazione della chiave esterna:

```
ALTER TABLE nome_tabella_figlia
DROP CONSTRAINT nome tabella figlia fk;
```

Nel file Oracle-DDL.ddl, ad esempio, si ritrova:

```
ALTER TABLE AULA_DOTAZIONE
   ADD  ( CONSTRAINT RELATION_3822019980 FOREIGN KEY (
   AULA_ID
   ) REFERENCES AULA (
   AULA_ID
   )
   ON DELETE CASCADE
);

ALTER TABLE AULA_DOTAZIONE
   ADD  ( CONSTRAINT RELATION_551855769 FOREIGN KEY (
   DOTAZIONE_ID
   ) REFERENCES DOTAZIONE (
   DOTAZIONE_ID
   )
   ON DELETE CASCADE
);

ALTER TABLE PRENOTAZIONE
   ADD  ( CONSTRAINT RELATION_268744941 FOREIGN KEY (
   AULA_ID
   ) REFERENCES AULA (
   AULA_ID
   )
   ON DELETE CASCADE
);
ALTER TABLE PRENOTAZIONE
   ADD  ( CONSTRAINT RELATION_4263442914 FOREIGN KEY (
   TIPO_PRENOTAZIONE_ID
   ) REFERENCES TIPO_PRENOTAZIONE (
   TIPO_PRENOTAZIONE_ID
```

```
    )
    ON DELETE CASCADE
);

ALTER TABLE PRENOTAZIONE_PERSONALE
    ADD  ( CONSTRAINT RELATION_1406111425 FOREIGN KEY (
    PERSONALE_ID
    ) REFERENCES PERSONALE (
    PERSONALE_ID
    )
    ON DELETE CASCADE
);
ALTER TABLE PRENOTAZIONE_PERSONALE
    ADD  ( CONSTRAINT RELATION_1087536345 FOREIGN KEY (
    PRENOTAZIONE_ID
    ) REFERENCES PRENOTAZIONE (
    PRENOTAZIONE_ID
    )
    ON DELETE CASCADE
);

ALTER TABLE SEC_EL_OP
    ADD  ( CONSTRAINT RELATION_1195491240 FOREIGN KEY (
    APP_ID
    ) REFERENCES SEC_APP (
    APP_ID
    )
    ON DELETE CASCADE
);

ALTER TABLE SEC_OP_LOG
    ADD  ( CONSTRAINT RELATION_747238161 FOREIGN KEY (
    APP_ID
   ,OP_ID
    ) REFERENCES SEC_EL_OP (
    APP_ID
   ,OP_ID
    )
    ON DELETE CASCADE
);

ALTER TABLE SEC_OP_X_PROF
    ADD  ( CONSTRAINT RELATION_2679956495 FOREIGN KEY (
    OP_APP_ID
   ,OP_ID
    ) REFERENCES SEC_EL_OP (
    APP_ID
   ,OP_ID
    )
    ON DELETE CASCADE
);

ALTER TABLE SEC_OP_X_PROF
    ADD  ( CONSTRAINT RELATION_4153345778 FOREIGN KEY (
    PROF_APP_ID
   ,PROF_NAME
    ) REFERENCES SEC_USER_PROF (
```

```
   APP_ID
  ,PROF_NAME
   )
   ON DELETE CASCADE
);

ALTER TABLE SEC_PROF_X_USER
   ADD  ( CONSTRAINT RELATION_674952014 FOREIGN KEY (
   USER_NAME
   ) REFERENCES SEC_USER (
   USER_NAME
   )
   ON DELETE CASCADE
);

ALTER TABLE SEC_PROF_X_USER
   ADD  ( CONSTRAINT RELATION_1039345821 FOREIGN KEY (
   APP_ID
  ,PROF_NAME
   ) REFERENCES SEC_USER_PROF (
   APP_ID
  ,PROF_NAME
   )
   ON DELETE CASCADE
);

ALTER TABLE SEC_USER_PROF
   ADD  ( CONSTRAINT RELATION_1504684564 FOREIGN KEY (
   APP_ID
   ) REFERENCES SEC_APP (
   APP_ID
   )
   ON DELETE CASCADE
);
```

- Drop table. Il comando drop table consente di distruggere una tabella, eliminandola fisicamente dal database. Come tutti i comandi DDL è un'operazione irreversibile, e provoca la perdita di tutti i dati contenuti nella tabella. La sintassi del comando è:

  ```
  DROP TABLE nome_tabella [CASCADE|RESTRICT]
  ```

  Ad esempio:

  ```
  DROP TABLE AULA CASCADE CONSTRAINTS;
  DROP TABLE AULA_DOTAZIONE CASCADE CONSTRAINTS;
  DROP TABLE DOTAZIONE CASCADE CONSTRAINTS;
  DROP TABLE PERSONALE CASCADE CONSTRAINTS;
  DROP TABLE PRENOTAZIONE CASCADE CONSTRAINTS;
  DROP TABLE PRENOTAZIONE_PERSONALE CASCADE CONSTRAINTS;
  DROP TABLE TIPO_PRENOTAZIONE CASCADE CONSTRAINTS;
  ```

- Create Index. È una scorciatoia per evitare ALTER TABLE complessi, non fa altro che aggiungere un indice a una tabella. La sintassi del comando Create Index è:

```
CREATE [UNIQUE] INDEX nome_indice
ON nome_tabella (nome_colonna [ASC|DESC])
```

Se si specifica UNIQUE, l'indice è unico, cioè i valori al suo interno non possono essere duplicati. Se si specifica ASC i valori all'interno dell'indice saranno in ordine ascendente, se si specifica DESC saranno in ordine discendente; il valore predefinito è ASC.

Per esempio, in Oracle-DDL.ddl si ha:

```
CREATE INDEX XFK1_AULA_DOTAZIONE ON AULA_DOTAZIONE
(
      AULA_ID                                   ASC
);

CREATE INDEX XFK2_AULA_DOTAZIONE ON AULA_DOTAZIONE
(
      DOTAZIONE_ID                              ASC
);

CREATE INDEX XFK1_PRENOTAZIONE_PERSONALE ON PRENOTAZIONE_PERSONALE
(
      PERSONALE_ID                              ASC
);

CREATE INDEX XFK2_PRENOTAZIONE_PERSONALE ON PRENOTAZIONE_PERSONALE
(
      PRENOTAZIONE_ID                           ASC
);

CREATE INDEX XFK1_PRENOTAZIONE ON PRENOTAZIONE
(
      AULA_ID                                   ASC
);

CREATE INDEX XFK2_PRENOTAZIONE ON PRENOTAZIONE
(
      TIPO_PRENOTAZIONE_ID                      ASC
);
```

- Drop Index. È una scorciatoia per evitare ALTER TABLE complessi. Elimina un indice da una tabella. La sintassi del comando è:

```
DROP INDEX nome_indice
ON nome_tabella.
```

In SQL, si definisce procedura un blocco di codice dotato di un nome attraverso cui viene salvato e può essere richiamata da altri blocchi SQL, da altre procedure e da altre funzioni. La sintassi per la definizione di una procedura è la seguente:

```
CREATE [or REPLACE] PROCEDURE <nome_procedura> [(<lista di parametri>)]
IS
<dichiarazione>
BEGIN
  <sequenza di istruzioni>
  [exception
  <routine di gestione dell''eccezione>]
END [<nome procedura>];
```

Si analizzino le clausole in essa presenti:

- La *CREATE PROCEDURE*, obbligatoria, indica che stiamo definendo una procedura.
- La *[or REPLACE]*, opzionale, serve per ricreare la procedura nel caso in cui essa sia già stata definita in precedenza.
- La *<nome_procedura>*, obbligatoria, indica il nome con cui la procedura sarà identificata;
- La *[(<lista parametri>)]*, opzionale, consente di inserire una serie di parametri che possono essere passati alla procedura. In pratica è una sequenza del tipo:

  ```
  <nome_parametro> [IN | OUT | IN OUT] <tipo_dato>
  ```

  dove *tipo_dato* non deve specificare lunghezza, precisione o scala in quanto essi vengono derivati dall''ambiente da cui la procedura viene richiamata. Ad esempio, ci sarebbe un errore se si scrivesse come tipo dato *varchar2(30 byte)*, in quanto basterebbe specificare il tipo di dato semplicemente come *varchar2*.
- La clausola *IN*, *OUT*, *IN OUT*, opzionale, specifica il modo nel quale il parametro è utilizzato. Per default, l''utilizzo di un parametro è in modalità IN. IN implica che il parametro può essere utilizzato nella procedura ma non può essere modificato. OUT implica che ad esso può essere assegnato un valore nella procedura ma non può essere utilizzato. IN OUT, infine, implica che il parametro può essere utilizzato sia in lettura che scrittura.

- La clausola *IS* sostituisce, in sostanza, la DECLARE.

Una procedura, inoltre, può essere richiamata utilizzando il comando *CALL* nel seguente modo: `CALL nome_procedura([lista parametri]);`

Invece, può essere eliminata tramite il comando *DROP* nel seguente modo: `DROP nome_procedura.`

Nel file ***SkyDoc_SQLData.sql***, è proprio ciò che viene fatto. In esso, infatti, è possibile ritrovare le seguenti procedure:

- `CREATE OR REPLACE FUNCTION GET_CRC`
- `CREATE OR REPLACE PROCEDURE CREATE_ATTRIBUTE`
- `CREATE OR REPLACE PROCEDURE CREATE_ENTITY`
- `CREATE OR REPLACE PROCEDURE CREATE_SUBJECT_AREA`
- `CREATE OR REPLACE PROCEDURE CREATE_ENTITIES`: Si occupa nello specifo di tutte le entità e di tutti gli attributi del Sotto-sistema di Sicurezza integrato. Il sotto-sistema di sicurezza integrato di SkyGen consente di definire regole di autenticazione e di autorizzazione, nell"ambito di uno o piu" sistemi esterni. In particolare, gestisce: gli utenti di un"organizzazione, con sistema di memorizzazione della password personalizzabile (hash, MDA5, plain, ecc...),la definizione di una o più applicazioni sottoposte acontrollo di sicurezza, la definizione di uno o più profili di autorizzazione, la definizione di operazioni elementari da sottoporre a controllo autorizzazione, la memorizzazione automatica delle operazioni eseguite a sistema. Inoltre, consente di definire regole di profilazione e di controllo autorizzazione avanzate basate sulla valutazione real-time di informazioni provenienti dal contesto di esecuzione dell"operazione stessa.
- `CREATE OR REPLACE PROCEDURE CREATE_USER_FUNCTION`
- `CREATE OR REPLACE PROCEDURE CREATE_CHOOSE_ELEM_FUNC`: Gestisce la lista complessa di selezione. In particolar modo verrà gestita la finestra popup di selezione, in quanto essa agevola la compilazione dei forms utente, attraverso la selezione di elementi da una lista che si apre in una nuova finestra. È possibile

filtrare gli elementi della lista attraverso la digitazione di criteri di ricerca nei campi del form dove risiede il pulsante di attivazione della lista. Inoltre la lista risulta ordinabile e paginabile (in presenza di molti elementi). Dall"elenco degli elementi è possibile selezionare un elemento attraverso la selezione del link in corrispondenza della riga prescelta, nonchè eliminare la selezione corrente, piuttosto che annullare l"operazione di selezione.

- **CREATE OR** REPLACE **PROCEDURE** CREATE_CHOOSE_SCENARIO
- **CREATE OR** REPLACE **PROCEDURE** CREATE_NEW_ELEM_FUNC
- **CREATE OR** REPLACE **PROCEDURE** CREATE_SEARCH_ELEM_FUNC: Funzione elementare di ricerca elementi presenti in base dati. La funzione consente di specificare diversi criteri di ricerca. Il risultato della ricerca viene riportato in una elenco.
- **CREATE OR** REPLACE **PROCEDURE** CREATE_LIST_ELEM_FUNC: L' elenco dei risultati può essere:
    - Paginato - attraverso la selezione di un numero di pagina specifico, oppure attraverso i link di paginazione: precedente, successivo, prima ed ultima pagina.
    - Ordinato - l"ordinamento può essere effettuato in maniera ascendente e discendente (un"icona mostrerà il livello attuale di ordinamento).
    - Perfezionato - gli elementi visualizzati possono essere ulteriormente filtrati utilizzando la funzione elementare precedente.
    - Visualizzo e Gestito - attraverso la selezione del link in corrispondenza dell"elemento prescelto, è possibile accedere alla pagina di visualizzazione di dettaglio dello stesso.
    - Esportato - l"intero elenco può essere esportato in formato Excel.
    - Stampato - l"intero elenco può essere stampato in formato PDF.

Inoltre dalla lista è possibile richiedere la creazione di un nuovo elemento attraverso apposito pulsante.

- **CREATE OR REPLACE PROCEDURE** CREATE_EXPORT_ELEM_FUNC: Funzione elementare di esportazione elenco elementi in formato Excel.
- **CREATE OR REPLACE PROCEDURE** CREATE_PRINT_ELEM_FUNC: Funzione elementare di stampa elenco elementi di tipo in formato PDF.
- **CREATE OR REPLACE PROCEDURE** CREATE_VIEW_ELEM_FUNC: Funzione elementare di visualizzazione dettaglio degli elementi nel database.
- **CREATE OR REPLACE PROCEDURE** CREATE_EDIT_ELEM_FUNC: Funzione elementare di modifica elemento.
- **CREATE OR REPLACE PROCEDURE** CREATE_DELETE_ELEM_FUNC: Funzione elementare di eliminazione elemento.
- **CREATE OR REPLACE PROCEDURE** CREATE_SUBLIST_ELEM_FUNC: Funzione elementare di visualizzazione elenco.
- **CREATE OR REPLACE PROCEDURE** CREATE_CHECK_ELEM_FUNC: Funzione elementare di associazione nuovo elemento.
- **CREATE OR REPLACE PROCEDURE** CREATE_UNCHECK_ELEM_FUNC: Funzione elementare di disassociazione elemento.
- **CREATE OR REPLACE PROCEDURE** CREATE_ADD_ELEM_FUNC: Funzione elementare di aggiunta nuovo elemento.
- **CREATE OR REPLACE PROCEDURE** CREATE_REMOVE_ELEM_FUNC: Funzione elementare di aggiunta nuovo elemento.
- **CREATE OR REPLACE PROCEDURE** CREATE_ELEM_FUNC_SCENARIO
- **CREATE OR REPLACE PROCEDURE** CREATE_RIGA_SCENARIO
- **CREATE OR REPLACE PROCEDURE** CREATE_ELEM_FUNC_ACTION
- **CREATE OR REPLACE PROCEDURE** CREATE_ELEM_FUNC_CHECK
- **CREATE OR REPLACE PROCEDURE** CREATE_ELEM_FUNC_FIELD
- **CREATE OR REPLACE PROCEDURE** CREATE_MACRO_FUNCTION1. È la macrofunzione che si occupa del sotto-sistema di Sicurezza. Alcuni suoi requisiti sono:
    - Gestione Security Application, che prevede:

- Definizione dell"entità "Security Application".
- Definizione logica di un' applicazione, esterna o interna, sottoposta a controllo di sicurezza.
- Definizione del requisito.

L'utente attraverso le funzionalità di Gestione "Security Application" può manutenere i dati del flusso "Security Application". Attraverso le schermate di gestione, infatti, l'utente può:

1) Ricercare e visualizzare liste di elementi Security Application.
2) Esportare tali liste sia in formato PDF che in formato Excel.
3) Inserire un nuovo elemento Security Application.
4) Modificare un elemento Security Application esistente in archivio.
5) Eliminare un elemento Security Application esistente dall"archivio.

Inoltre, dalla funzionalità di visualizzazione dettaglio e modifica, è possibile visualizzare e gestire, attraverso apposite schede, le seguenti entità correlate:

- "Security Elementary Operation" per la quale è possibile: visualizzare l'elenco di Security Elementary Operation appartenenti a Security Application, aggiungere Security Elementary Operation a Security Application, eliminare Security Elementary Operation da Security Application.
- "Security User Profile" per la quale è possibile: Visualizzare l"elenco di Security User Profile appartenenti a Security Application, Aggiungere Security User Profile a Security Application, Eliminare Security User Profile da Security Application.

- "Gestione Security User", che prevede:
  - Definizione dell"entità "Security User"
  - Anagrafica degli utenti definiti nel sotto-sistema di sicurezza
  - Definizione del requisito

L'utente attraverso le funzionalità di Gestione "Security User" può manutenere i dati del flusso "Security User". Infatti, attraverso le schermate di gestione l"utente può: ricercare e visualizzare liste di elementi Security User, esportare tali liste sia in formato PDF che in formato Excel, inserire un nuovo elemento Security User, modificare un elemento Security User esistente in archivio, eliminare un elemento Security User esistente dall"archivio.

Inoltre, dalla funzionalità di visualizzazione dettaglio e modifica, è possibile visualizzare e gestire, attraverso apposite schede, la seguente entità correlata: "Security Profile Per User", per la quale è possibile: visualizzare l' elenco di Security Profile Per User appartenenti a Security User, associare nuovi elementi Security Profile Per User a Security User, rimuovere una o più associazioni di Security Profile Per User da Security User.

- "Gestione Security User Profile", che prevede:
  - Definizione dell"entità "Security User Profile"
  - Profili definiti per ogni applicazione definita al sotto-sistema di sicurezza
  - Definizione del requisito

L'utente attraverso le funzionalità di Gestione "Security User Profile" può manutenere i dati del flusso "Security User Profile". Attraverso le schermate di gestione l'utente può: ricercare e visualizzare liste di elementi Security User Profile, esportare tali liste sia in formato PDF che in formato Excel, inserire un nuovo elemento Security User Profile, modificare un elemento Security User Profile esistente in archivio, eliminare un elemento Security User Profile esistente dall"archivio. Inoltre, dalla funzionalità di visualizzazione dettaglio e modifica, è possibile visualizzare e gestire, attraverso apposite schede, le seguenti entità correlate:"Security Operation Per Profile" per la quale è possibile: visualizzare l"elenco di Security Operation Per Profile appartenenti a Security User Profile, associare nuovi elementi Security Operation Per Profile a Security User Profile, rimuovere una o più associazioni di Security Operation Per Profile da Security User Profile. "Security Profile Per User" per la quale è possibile:

visualizzare l"elenco di Security Profile Per User appartenenti a Security User Profile, aggiungere Security Profile Per User a Security User Profile, eliminare Security Profile Per User da Security User Profile.

Un'ulteriore macrofunzione, invece, raccoglie tutte le funzionalità necessarie a soddisfare le esigenze comuni all"intero sistema come:

- Liste di selezione elementi semplici (combo boxes).
- Funzioni di selezione elementi complesse con possibilità di ricerca, ordinamento e paginazione tra gli elementi proposti.

Il requisito, qui presente, raccoglie tutte le funzionalità necessarie a soddisfare le esigenze comuni all"intero sistema. Le liste di dati selezionabili da combo riguardano le entità: Dotazione, Prenotazione, Security Application, Security Elementary Operation, Security User Profile, Tipo Prenotazione. Le liste di dati selezionabili da finestra di popup, invece, riguardano le entità: Aula, Personale, Security Elementary Operation, Security User, Security User Profile.

Si consideri, infine, il file ***SQL_Data.sql***. In esso è possibile trovare le operazioni che si occupano di popolare il database. Sono riportare di seguito alcune di esse e di alcune è anche riportato il codice :

- Creazione Applicazione

```
INSERT INTO SEC_APP ( APP_ID, APP_DESC )
VALUES ('SPACE', 'Space rel. 1.0');
```

- Creazione Profili

```
INSERT INTO SEC_USER_PROF ( APP_ID, PROF_NAME, PROF_DESC, I_E_FLAG)
VALUES ('SPACE', 'ADMIN', 'Profilo che puo operare soltanto sulle
funzioni di sicurezza', 'I');
INSERT INTO SEC_USER_PROF ( APP_ID, PROF_NAME, PROF_DESC, I_E_FLAG)
VALUES ('SPACE', 'READONLY', 'Profilo che puo operare soltanto
sulle funzioni di gestione ed in sola lettura', 'I');
INSERT INTO SEC_USER_PROF ( APP_ID, PROF_NAME, PROF_DESC, I_E_FLAG)
VALUES ('SPACE', 'INSERTONLY', 'Profilo che aggrega solo le
funzioni di inserimento sulle tabelle non amministrative', 'I');
INSERT INTO SEC_USER_PROF ( APP_ID, PROF_NAME, PROF_DESC, I_E_FLAG)
VALUES ('SPACE', 'EDITONLY', 'Profilo che aggrega solo le funzioni
di modifica sulle tabelle non amministrative', 'I');
INSERT INTO SEC_USER_PROF ( APP_ID, PROF_NAME, PROF_DESC, I_E_FLAG)
VALUES ('SPACE', 'DELETEONLY', 'Profilo che aggrega solo le
funzioni di cancellazione sulle tabelle non amministrative', 'I');
```

- Creazioni Utenti

```
INSERT INTO SEC_USER ( USER_NAME, PASSWORD, FIRST_NAME, LAST_NAME )
VALUES ('SPACE_ALL_USER', 'd41d8cd98f00b204e9800998ecf8427e',
'Utente che', 'fa tutto');
INSERT INTO SEC_USER ( USER_NAME, PASSWORD, FIRST_NAME, LAST_NAME )
VALUES ('SPACE_ADMIN_USER', 'd41d8cd98f00b204e9800998ecf8427e',
'Utente che', 'amministra');
INSERT INTO SEC_USER ( USER_NAME, PASSWORD, FIRST_NAME, LAST_NAME )
VALUES ('SPACE_USER', 'd41d8cd98f00b204e9800998ecf8427e', 'Utente
che', 'gestisce');
INSERT INTO SEC_USER ( USER_NAME, PASSWORD, FIRST_NAME, LAST_NAME )
VALUES ('SPACE_READ_USER', 'd41d8cd98f00b204e9800998ecf8427e',
'Utente che', 'vede soltanto');
INSERT INTO SEC_USER ( USER_NAME, PASSWORD, FIRST_NAME, LAST_NAME )
VALUES ('SPACE_EDIT_USER', 'd41d8cd98f00b204e9800998ecf8427e',
'Utente che', 'vede e modifica');
INSERT INTO SEC_USER ( USER_NAME, PASSWORD, FIRST_NAME, LAST_NAME )
VALUES ('SPACE_NODEL_USER', 'd41d8cd98f00b204e9800998ecf8427e',
'Utente che', 'non elimina');
```

- Profilazioni degli utenti: associazione degli utenti ai profili

- Entità Aula

```
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 0, 0, '/manage/aula/init*', 'Gestione Aula -
Attivazione da menu', NULL, 'Gestione Aula selezionata da menu');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 0, 'SPACE', 'READONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 1, 0, '/manage/aula/search*', 'Gestione Aula -
Esecuzione di una ricerca', 'Y', 'Effettuata ricerca su Aula');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 1, 'SPACE', 'READONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 2, 0, '/manage/aula/view/**', 'Gestione Aula -
Visualizzazione dettaglio', NULL, 'Visualizzato dettaglio elemento
Aula. Dati: $ctx.form.theAula ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 2, 'SPACE', 'READONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 3, 0, '/manage/aula/new*', 'Gestione Aula -
Richiesta creazione nuovo elemento', NULL, 'Effettuata Richiesta
creazione nuovo elemento su Aula');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 3, 'SPACE', 'INSERTONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 4, 0, '/manage/aula/saveNewElement*', 'Gestione
Aula - Inserimento nuovo elemento in base dati', NULL, 'Effettuato
inserimento nuovo elemento Aula. Dati: $ctx.form.theAula ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 4, 'SPACE', 'INSERTONLY');
```

```
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 5, 0, '/manage/aula/edit/**', 'Gestione Aula -
Richiesta modifica elemento', NULL, 'Effettuata richiesta modifica
elemento Aula. Dati: $ctx.form.theAula ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 5, 'SPACE', 'EDITONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 7, 0, '/manage/aula/delete/**', 'Gestione Aula -
Richiesta eliminazione elemento', NULL, 'Effettuata richiesta
eliminazione elemento Aula. Dati: $ctx.form.theAula ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 7, 'SPACE', 'DELETEONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 8, 0, '/manage/aula/print/**', 'Gestione Aula -
Esportazione del dettaglio in PDF, XLS e RTF', NULL, 'Esporta
dettaglio elemento Aula. Dati: $ctx.form.theAula ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 8, 'SPACE', 'READONLY');
```

- Entità Aula Dotazione
- Entità Dotazione
- Entità Personale
- Entity Prenotazione

```
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 40, 400, '/manage/prenotazione/init*', 'Gestione
Prenotazione - Attivazione da menu', NULL, 'Gestione Prenotazione
selezionata da menu');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 40, 'SPACE', 'READONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 41, 400, '/manage/prenotazione/search*', 'Gestione
Prenotazione - Esecuzione di una ricerca', 'Y', 'Effettuata ricerca
su Prenotazione');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 41, 'SPACE', 'READONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 42, 400, '/manage/prenotazione/view/**', 'Gestione
Prenotazione - Visualizzazione dettaglio', NULL, 'Visualizzato
dettaglio elemento Prenotazione. Dati: $ctx.form.thePrenotazione
');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 42, 'SPACE', 'READONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 43, 400, '/manage/prenotazione/new*', 'Gestione
Prenotazione - Richiesta creazione nuovo elemento', NULL,
'Effettuata Richiesta creazione nuovo elemento su Prenotazione');
```

```
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 43, 'SPACE', 'INSERTONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 44, 400, '/manage/prenotazione/saveNewElement*',
'Gestione Prenotazione - Inserimento nuovo elemento in base dati',
NULL, 'Effettuato inserimento nuovo elemento Prenotazione. Dati:
$ctx.form.thePrenotazione ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 44, 'SPACE', 'INSERTONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 45, 400, '/manage/prenotazione/edit/**', 'Gestione
Prenotazione - Richiesta modifica elemento', NULL, 'Effettuata
richiesta modifica elemento Prenotazione. Dati:
$ctx.form.thePrenotazione ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 45, 'SPACE', 'EDITONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 47, 400, '/manage/prenotazione/delete/**',
'Gestione Prenotazione - Richiesta eliminazione elemento', NULL,
'Effettuata richiesta eliminazione elemento Prenotazione. Dati:
$ctx.form.thePrenotazione ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 47, 'SPACE', 'DELETEONLY');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 48, 400, '/manage/prenotazione/print/**',
'Gestione Prenotazione - Esportazione del dettaglio in PDF, XLS e
RTF', NULL, 'Esporta dettaglio elemento Prenotazione. Dati:
$ctx.form.thePrenotazione ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 48, 'SPACE', 'READONLY');
```

- Entità Prenotazione Personale

- Entità Security Application

```
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 60, 600, '/manage/securityApplication/init*',
'Gestione Security Application - Attivazione da menu', NULL,
'Gestione Security Application selezionata da menu');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 60, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 61, 600, '/manage/securityApplication/search*',
'Gestione Security Application - Esecuzione di una ricerca', 'Y',
'Effettuata ricerca su Security Application');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 61, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 62, 600, '/manage/securityApplication/view/**',
'Gestione Security Application - Visualizzazione dettaglio', NULL,
```

```
'Visualizzato dettaglio elemento Security Application. Dati:
$ctx.form.theSecurityApplication ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 62, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 63, 600, '/manage/securityApplication/new*',
'Gestione Security Application - Richiesta creazione nuovo
elemento', NULL, 'Effettuata Richiesta creazione nuovo elemento su
Security Application');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 63, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 64, 600,
'/manage/securityApplication/saveNewElement*', 'Gestione Security
Application - Inserimento nuovo elemento in base dati', NULL,
'Effettuato inserimento nuovo elemento Security Application. Dati:
$ctx.form.theSecurityApplication ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 64, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 65, 600, '/manage/securityApplication/edit/**',
'Gestione Security Application - Richiesta modifica elemento',
NULL, 'Effettuata richiesta modifica elemento Security Application.
Dati: $ctx.form.theSecurityApplication ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 65, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 67, 600, '/manage/securityApplication/delete/**',
'Gestione Security Application - Richiesta eliminazione elemento',
NULL, 'Effettuata richiesta eliminazione elemento Security
Application. Dati: $ctx.form.theSecurityApplication ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 67, 'SPACE', 'ADMIN');
```

- Entità Security Elementary Operation
- Entità Security Operation Log
- Entità Security User Profile
- Entità Tipo Prenotazione
- Entità Security User

```
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 90, 900, '/manage/securityUser/init*', 'Gestione
Security User - Attivazione da menu', NULL, 'Gestione Security User
selezionata da menu');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 90, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
```

```
VALUES ('SPACE', 91, 900, '/manage/securityUser/search*', 'Gestione
Security User - Esecuzione di una ricerca', 'Y', 'Effettuata
ricerca su Security User');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 91, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 92, 900, '/manage/securityUser/view/**', 'Gestione
Security User - Visualizzazione dettaglio', NULL, 'Visualizzato
dettaglio elemento Security User.Dati:$ctx.form.theSecurityUser ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 92, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 93, 900, '/manage/securityUser/new*', 'Gestione
Security User - Richiesta creazione nuovo elemento', NULL,
'Effettuata Richiesta creazione nuovo elemento su Security User');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 93, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 94, 900, '/manage/securityUser/saveNewElement*',
'Gestione Security User - Inserimento nuovo elemento in base dati',
NULL, 'Effettuato inserimento nuovo elemento Security User. Dati:
$ctx.form.theSecurityUser ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 94, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 95, 900, '/manage/securityUser/edit/**', 'Gestione
Security User - Richiesta modifica elemento', NULL, 'Effettuata
richiesta modifica elemento Security User. Dati:
$ctx.form.theSecurityUser ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 95, 'SPACE', 'ADMIN');
INSERT INTO SEC_EL_OP ( APP_ID, OP_ID, SEQ_NO, OP_NAME,
OP_DESCRIPTION, ALLOW_DENY_FLAG, AUD_DATA )
VALUES ('SPACE', 97, 900, '/manage/securityUser/delete/**',
'Gestione Security User - Richiesta eliminazione elemento', NULL,
'Effettuata richiesta eliminazione elemento Security User. Dati:
$ctx.form.theSecurityUser ');
INSERT INTO SEC_OP_X_PROF ( OP_APP_ID, OP_ID, PROF_APP_ID,
PROF_NAME ) VALUES ('SPACE', 97, 'SPACE', 'ADMIN');
```

Un'ultima considerazione da fare riguardo il database è capire come si configura la connessione della web application al database. In un'applicazione web data intensive, in cui gli accessi al database sono molti e spesso concorrenti, più utenti connessi all'applicazione contemporaneamente, il pattern d'uso standard del JDBC presenta notevoli problemi. Infatti, l'apertura di una connessione al database è di solito un'operazione costosa, quanto richiede: caricamento driver, connessione al DBMS server

e autenticazione. È necessario limitare l'overhead dovuto a queste operazioni, per rendere la web application più veloce possibile. Il connection pooling è una tecnica che permette di snellire le procedure di apertura delle connessioni JDBC utilizzando una cache di connessione, chiamata connection pool. Il pool mantiene una serie di connessioni al database già aperte e inizializzate. Quando l'applicazione vuole connettersi, preleva dal pool una connessione già pronta, sulla quale potrà operare direttamente. Quando l'applicazione chiude la connessione, questa, in realtà, rimane aperta e viene inserita nel pool, in attesa di essere riutilizzata per altre richieste. Il pool viene inizialmente riempito con un certo numero di connessioni pronte. Tuttavia, se il pool si svuota e c'è richiesta di nuove connessioni, queste vengono create automaticamente allargando il pool. Le connessioni lasciate inutilizzate nel pool per troppo tempo possono venire chiuse automaticamente per liberare le corrispondenti risorse del DBMS.

Il supporto al connection pooling, incorporato nelle più recenti versioni del JDBC, deve ovviamente avere supporto da parte del software. Tutti i più diffusi application server forniscono un'implementazione del connection pooling, ed esistono anche librerie esterne utilizzabili per aggiungere questo supporto al di fuori degli application server.

In particolare, per configurare il connection pooling in Tomcat, si procede nel modo seguente:

- si configura la connessione al database nel contesto della web application, agendo sul file context.xml, file di configurazione specifica dell'applicazione.
- Si inserisce un riferimento alla connessione, che diventa così una risorsa dell'applicazione di tipo DataSource.
- Si crea una normale connessione JDBC attraverso il DataSource.
- Si chiude la connessione al termine del suo uso.

Si consideri nello specifico la configurazione dell'applicazione per Tomcat, attraverso l'analisi del file context.xml. Gli elementi principali che si possono esaminare in tal file di configurazione sono:

- Resource: è l'elemento un cui è configurata la connessione. Esso contiene tutte le caratteristiche, compreso il driver, l' username, la password e la stringa di connessione.
- Type: tale campo deve contenere il package utilizzato per l'estensione di JDBC. Nel caso specifico, questo campo dovrà contenere il valore : javax.sql.DataSource. La suddetta libreria permette di estendere JDBC attraverso l'introduzioni dei seguenti meccanismi : JDBC Data Source, Connection Pooling, Transazioni distribuite, Rowset. È proprio la tecnica del Connection Pooling che viene utilizzata in tal contesto.
- MaxActive e MaxIdle: servono a dimensionare il pool. In particolare, maxActive indica quante connessioni al massimo il pool dovrà contenere, invece maxIdle indica quante connessioni inutilizzate sono ammesse nel pool in ogni momento.

Di seguito è riportato il file context.xml dell'applicazione:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
 <!--
  <Resource name="jdbc/space/DataSource"
                auth="Container"
                type="javax.sql.DataSource"
                username="u_space_svil"
                password="space"
                driverClassName="oracle.jdbc.driver.OracleDriver"
                url="jdbc:oracle:thin:@95.110.190.253:1521:ORCLSVI"
                maxActive="8"
                maxIdle="4"/>
    -->

     <Resource name="jdbc/space/DataSource"
               auth="Container"
               type="javax.sql.DataSource"
               username="u_space_svil"
               password="space"
               driverClassName="oracle.jdbc.driver.OracleDriver"
               url="jdbc:oracle:thin:@localhost:1522:ORCL11"
               maxActive="8"
               maxIdle="4"/>


     <!-- Transaction declaration example for Tomcat using JOTM see
http://static.raibledesigns.com/downloads/howto-tomcat-jotm.html -->
     <!--
    <Transaction factory="org.objectweb.jotm.UserTransactionFactory"
jotm.timeout="60"/>
     -->

</Context>
```

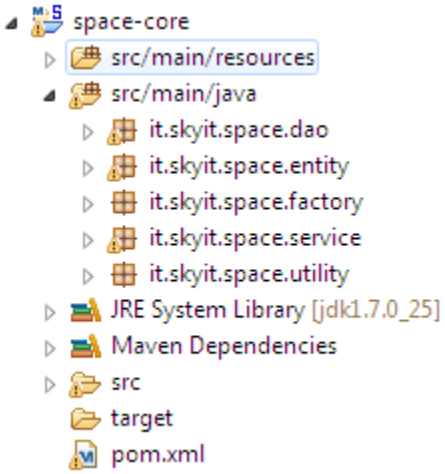
Come è possibile notare dal codice esistono due elemento Resource. Essi possono essere utilizzati solo in modo esclusivo, in quanto la differenza tra di loro sta proprio nella stringa di connessione, l'url. Nel primo caso l'url è quello relativo al database presente sul server dell'azienda SkyIT, nel secondo caso, invece, è la stringa di connessione al database locale.

5.8.4 Codice generato

Dalla generazione, effettuata grazie all'utilizzo dello strumento SkyGen, sono stati ottenuti, come già accennato in precedenza i seguenti progetti, che implementano l'applicazione web: space-core, space-ear, space-impl, space-web.

Soffermiamoci, nello specifico, su cosa è contenuto in ognuno di essi, per quanto concerne il codice java generato.

- **Space-core.** Come si evince dall'immagine tale progetto contiene nella source folder 5 package:



1. it.skyit.space.dao: le interfacce contenute in questo package definiscono la persistanza, limitatamente agli accessi elementari, relativa a tutte le entità: Aula,AulaDotazione,Dotazione,Prenotazione,TipoPrenotazione,PrenotazioneP

ersonale,Personale,BaseEntity,SecurityApplication,SecurityElementaryOperation,SecurityOperationLog,SecurityOperationPerProfile,SecurityProfilePerUser, SecurityUser, e SecurityUserProfile. Le operazioni elementari sono:

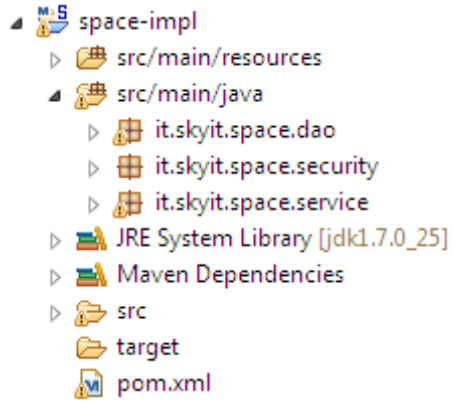
- FindByPrimaryKey: ricerca un'istanza di un'entità accedendo per chiave primaria.
- ExistByPrimaryKey: Controlla l'esistenza di un'entità accedendo per chiave primaria.
- FindByFilter: estrazione di tutte le istanze di un'entità corrispondenti ai parametri di ricerca specificati.
- FindAll: estrazione di tutte le istanze di un'entità accedendo senza parametri.
- Insert: inserimento di un'entità.
- Update: aggiornamento di un'entità.
- Delete: cancellazione di un'entità accedendo per chiave primaria.
- Store: esegue l'inserimento oppure l'aggiornamento, in funzione dello stato interno dell'entity fornito in input.
- Inoltre è previsto un finder di un'entità per ogni Entity da essa referenziata, ognuno dei quali denominato: FindBy <code> EntityName </code>.

2. it.skyit.space.entity : Ogni file java ivi contenuto fornisce tutte le proprietà elementari relative ad ogni tabella. Esso fornisce, inoltre, i metodi get e set per ognuna delle proprietà. La navigazione da un'entità verso le entity correlate è garantita dalla presenza di una proprietà Collection per ogni tabella derivata, mentre la navigazione verso le entity superiori è garantita dalla presenza di una Entity per ogni foreign key definita per la tabella referenziata. Infine viene offerto un get e set anche per tutti gli attributi esterni.

3. it.skyit.space.factory: questo package presenta tutti i factory che hanno l'obiettivo di fornire metodi utili per ottenere oggetti relativi ad ogni entità. Tali metodi hanno l'obiettivo di:
    - restituire una nuova istanza dell'implementazione di default di un' entità;
    - restituire una nuova istanza dell'implementazione di default di un'entità già inizializzata con il rowId fornito in input;
    - restituire una nuova istanza dell'implementazione di default di una EntityList, si può trattare di un EntityList implementation in caso di JDBC persistence implementation, oppure di una semplice Collection implementation nel caso di utilizzo di hibernate come gestore della persistenza;
    - restituire l'istanza singletone del servizio EntityService. Il servizio restituito è un proxy implementation service bean ed eredita tutto il supporto al transaction management e technology implementation iniettata dal container spring. Si utilizza tale funzionalità per mantenersi disaccoppiati dalla tecnologia sottostante (POJO, EJB, etc...). Altrimenti, se si è proprio sicuri che il servizio sarà un local POJO, si utilizza la sintassi: *<code> LocalContext.getBean ("aulaServiceImpl") </code>*, che fornisce prestazioni leggermente superiori.
4. it.skyit.space.service: contiene l'insieme delle interfacce in cui sono definiti tutti i metodi di gestione di ogni entità.
5. it.skyit.space.utility: Contiene provider in grado di fornire qualsivoglia oggetto definito nel container spring quali: Beans, Services e Dao. I metodi qui contenuti consentono di :

- distinguere gli oggetti di dominio dell'applicazione corrente in un contesto Enterprise, ovvero nel caso di interoperabilità con altre applicazioni integrate;
- restituire il riferimento allo Spring bean Factory indentificato dalla chiave univoca CONTEXT_ID;
- impostare il beanFactory nel local context definito per l'applicazione;
- restituire lo spring bean in base all'identificativo fornito in input;
- restituire il riferimento all'info bar manager.

- **Space-impl:** come si evince dall'immagine tale progetto contiene nella source folder 3 package :

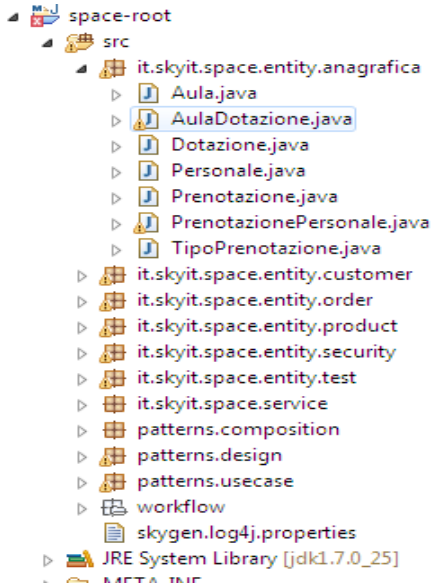


1. it.skyit.space.dao: i file .java contenuti in questo package implementano la persistanza, utilizzando Hibernate come ORM, di tutte le entità presenti nell'applicazione. Le operazioni elementari implementate sono :
   - FindByPrimaryKey: ricerca un'istanza di un'entità accedendo per chiave primaria.
   - ExistByPrimaryKey: Controlla l'esistenza di un'entità accedendo per chiave primaria.
   - FindByFilter: estrazione di tutte le istanze di un'entità corrispondenti ai parametri di ricerca specificati.

- FindAll: estrazione di tutte le istanze di un'entità accedendo senza parametri.
- Insert: inserimento di un'entità.
- Update: aggiornamento di un'entità.
- Delete: cancellazione di un'entità accedendo per chiave primaria.
- Store: esegue l'inserimento oppure l'aggiornamento, in funzione dello stato interno dell'entity fornito in input.
- Inoltre è previsto un finder di un'entità per ogni Entity da essa referenziata, ognuno dei quali denominato: FindBy <code> EntityName </code>.

2. it.skyit.space.security. Sono contenuti, in tale package, tre file:
   - *ApplicationHolderImpl.java:* mantiene informazioni identificative dell'applicazione in oggetto e fornisce servizi utili al sotto-sistema di sicurezza.
   - *AuditLoggerProviderJdbcImpl.java:* è un logger provider che opportunamente pluggato tramite Log4J, consente di storicizzare gli audit records direttamente nella tabella di sicurezza, Security Operation Log.
   - *UserDetailsJdbcDaoImpl.java:* implemetazione JDBC del provider delle credenziali richiesto dal framework di sicurezza Spring Security.
3. it.skyit.space.service: Contiene le Service POJO implementations relative a tutte le entità di dominio, individuate nella stesura del diagramma dell classi. Le classi presenti in tal package implementano tutti i metodi di gestione delle diverse entità, vi è, inoltre, il metodo che restituisce il riferimento al corrispettivo DAO e il metodo che imposta il riferimento al DAO.

- **Space-root**: di questo progetto ci occuperemo solo del package it.skyit.space.entity.anagrafica, in cui sono contenute le classi java che descrivono l'anagrafica delle diverse entità facenti parti dell'applicazione.



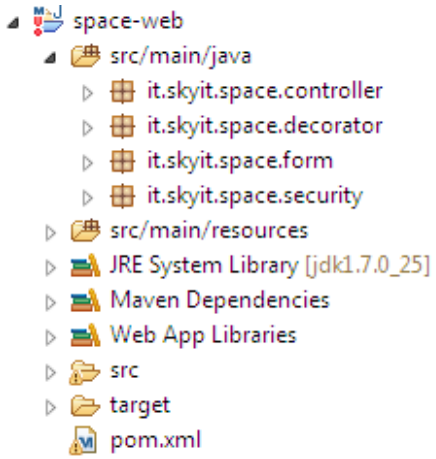
È riportata di seguito, come esempio la classe Aula.java :

```
package it.skyit.space.entity.anagrafica;
/**
*  Rappresenta l'anagrafica delle Aule universitarie oggetto di prenotazione.
* @entityLabelName
* @physicalTableName
* @sequenceName
* @stereotype Entity
*/
public class Aula {
/**
 * Codice univoco dell'Aula
* @isPrimaryKey true
* @isRequired true
* @dataType VARCHAR2(20)
* @attributeDomain String
* @case none
* @attributeLabelName
* @physicalColumnNam
*/
static String aulaId;
/**
* Descrizione Edificio di Ubicazione
* @isPrimaryKey false
* @isRequired true
* @dataType VARCHAR2(40)
* @attributeDomain String
* @attributeLabelName
* @case none
```

```
* @physicalColumnName
*/
String descrizioneEdificio;
/**
* Piano Edificio di Ubicazione
* @isPrimaryKey false
* @isRequired true
* @dataType VARCHAR2(10)
* @attributeDomain String
* @attributeLabelName
* @case none
* @physicalColumnName
*/
String pianoEdificio;
/**
* Capienza in termini di posti a sedere
* @isPrimaryKey false
* @isRequired false
* @dataType NUMBER(4)
* @attributeDomain Code
* @attributeLabelName
* @case none
* @physicalColumnName
*/
Short numeroPosti;
}
```

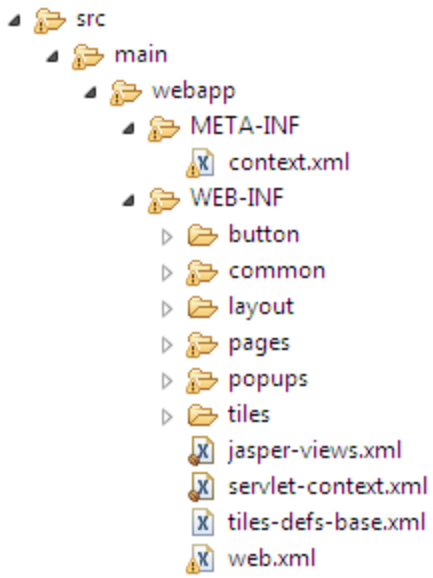
Questo è il file che bisognerà prendere in considerazione nel caso in cui si vogliano apportare modifiche agli attributi delle entità; le modifiche saranno poi, ovviamente, seguite dalla rigenerazione del codice.

- **Space-web.** Come si evince dall'immagine tale progetto contiene nella source folder 4 package:



1. it.skyit.space.controller. Package che contiene Spring MVC POJO Controller per la gestione delle entità dell'applicazione. Il controller effettua la gestione completa di ogni entità, in termini di CRUD operations nonchè, se previsto, operazioni di gestione Master/Details. Attraverso la RequestMapping annotation è possibile controllare l'url base path a cui tale controller risponde. Inoltre ogni metodo di questi controller riportante una RequestMapping annotation, è associato ad una richiesta http il cui path relativo è controllabile attraverso la annotation property "value".
2. it.skyit.space.decorator: Il design pattern decorator consente di aggiungere durante il run-time nuove funzionalità ad oggetti già esistenti. Questo viene realizzato costruendo una nuova classe *decoratore* che "avvolge" l'oggetto originale. Al costruttore del decoratore si passa come parametro l'oggetto originale. È altresì possibile passarvi un differente decoratore. In questo modo, più decoratori possono essere concatenati l'uno all'altro, aggiungendo così in modo incrementale funzionalità alla classe concreta, che è rappresentata dall'ultimo anello della catena.
3. it.skyit.space.form. In questo package sono raccolti tutti i "POJO Form beans" contenenti le entity necessarie alla gestione di ogni possibile entità considerata.
4. it.skyit.space.security: Dopo l'elaborazione, Spring programma il bean FilterInvocationSecurityMetadataSource caricondo in esso tutti i security config parameters leggendoli dalla tabella Security Elementary Operation, facente parte del sottosistema di sicurezza.

In questo progetto, altrettanto importante è la cartella src:



Soffermiamoci in particolare sulla cartella WEB-INF, in essa, come si può vedere sono contenute le sottocartelle button, common, layout, pages, popups, e tiles. Tutte queste cartelle contengono il codice che va a costiture la View dell'applicazione web che si vuole realizzare. Nella sottocartella button, ad esempio, sono presenti le jsp relative ai generici pulsanti, quali elimina, modifica, inserisci nuovo, visualizza, stampa e cosi via, che sono presenti nelle interfacce dell'applicazioni e quindi utilizzabili dagli utenti abilitati. Nella sottocartella pages, invece, sono contenute tutte le jsp "list" e "detail" di ogni possibile interfaccia dell'applicazione Space. Tali jsp offrono contenuti dinamici in formato XML e si basano su un insieme di speciali tag, all'interno di una pagina HTML, con cui possono essere invocate funzioni predefinite sotto forma di codice Java (JSTL) e/o funzioni Javascript.

5.8.5 Customizzazioni del codice generato

Il codice generato dallo strumento SkyGen può essere già considerato una prima versione funzionante dell'applicazione, basterà, infatti, caricare i file di configurazione per il database, avviare il server e aprendo il browser digitare l'indirizzo, a questo punto sarà possibile navigare all'interno dell'applicazione e utilizzare tutte le funzionalità abilitate per

quel particolare profilo con cui si è effettuato il login. È vero che l'applicazione risulta funzionante fin da subito, ma è altrettanto vero che bisognerà apportare delle modifiche per customizzare ciò che è stato generato. Alcune customizzazioni, inizialmente effettuate sono state di semplici modifiche di alcuni elementi della view. Da tutte le interfacce di ricerca sono stati eliminati elementi considerati superflui a ogni tipo di ricerca possibile. Questo è stato possibile accedendo al progetto space-web ed in particolare alle jsp contenute nella cartella pages. Sono state, anche, modificate una serie di label, affinché gli elementi dell'interfaccia si presentassero maggiormente comprensibili agli utenti. Per effettuare queste modifiche si è acceduto al file "ApplicationResources.properties", in cui sono contenute tutte le label. Un'ulteriore e rilevante modifica è stata apportata nella pagina relativa alla visualizzazione delle prenotazioni effettuate. In tale pagina, infatti, era possibile visualizzare una tabella nella quale ogni riga conteneva le informazioni di una prenotazione, ovvero il numero della prenotazione, il nome dell'aula, la data di inizio e fine e l'ora di inizio e fine, ma mancava, ad esempio, il nominativo del responsabile della prenotazione e la motivazione della suddetta prenotazione. Per assolvere a tale mancanza è sorta l'esigenza di aggiungere colonne alla tabella già esistente, ma per far ciò c'è stato il bisogno di creare un decorator, in particolare si è creato un package it.skyit.space.decorator, facente parte del progetto spce-web, nel quale è stato creato il file PrenotazioneDecorator.java. Si ricordi che Il design pattern decorator consente di aggiungere durante il run-time nuove funzionalità ad oggetti già esistenti. Questo è realizzato costruendo una nuova classe *decoratore* che "avvolge" l'oggetto originale. Al costruttore del decoratore si passa come parametro l'oggetto originale. È altresì possibile passarvi un differente decoratore. In questo modo, più decoratori possono essere concatenati l'uno all'altro, aggiungendo così in modo incrementale funzionalità alla classe concreta, che è rappresentata dall'ultimo anello della catena.

Nel nostro caso il file PrenotazioneDecorator conterrà due metodi get, il primo permettere di prelevare dalle informazioni della prenotazione il Responsabile della prenotazione, in caso non ci sia tale metodo restituirà una stringa null; il secondo, invece, restituirà il motivo della prenotazione.

```
public String getResponsabile(){
        String personaleString = "";
        Prenotazione obj = (Prenotazione)getCurrentRowObject();
        Collection personale =  obj.getThePrenotazionePersonale();
        Iterator it = personale.iterator();

        while(it.hasNext()){
           PrenotazionePersonale prenP=(PrenotazionePersonale)it.next();
            if("Responsabile Prenotazione".equals(prenP.getRuoloAsString())){
               personaleString = prenP.getThePersonale().getNome() +"
                                 "+prenP.getThePersonale().getCognome()+ "\n";
            return personaleString;
            }
           }
         return personaleString;
      }

      public String getMotivazione(){
             String MotivazioneString = "";
             Prenotazione obj = (Prenotazione)getCurrentRowObject();
             TipoPrenotazione tipoP =  obj.getTheTipoPrenotazione();
             MotivazioneString =tipoP.getDescrizione();

     return MotivazioneString;

      }
```

Tali metodi saranno richiamati dalla listPrenotazione.jsp, in essa, infatti, è riportata, in formato xml, la tabella che inizialmente si voleva modificare. La modifica avviene aggiungendo le seguenti righe al codice già esistente:

```
decorator="it.skyit.space.decorator.PrenotazioneDecorator">
…
<display:column property="responsabile"
    titleKey="label.thePrenotazione.responsabile" sort="true" class="right"/>
<display:column property="motivazione"
    titleKey="label.thePrenotazione.theTipoPrenotazione" sort="true"
    class="right"/>
```

Apportate tali modifiche, sarà possibile visualizzare la nuova tabella nell'interfaccia di visualizzazione delle prenotazioni effettuate.

Infine, sono state effettuate alcune modifiche che hanno richiesto la rigenerazione. L'entità Aula è stata dotata di un identificativo, AulaId, di tipo integer, ma, in realtà, le aule del dipartimento sono tutte identificate da un nome che è alfanumerico, per questo si ha la necessità di modificare il tipo dell'attributo AulaId, da Integer a String, e per far ciò bisogna effettuare una rigenerazione del codice. Quindi, il primo passo da compiere è la

modifica della classe Aula.java, contenuto nel package it.skyit.space.anagrafica, a questo punto è possibile ripetere la generazione tramite l'utilizzo di SkyGen.

Durante lo sviluppo di una qualsiasi applicazione web, qualsiasi tipo di modifica potrebbe essere effettuato manualmente, mettendo, cioè, direttamente mano al codice, senza pertanto ricorrere alla rigenerazione. In realtà, per alcune modifiche la rigenerazione si ritiene la soluzione ottimale, in quanto la correzione manuale richiederebbe troppo tempo e uno sforzo enorme.

Come già accennato nella descrizione di un diagramma delle classi, i vincoli rappresentano delle informazioni che non possono essere rappresentate in maniera diretta mediante la sintassi grafica del diagramma UML delle classi. Tipicamente, per esprimere queste informazioni viene, infatti, utilizzato il linguaggio OCL, acronimo di Object Constraint Language. La maggior parte dei tool e degli ambienti integrati per la modellazione in UML, però, non gestisce ancora OCL, o lo gestiscono in modo solo parziale. Per questo motivo, un'importante customizzazione si è effettuata manualmente. Il problema che presentava l'applicazione era la possibilità di prenotare una stessa aula in un medesimo giorno e nello stesso orario, oppure in un intervallo temporale in cui risultava essere già prenotata. Per risolvere questo problema si è agito sui seguenti file:

- *PrenotazioneService.java*: qui è dichiarata la seguente funzione `boolean isPrenotazioneValida(Prenotazione thePrenotazione);` tale funzione, verrà utilizzata per trasformare la data e l'ora di inizio e fine prenotazione nel formato *Date*, in modo tale che possano essere adoperate nella query successivamente implementata.
- *PrenotazioneServiceImpl.java*: è, in tal file, implementata la funzione `boolean isPrenotazioneValida(Prenotazione thePrenotazione);` e al suo interno viene chiamata, invece la funzione che effettivamente si occuperà di interrogare il database.

```
public boolean isPrenotazioneValida(Prenotazione thePrenotazione){
    SimpleDateFormat formatterDataParziale = new SimpleDateFormat("dd-
MM-yyyy");
```

```
        SimpleDateFormat formatterDataCompleta = new SimpleDateFormat("dd-
    MM-yyyy-HH:mm:ss");

      //Conversione dataInizio+oraInizio nella dataInizioPrenotazione di
    tipo Date
        String dataInizioAsString =
    formatterDataParziale.format(thePrenotazione.getDateInizio());
         System.out.println(dataInizioAsString);
        String dataInizioCompleta = dataInizioAsString+"-
    "+thePrenotazione.getOraInizio();
         System.out.println(dataInizioCompleta);
         Timestamp dataInizioPrenotazione;
         try {
                dataInizioPrenotazione = new
       Timestamp(formatterDataCompleta.parse(dataInizioCompleta).getTime());
         } catch (ParseException e) {
             e.printStackTrace();
             throw new BusinessException(new UserMessage("Impossibile
             eseguire il parse della data inizio compelta"));
         }

       //Conversione dataFine+oraFine nella dataFinePrenotazione di tipo Date
         String dataFineAsString =
           formatterDataParziale.format(thePrenotazione.getDateFine());
         String dataFineCompleta = dataFineAsString+"-
                                   "+thePrenotazione.getOraFine();
         Timestamp dataFinePrenotazione;
         try {
                dataFinePrenotazione = new
         Timestamp(formatterDataCompleta.parse(dataFineCompleta).getTime());
         } catch (ParseException e) {
                e.printStackTrace();
                throw new BusinessException(new UserMessage("Impossibile
               eseguire il parse della data fine compelta"));
         }

         //chiamata al metodo del dao con
Aula,dataInizioPrenotazione,dataFinePrenotazione per verificare la validità
della prenotazione che si sta
         //inserendo/modificando
         return getPrenotazioneDao().isPrenotazioneValida(
                      thePrenotazione.getAulaId(),
                      dataInizioPrenotazione,
                      dataFinePrenotazione);

   }
```

- *PrenotazioneDao.java*: in questo file è riportate la dichiarazione della seguente funzione `boolean isPrenotazioneValida (String aula, Timestamp dataInizioPrenotazione, Timestamp dataFinePrenotazione);`
- *PrenotazioneDaoImpl.java*: è qui riportata l'implementazione della funzione dichiarata nel file PrenotazioneDao.java

```
public boolean isPrenotazioneValida (String aula, Timestamp
dataInizioPrenotazione, Timestamp dataFinePrenotazione){

    StringBuilder builder = new StringBuilder("");
    builder.append("SELECT * ");
    builder.append("FROM prenotazione");
    builder.append("WHERE aula_id =:aula");
    builder.append("AND(:dataInizio =");
    builder.append("TO_DATE ((TO_CHAR(date_inizio,'DD/MM/YYYY'))");
    builder.append("|| ' '");
    builder.append("||NVL (ora_inizio, '00:00:00'),");
    builder.append("'DD/MM/YYYY HH24:MI:SS'");
    builder.append(")");
    builder.append("OR (:dataFine =");
    builder.append("TO_DATE ((TO_CHAR (date_fine, 'DD/MM/YYYY'))");
    builder.append("|| ' '");
    builder.append("|| NVL (ora_fine, '00:00:00'),");
    builder.append("'DD/MM/YYYY HH24:MI:SS'");
    builder.append(")");
    builder.append("OR (:dataInizio >");
    builder.append("TO_DATE ((TO_CHAR (date_inizio, 'DD/MM/YYYY'))");
    builder.append("|| ' '");
    builder.append("|| NVL (ora_inizio, '00:00:00'),");
    builder.append("'DD/MM/YYYY HH24:MI:SS'");
    builder.append(")");
    builder.append("AND:dataInizio <");
    builder.append("TO_DATE (   (TO_CHAR (date_fine, 'DD/MM/YYYY'))");
    builder.append("|| ' '");
    builder.append("|| NVL (ora_fine, '00:00:00'),");
    builder.append("'DD/MM/YYYY HH24:MI:SS'");
    builder.append(")");
    builder.append(")");
    builder.append("OR (:dataFine <");
    builder.append("TO_DATE ((TO_CHAR (date_fine, 'DD/MM/YYYY'))");
    builder.append("|| ' '");
    builder.append("|| NVL (ora_fine, '00:00:00'),");
    builder.append("'DD/MM/YYYY HH24:MI:SS'");
    builder.append(")");
    builder.append("AND:dataFine >");
    builder.append("TO_DATE ((TO_CHAR (date_inizio, 'DD/MM/YYYY'))");
    builder.append("|| ' '");
    builder.append("|| NVL (ora_inizio, '00:00:00'),");
    builder.append("'DD/MM/YYYY HH24:MI:SS'");
    builder.append(")");
    builder.append(")");
    builder.append("OR (:dataInizio <");
    builder.append("TO_DATE ((TO_CHAR (date_inizio, 'DD/MM/YYYY'))");
    builder.append("|| ' '");
    builder.append("|| NVL (ora_inizio, '00:00:00'),");
    builder.append("'DD/MM/YYYY HH24:MI:SS'");
    builder.append(")");
    builder.append("AND:dataFine >");
    builder.append("TO_DATE (   (TO_CHAR (date_fine, 'DD/MM/YYYY'))");
    builder.append("|| ' '");
    builder.append("|| NVL (ora_fine, '00:00:00'),");
    builder.append("'DD/MM/YYYY HH24:MI:SS'");
    builder.append(")");
    builder.append(")");
```

```
            builder.append(")");
            builder.append(")");

            SQLQuery query =
                  getCurrentSession().createSQLQuery(builder.toString());
            query.setString("aula", aula);
            query.setTimestamp("dataInizio",dataInizioPrenotazione);
            query.setTimestamp("dataFine",dataFinePrenotazione);

            List <Object> result =  query.list();

            return result.isEmpty();

      }
```

Come è possibile vedere dal codice, la query ha l'obiettivo di individuare all'interno della tabella "Prenotazione" se esiste già una prenotazione relativa ad una determinata aula in una determinata data ed in un determinato arco temporale. In questo modo, infatti, sarà possibile effettuare prenotazioni solo nel caso in cui la nuova prenotazione non si sovrappone in alcun modo con quelle già esistenti. Naturalmente, questo particolare controllo dovrà esser fatto nel momento in cui si vuole effettuare una nuova prenotazione. È per questo motivo che la funzione *isPrenotazioneValida* verrà richiamata nella funzione *public String saveNewElement (@Valid PrenotazioneForm form, BindingResult errorResult, Model model)*, funzione che elabora i dati inseriti e procede all'attivazione della funzione di inserimento dati nel db, relativamente all'elemento di tipo Prenotazione, ed è presente nel file *ManagePrenotazioneController*.

Un'ulteriore modifica, che ha richiesto l'aggiunta di codice ai file elencati in precedenza, è stata il controllo della coerenza delle date. Tale controllo deve essere fatto per evitare che la data o l'ora di fine di una prenotazione non siano anteriori alla data o all'orario di inizio della prenotazione. In questo caso la funzione utilizzata sarà:

```
public void checkCoerenzaDate(Prenotazione thePrenotazione){
            Date dataInizio = thePrenotazione.getDateInizio();
            Date dataFine = thePrenotazione.getDateFine();
            if ((dataInizio.compareTo(dataFine)) > 0) {
                  UserMessages userMessages = new UserMessages();
                  UserMessage msg = new UserMessage("message.ErrDate");
                  userMessages.add(msg);
                  throw new BusinessException(userMessages);
            }
            try{
                  if ((dataInizio.compareTo(dataFine)) == 0) {
```

```
                        // Controllo dell'ora
                        String oraIn = thePrenotazione.getOraInizio();
                        String oraFin = thePrenotazione.getOraFine();
                        SimpleDateFormat sdfDaTogliere = new
                                        SimpleDateFormat("HH:mm:ss");
                        if (oraIn != null && oraFin != null) {
                              Date oraInDate = sdfDaTogliere.parse(oraIn);
                              Date orafineDate = sdfDaTogliere.parse(oraFin);
                              if ((oraInDate.compareTo(orafineDate)) >= 0) {
                                    UserMessages userMessages = new
                                                  UserMessages();
                                    UserMessage msg = new
                                                UserMessage("message.ErrOra");
                                    userMessages.add(msg);
                                    throw new
BusinessException(userMessages);
                              }
                        }
                  }
            }catch(ParseException ex){
                  throw new RuntimeException("Ora inizio e fine errata");

            }

      }
```

Tale implementazione sarà riportata nel file PrenotazioneServiceImpl.java, mentre la sua dichiarazione risiederà in PrenotazioneServiceDao.java. Infine, questa funzione di controllo sarà richiamata nel momento in cui verranno immessi i dati per una nuova prenotazione, quindi verrà richiamata nel file *ManagePrenotazioneController.java*, all'interno dell'operazione elementare *saveNewElement*.

5.9 Attività di manutenzione

Quando si decide di adoperare un generatore di codice, come SkyGen, per la realizzazione di una web application, l'approccio utilizzato risulta essere opposto a quello che, invece, si utilizzerebbe nel caso in cui l'applicazione sia realizzata senza l'utilizzo di un generatore. L'approccio di cui si sta parlando è il bottom-up. Volendo adoperare, infatti, SkyGen, l'applicazione viene anzitutto pensata sotto il profilo di quello che sarà il suo database, quali saranno, quindi, le tabelle (o entità) che l'andranno a comporre, ed in base a tale idea si disegna il modello UML, ovvero un diagramma delle classi di dominio.

Nel caso, invece, si voglia scrivere il codice dell'applicazione manualmente verrà adottando un procedimento che spesso viene definito top-down. Si definiranno, dunque, le funzionalità dell'applicazione, chiamate a user story, si implementeranno le funzionalità dell'applicazione partendo dagli strati di più alto livello, ovvero quelli visibili all'utente, scendendo mano a mano nel "core" del sistema e si eseguiranno di volta in volta dei test per la correzione di eventuali bug.

Da ciò è facile dedurre come in quest'ultima situazione le attività di manutenzione siano effettuate ogni qual volta venga aggiunto un nuovo pezzo di codice al progetto, soprattutto se il metodi di progettazione utilizzato è quello agile. Nel caso, invece, si utilizza un generatore di codice, la fase di manutenzione è rimandata in seguito alla prima generazione, questo perché non vi è alcuna linea di codice prima della generazione.

Dando in input allo strumento SkyGen il diagramma UML dell'applicazione, viene restituito il codice dell'applicazione e l'applicazione risulta già essere perfettamente funzionante. È naturale che questa prima versione dell'applicazione abbia bisogno di modifiche affinché tutti i requisiti, preventivamente analizzati, siano rispettati e siano implementati tutti i vincoli tra le varie classi del diagramma.

Per rendere l'applicazione conforme a vincoli e requisiti si ha, dunque, il bisogno di modificare il codice. La modifica può avvenire in modo manuale, soprattutto se si tratta di un'attività di manutenzione evolutiva, ad esempio l'aggiunta di nuove funzionalità, oppure si realizza mediante una rigenerazione. Questo secondo caso si concretizza quando si deve modificare un aspetto del diagramma UML disegnato inizialmente.

Si prenda ad esempio l'applicazione Space ClassRoom realizzata con SkyGen. Inizialmente il diagramma UML utilizzato per la generazione prevedeva che l'entità Aula avesse un attributo, "AulaId", come identificativo univoco dell'aula, e che fosse di tipo Integer. In realtà, ogni aula del dipartimento ha un nome proprio alfanumerico che la identifica, era per questo necessaria la seguente modifica: da Integer, l'attributo "AulaId" doveva diventare String. Per realizzare questa modifica si è semplicemente ritoccato il diagramma delle classi, trasformando il tipo dell'attributo in questione, e dato in input a

SkyGen. L'output di SkyGen è stato ancora una volta l'intero codice sorgente dell'applicazione, che risultava, però, diverso dal primo per la modifica apportata. È lecito, a questo punto, domandarsi se le customizzazioni fatte sulla prima generazione siano andate perdute con la seconda generazione. In realtà un vantaggio che viene spesso sottovalutato in un generatore di codice è la presenza di un repository. In esso sono comunque conservate tutte le versione del codice. In particolare, dopo una generazione bisognerà collegarsi al repository e scaricare la versione più recente in esso contenuta. Nello scaricare però, l'ambiente di lavoro compara la nuova versione con la precedente e mostra al programmatore i punti in cui le due versioni divergono, in modo tale che sia il programmatore a scegliere quali linee di codice andranno a comporre il progetto e quali invece non devono più farne parte. Così facendo, non solo si salvano le modifiche apportate al diagramma UML, ma anche le customizzazioni fatte prima di una nuova generazione.

SkyGen, ha però un vantaggio di fondamentale importanza, oltre a poter permettere l'operazione di merge tra più generazioni, permette di modificare manualmente qualsiasi linea di codice essendo sostanzialmente linguaggio Java quello generato. La modifica appena vista, infatti, si sarebbe potuta concretizzare anche senza una nuova rigenerazione, ma attraverso la scrittura di codice manuale. Dunque, qualsiasi modifica può essere fatta manualmente, il codice generato è semplice codice per web application, qualsiasi programmatore con la preparazioni adeguata potrà metterci mano e modificare qualsiasi suo aspetto, dalla parte grafica fino al parte del database. L'unico motivo per cui si preferisce una rigenerazione nel caso di modifiche su database è la difficoltà di farle manualmente. La rigenerazione è semplice e non costa nulla, una modifica manuale potrebbe richiedere troppo tempo e un notevole sforzo da parte del programmatore. La relazione tempo/forza lavoro in un'azienda informatica attualmente è di fondamentale importanza per poter rispondere in modo positivo e in tempo breve alle esigenze di mercato.

Ritornando al discorso delle modifiche, l'ultimo aspetto che ci resta da esaminare è l'aggiunga di nuove funzionalità per una manutenzione evolutiva. In questo caso, se l'applicazione presenta delle discordanze con i requisiti analizzati e specificati in un primo momento, dovrà essere modificata affinché tutti i requisiti richiesti esplicitamente dal cliente siano soddisfatti e, quindi, se vi è bisogno di aggiungere nuove funzionalità ciò verrà fatto principalmente manualmente, sempre che non si debba modificare il database.

Infatti, se al codice manca sostanzialmente una funzione conviene inserirla manualmente, non è detto, infatti che la funzione possa essere generata, in quanto se non è una funzione esprimibile nel diagramma delle classi, input di SkyGen, non sarebbe possibile la sua generazione. In caso contrario, alla scrittura manuale di codice conviene la rigenerazione.

Un ultimo caso da esaminare è l'aggiunta o l'eliminazione di elementi dal diagramma UML iniziale. Supponiamo di voler eliminare da esso una funzionalità, in questo caso sarà possibile effettuare tale modifica e per concretizzarla bisogna utilizzare nuovamente SkyGen. La nuova generazione risulterà essere completamente disaccoppiata dalla precedente, questo il generatore lavora in modo disaccoppiato su ogni componente che dovrà trasformare in codice sorgente. Lo stesso discorso vale nel momento in cui anziché eliminare una funzionalità ne viene aggiunta una nuova. Questo aspetto di SkyGen permette di effettuare una qualsiasi modifica al modello inizialmente pensato e di ricreare ogni volta un' applicazione funzionante fin da subito.

Capitolo 6: Analisi di qualità

Nello sviluppo tecnico delle applicazioni web e mobile la scelta del framework e della modalità di sviluppo del progetto, che si adatta meglio alle esigenze di mercato del cliente, è un elemento critico che deve essere valutato con grande attenzione visti gli impatti sul business nel medio e breve termine. Quando si parla di framework, s'intende un insieme di librerie sulle quali un software può essere progettato, realizzato e distribuito su più piattaforme scrivendo il codice sorgente una sola volta. Il compito del framework è, infatti, quello di ricompilare il codice in modo che si adatti automaticamente a sistemi operativi diversi. Oggi sul mercato esistono diversi tipi di framework, ognuno con caratteristiche e vantaggi differenti. Per questo la scelta del framework giusto deve essere oggetto di una attenta valutazione da parte delle aziende. Questa valutazione deve prendere in considerazione numerosi parametri quali, ad esempio, le funzionalità che il cliente intende integrare nell'applicazione, i device sui quali questa applicazione deve girare, il budget a disposizione, la deadline per commercializzare l'applicazione sul mercato ed infine la strategia di lungo termine legata alla quantità di progetti che devono essere sviluppati. Il vantaggio principale nell'utilizzo di un framework consiste nella possibilità di effettuare un delivery rapido e multipiattaforma di un'applicazione: il codice, infatti, si scrive velocemente con un meta linguaggio ed attraverso librerie esistenti precompilate, ed esso viene automaticamente adattato per i diversi sistemi operativi in fase di compilazione. Una criticità di questo tipo di framework consiste nel fatto che, all'inizio, l'azienda deve sostenere costi piuttosto elevati per l'acquisto della licenza. Occorre inoltre

investire anche nelle attività di formazione per le risorse interne che dovranno occuparsi dell'implementazione e dell'uso del framework. Un limite ulteriore dei framework consiste nell'eccesso di rigidità in fase progettuale: il loro utilizzo, infatti, limita in parte la possibilità di sfruttare le caratteristiche native del device, delle linee guida dell'interfaccia e del sistema operativo che utilizzerà l'applicazione, e spesso, quindi, si corre il rischio di non riuscire a sviluppare interfacce utente pienamente ergonomiche e intuitive. Nella scelta del framework occorre anche concentrarsi su quelli che offrono una maggiore garanzia di supporto nel corso del tempo. Alcuni framework, infatti, tendono a non essere più supportati dopo un certo periodo di tempo; in questo caso, se occorre implementare sviluppi ulteriori, si corre il rischio di dover riprogettare l'applicazione da zero utilizzando un framework diverso. Nel caso di framework nativi i costi iniziali sono più bassi rispetto ai precedenti (non vi è, ad esempio, la necessità di acquistare la licenza per il loro utilizzo), tuttavia i costi tendono a crescere esponenzialmente in funzione della quantità di aggiornamenti e rilasci da effettuare nel corso del tempo, o nel caso in cui si intenda sviluppare un'applicazione per più sistemi operativi diversi tra loro. Un tipo di framework scelto nel contesto giusto, che tenga concretamente conto degli obiettivi di marketing dell'azienda e le strategie di investimento in sviluppo di prodotti, consente di ottimizzare i tempi di sviluppo, presentare velocemente sul mercato la propria applicazione ed ottimizzare il processo di delivery abbattendo costi e sfruttando le potenzialità specifiche del framework selezionato.

L'azienda SkyIT, estremamente attenta ai cambiamenti del mercato dell' *Information Technology*, offre la realizzazione, in tempi estremamente brevi, di soluzioni efficaci, complesse e funzionali e opportunamente documentate. Sky* è la suite di prodotti nativi che nasce con l'intento di soddisfare queste esigenze. Tra i prodotti quello utilizzato durante lo sviluppo dell'applicazione Space Classroom, descritta nel capitolo precedente, è SkyGen. SkyGen è un "Application Framework" e un "Code Generation", indubbiamente il componente più importante dell'intera suite di sviluppo, basato, come già ampiamente descritto, sul principio del Model Driven Development (MDD), consentendo di

trasformare modelli UML in artefatti. Grazie a questa soluzione le strutture prodotte riescono a registrare produttività elevatissime nello sviluppo di soluzioni software su tecnologia JEE ad alto contenuto tecnologico e funzionale.

6.1 La Qualità

Secondo la definizione data da Pressman, la qualità di un software può essere definita come la sua conformità ai requisiti funzionali e prestazionali enunciati esplicitamente, agli standard di sviluppo esplicitamente documentati, e alle caratteristiche implicite, che è lecito aspettarsi da un prodotto professionale.

Quindi, volendo schematizzare, sono tre i punti fondamentali nell'analisi di qualità del software:

1. I requisiti sono il fondamento su cui misurare la qualità.
2. Gli standard di qualità definiscono i criteri da seguire nello sviluppo software.
3. Esistono requisiti impliciti la cui assenza compromette la qualità del software.

Esistono, però, dei problemi nella definizione di qualità introdotta, ovvero che le specifiche software sono in genere incomplete e spesso inconsistenti, che alcuni requisiti di qualità sono difficili da specificare in maniera non ambigua, che può esistere contrapposizione fra i requisiti di qualità attesi dal cliente (efficienza, affidabilità, etc.) e quelli dello sviluppatore (manutenibilità, riusabilità, etc.), e, infine, che alcuni requisiti di qualità sono difficili da valutare, non è, infatti, possibile valutarli direttamente, ma soltanto indirettamente. Dunque, non si può valutare la qualità del software in maniera assoluta, ma solo attraverso alcune sue manifestazioni. Ciò equivale a dire che è impossibile valutare direttamente la qualità del software, ma solo indirettamente, attraverso la valutazione di attributi che si correlano a questa, supponendo, però, che la relazione tra la qualità e questi attributi sia valida.

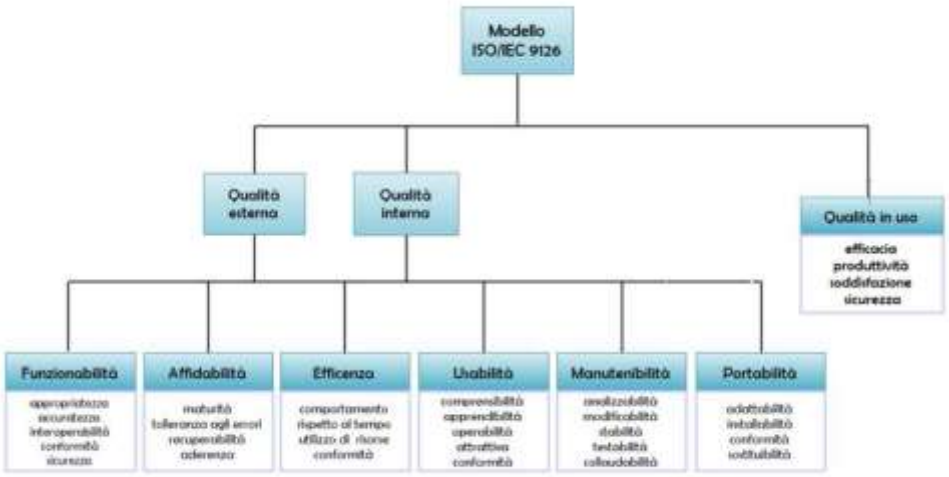
La qualità di un prodotto software si caratterizza attraverso un insieme finito e definito di attributi, che devono essere ragionevolmente esaustivi, in modo che per una qualsiasi richiesta di caratteristica di qualità sia possibile associarvi un sottoinsieme degli attributi

definiti in modo da poterla valutare, e privi di reciproche sovrapposizioni, per evitare che più attributi riguardino la stessa caratteristica del software.

Il Modello di Qualità del software è un insieme di attributi del software che fornisce uno schema di riferimento che, con un'opportuna distribuzione di pesi per ciascun attributo, va adeguato e tarato per la rappresentazione dei requisiti di qualità desiderati dal committente o posseduti dal software. Lo standard più utilizzato per la descrizione di un modello di qualità del software è lo standard ISO/IEC 9126. Esso è composto di quattro parti:

1. Quality Model: un insieme di caratteristiche di qualità che descrivono i fattori di qualità di un prodotto. Sono i fattori visti nel modello di Mc Call.
2. External Metrics: un insieme di metriche indirette attraverso le quali si può valutare la conformità di un prodotto in relazione all'ambiente operativo in cui si trova.
3. Internal Metrics: un insieme di metriche applicabili al software non eseguibile (come il codice sorgente), legate alle specifiche richieste dall'utente, che permettono di individuare eventuali problemi nelle metriche esterne, prima che il software sia eseguibile.
4. Quality In Use Metrics: un insieme di metriche, dipendenti dalle metriche esterne ed interne, legate alle caratteristiche positive che un utente riscontra nell'utilizzo del software.

Per determinare la qualità di un prodotto software si possono quindi osservare i tre punti di vista: qualità esterna, qualità interna e qualità in uso. Essi si influenzano a vicenda, ma è ovvio che un prodotto software percepito positivamente dall'utente è sintomo di una buona qualità di base del codice sorgente. Una rappresentazione di questo modello, ci può essere data dal seguente grafico:



6.2 Metriche Qualitative e Quantitative

Vi sono quattro motivi che spingono a misurare il processo ed il progetto software:

- *Caratterizzare*. Si caratterizza per acquisire conoscenze sui processi, sui prodotti, sulle risorse e sugli ambienti e per stabilire indicazioni di base per il confronto con i prossimi lavori.
- *Valutare*. Si valuta per determinare lo stato rispetto ai piani. Le misurazioni sono i sensori che ci fanno capire quando i progetti e i processi sono fuori controllo, in modo da poter applicare misure correttive. Inoltre, si valuta per stabilire il raggiungimento degli obiettivi di qualità e per stabilire l'impatto dei miglioramenti tecnologici e di processo sui prodotti e sui processi stessi.
- *Prevedere*. Si prevede per pianificare. La misura per scopi di previsione prevede l'acquisizione di conoscenze sulle relazioni fra processi e prodotti e la creazione di modelli per la realizzazione di tali relazioni in modo che i valori osservati per determinati attributi possano essere utilizzati per prevederne altri. Ci si comporta in questo modo perché si vogliono stabilire degli obiettivi raggiungibili nell'ambito dei costi, dei tempi e della qualità in modo da poter applicare le risorse appropriate. Le misure di previsione fungono anche da base per determinare le tendenze in modo da poter stimare i costi, i tempi e la qualità sulla base dei valori attuali. Le

proiezioni e le stime basate su dati storici aiutano anche ad analizzare i rischi e a seguire valutazioni sul progetto e sul costo.

- *Migliorare*. Si misura per ottenere un miglioramento quando si raccolgono informazioni quantitative che aiutano a identificare i punti critici, le inefficienze e le altre opportunità di miglioramento della qualità del prodotto e delle prestazioni di un processo.

Anche se i termini misura, misurazione e metrica sono spesso considerati sinonimi, e importante sottolineare le sottili differenze:

- Misura: fornisce una misurazione quantitativa del grado, dell'ammontare, della dimensione e della capacità di un attributo di un prodotto o di un processo. Le misure possono essere dirette (righe di codice prodotte velocità di esecuzione, memoria occupata, difetti rilevati in un dato intervallo temporale, ecc.), oppure indirette (funzionalità, qualità, complessità, efficienza, affidabilità e facilità di manutenzione).
- Misurazione: è l'atto di determinare la misura.
- Metrica: mette in relazione, secondo certi criteri, le singole misure.
- Indicatore: è una combinazione di metriche che mette in risalto determinate caratteristiche di un processo, di un progetto o di un prodotto.

Il difetto delle metriche software è che esse non misurano qualcosa di tangibile, e per questo non vi è accordo su cosa si deve misurare, come misurarlo e quali metriche ed indicatori adottare. Esistono tantissime metriche generali che ci permettono di capire quali sono i punti su cui focalizzare la nostra attenzione per confrontare la qualità del codice sorgente. Possiamo dividere le principali metriche in cinque categorie secondo le caratteristiche su cui esse agiscono: Size, Tests, Duplication, Design e Rules.

- **Metriche "Size"**
    - Physical lines: numero di ritorni a capo. Deve essere un valore basso.
    - Comment lines: numero di linee di commento. Deve essere un valore basso.

- Lines of code Numero di linee di codice: il progetto che possiede meno righe di codice ha un design superiore e richiede una manutenzione minore.
- Packages: numero di pacchetti.
- Classes Numero di classi: confrontando progetti con le stesse funzionalità, il progetto che possiede più classi è quello che realizza l'astrazione migliore.
- Files: numero di files analizzati.
- Directories: numero di directories analizzate.
- Methods: numero di metodi.

– **Metriche "Tests"**

- Unit tests: numero di test.
- Unit tests duration: tempo impiegato per effettuare un test.
- Unit test error: numero di test falliti.
- Line coverage: linee coperte dal test.
- Branch coverage: rami di un'espressione booleana coperti(true e false).

– **Metriche "Duplication"**

- Duplicated lines: numero di linee duplicate.

– **Metriche "Rules"**

- Violations: numero di regole violate
- Weighted violations: somma delle violazioni calcolate in base ad un coefficiente di priorità
- Indice di conformità: è calcolato come [100 - (weighted_violations / nloc * 100)]

– **Metriche "Design"**

- Depth of inheritance tree: è la posizione della classe all'interno dell'albero di eredità. Si consiglia di mantenere il numero dei livelli da attraversare per giungere la radice, al di sotto del 5.
- Response for class: è il numero di tutti i metodi implementati da una classe più il numero di metodi accessibili tramite un oggetto della classe. Si

consiglia di mantenere questo numero basso. Maggiore è la RFC, maggiore è lo sforzo dato per apportare modifiche.

- Afferent couplings: è il numero delle classi che utilizzano una classe.
- Efferent couplings: è il numero delle classi che una classe utilizza.
- Lack of cohesion of methods: essa misura il numero di componenti collegati ad una classe. Un valore basso indica che il codice è semplice da comprendere e riutilizzare. Un valore alto suggerisce di dividere la classe in classi più piccole. Si consiglia di effettuare un refactoring, se questo valore diventa maggiore di 2.
- Weighted methods per class: è il numero medio di metodi contenuti in una classe. Si consiglia di mantenere tale numero al di sotto del valore 14.
- Cyclomatic Complexity (CC): essa misura il numero di cammini linearmente indipendenti all'interno di una parte di codice. Più il numero è alto, più lo sforzo per effettuare un testing del codice sarà alto. Si consiglia di mantenere questo numero al di sotto del valore 10.
- Distance: rappresenta la distanza dalla linea ideale. Identifica quanto una categoria è lontana dal caso ideale. Minore è la distanza del software dalla linea, maggiore è la sua qualità.

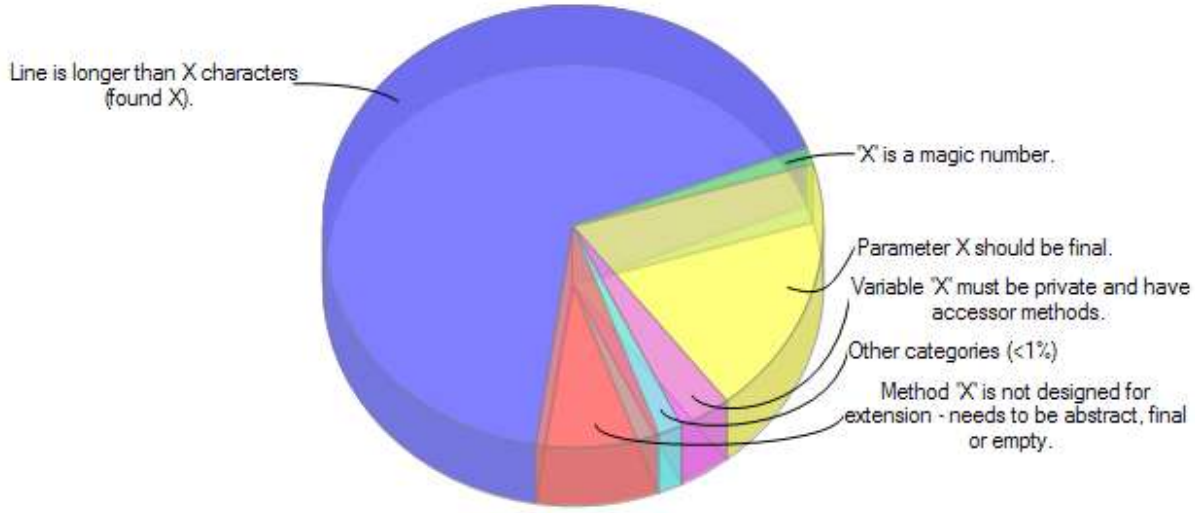
6.3 Qualità del codice

Un importante requisito di progetto di un prodotto software è la manutenibilità, in quanto definisce la capacità del prodotto di essere facilmente ripristinato qualora in futuro sia necessario un intervento di manutenzione. La manutenibilità rappresenta, quindi, un requisito indispensabile del sistema per ottimizzare l'implementazione delle attività manutentive. Per poter, però, effettuare un'operazione di manutenzione in futuro è di fondamentale importanza che la complessità del codice sia bassa, in caso contrario qualsiasi modifica potrebbe risultare difficile da effettuare.

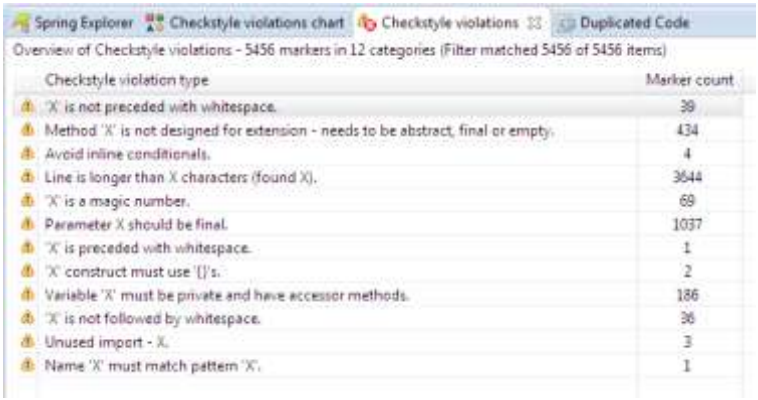
Esistono diversi strumenti open source in gradi di effettuare una scansione del codice e di individuare gli errori in maniera automatica con un elevato livello di fiducia. Essi consentono, attraverso numerosi strumenti, di analizzare il codice in base ad un vasto numero di metriche e di conseguenza garantiscono una copertura quasi totale degli aspetti principali riguardanti il grado di qualità del codice. Nella maggior parte dei casi essi vengono utilizzati come supporto all'analista per individuare delle falle all'interno del codice ed indirizzarlo verso modifiche appropriate. Alcuni di questi strumenti agiscono all'interno degli IDE (ambienti di sviluppo integrato), il che permette di effettuare l'analisi durante la fase di sviluppo. Questo è effettivamente il momento migliore per consentire al team di tornare indietro (azione di feedback) e aggiustare parti di codice che potrebbero, nello sviluppo futuro, provocare danni irreparabili. Tra tutti gli strumenti messi a disposizione, quello che utilizzeremo per l'analisi è Checkstyle, che sarà utilizzato come plug-in dell'ambiente di sviluppo Eclipse.

Checkstyle è uno strumento di sviluppo per aiutare i programmatori a scrivere codice Java che aderisce ad uno standard di codifica. Si automatizza il processo di controllo del codice Java per risparmiare gli esseri umani di questo noioso, ma importante compito. Questo lo rende ideale per i progetti che vogliono imporre uno standard di codifica. Checkstyle è, inoltre, altamente configurabile e può essere fatto per supportare quasi qualsiasi standard di codifica. Un esempio di file di configurazione viene fornito a supporto di Sun Code Conventions. Come pure, altri file di configurazione di esempio vengono forniti per altre convenzioni ben note.

Checkstyle, quindi, è un tool utilizzato per garantire che il codice Java aderisce a una serie di standard di codifica. Analizzando il codice dell'applicazione web realizzata, Space Classroom, con un altro strumento, alcune caratteristiche cambiano. Si noti, infatti, che Checkstyle, individua una serie di violazioni che non è possibile riscontrare con altri strumenti. Questo perchè Checkstyle aderisce a delle regole di codifica diverse e molto rigide.



L'analisi condotta da Checkstyle ha individuato nell'applicazione Space Classroom, composta in totale da 34477 linee di codice, 5456 violazione, suddivise in 12 categorie:


Spring Explorer | Checkstyle violations chart | Checkstyle violations | Duplicated Code

Overview of Checkstyle violations - 5456 markers in 12 categories (Filter matched 5456 of 5456 items)

Checkstyle violation type	Marker count
'X' is not preceded with whitespace.	39
Method 'X' is not designed for extension - needs to be abstract, final or empty.	434
Avoid inline conditionals.	4
Line is longer than X characters (found X).	3644
'X' is a magic number.	69
Parameter X should be final.	1037
'X' is preceded with whitespace.	1
'X' construct must use '{}'s.	2
Variable 'X' must be private and have accessor methods.	186
'X' is not followed by whitespace.	36
Unused import - X.	3
Name 'X' must match pattern 'X'.	1

Checkstyle ci fa notare che nel progetto in esame sono presenti linee di codice con più di 80 caratteri, o ancora che sono presenti dei numeri, che Checkstyle definisce "magic number", che non sono stati definiti come costanti. Si capisce, quindi, che la maggior parte di queste violazioni guarda più alla forma del progetto che alla qualità, infatti il 66,8% delle violazioni si ha con linee di codice con più di 80 caratteri. Checkstyle cerca di rendere il codice sorgente il più chiaro possibile, ma nella maggior parte dei casi le violazioni individuate vengono sorvolate. D'altronde in un progetto di vasta entità sarebbe inutile concentrarsi sul numero di caratteri che ogni linea possiede, piuttosto che su metriche che inficiano in modo maggiormente negativo sulla qualità del codice sorgente.

6.4 Qualità della progettazione

Negli ultimi anni si è osservato come sempre più aziende hanno riconosciuto l'importanza della qualità del software, proprio per questo il test è stato ampiamente accettato come parte fondamentale e integrante dello sviluppo. Tuttavia, la fase di test è solo un passo per arrivare a ottenere prodotti software di alta qualità. Altrettanto importante è, infatti, il bisogno di validare continuamente il software rispetto ai principi di progettazione consolidati, contribuendo, così, a migliorare la manutenibilità, la flessibilità e la verificabilità.

È un dato di fatto, molti progetti falliscono per mancanza di qualità del software. Pertanto, avere un occhio sulla qualità del codice non è solo un'opzione, ma può rivelarsi un impegno gravoso. Inoltre, non è utile effettuare il controllo di qualità solo alla fine del processo di progettazione software. Dal momento che una diagnosi precoce sulla qualità rende eventuali problemi facili da risolvere, il controllo della qualità deve essere effettuato il più annesso possibile alla fase di sviluppo.

Quando si cercano soluzioni per facilitare lo sviluppo, la progettazione, il controllo e il miglioramento della qualità, vale la pena considerare l'integrazione di strumenti in grado di effettuare un'analisi della struttura. È questo l'ambito in cui entra in gioco STAN, uno strumento di analisi della struttura per il linguaggio Java, esso permette allo sviluppatore di avere nelle proprie mani la garanzia della qualità del software con il minimo sforzo possibile.

STAN è il principale strumento, basato su Eclipse, che permette di effettuare l'analisi della struttura del codice Java, esso riunisce lo sviluppo e la garanzia della qualità in modo naturale.

Tale strumento incoraggia gli sviluppatori e i project manager nel visualizzare il loro design, nella comprensione del codice, nella misurazione della qualità ed, infine, nel reporting dei difetti di progettazione. STAN supporta un insieme di parametri accuratamente selezionati, ed è adatto a coprire gli aspetti più importanti della qualità strutturale. Particolare attenzione è stata data all'analisi delle dipendenze visiva, una

chiave per l'analisi della struttura. STAN si integra perfettamente nel processo di sviluppo. In questo modo, gli sviluppatori possono prendersi cura della qualità del design fin dall'inizio. I project manager, infatti, utilizzano STAN come strumento di monitoraggio e di reporting.

Fino a quando un progetto è piccolo, gli sviluppatori hanno in mente un'immagine della loro disegno, conoscono ogni angolo del loro codice e tutto sembra essere sotto controllo. Come la dimensione del progetto si evolve, però, le cose cambiano: improvvisamente il software è difficile da testare. Robert C. Martin descrive tale situazione con la seguente affermazione: "Il software inizia a marcire come un pezzo di carne cattiva". Inoltre, egli individua le seguenti caratteristiche:

- *Rigidità*. Il sistema è difficile da cambiare perché ogni modifica impone molti altri cambiamenti.
- *Fragilità*. I cambiamenti portano il sistema a "rompersi" in luoghi concettualmente indipendenti.
- *Immobilità*. E 'difficile separare il sistema in componenti riutilizzabili.
- *Viscosità*. Fare le cose bene è più difficile che fare le cose sbagliate.
- *Opacità*. E 'difficile da leggere e capire. Esso non esprime bene il suo intento.

Al contrario, una buona progettazione risulta essere flessibile, solida, mobile, fluida e trasparente. La struttura di grandi quantità di codice tende a diventare molto complessa, e i sistemi più complessi sono i più difficili da capire e da gestire, e quindi spesso si "rompono". Mantenere la complessità ad un livello gestibile è una sfida.

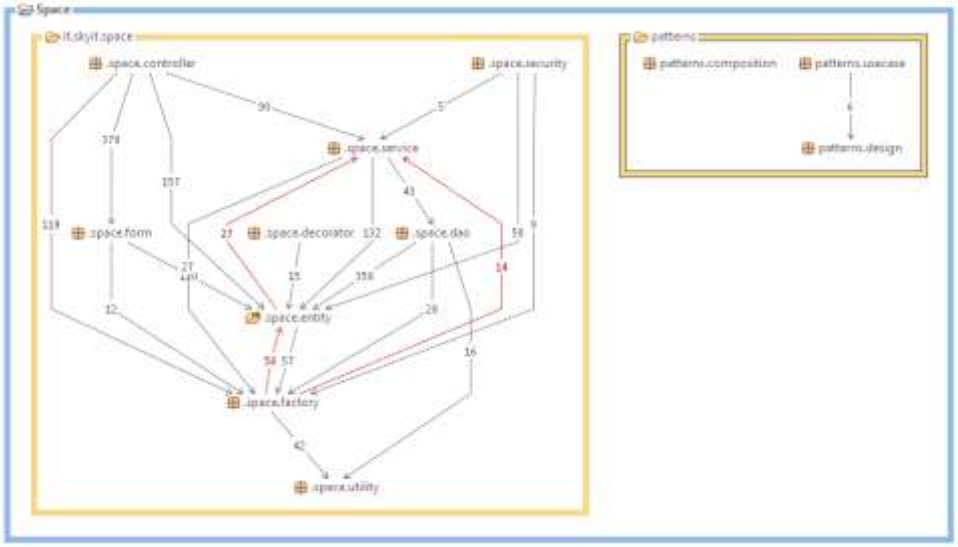
Certamente si fallisce nel momento in cui la struttura software si evolve in modo arbitrario, è quindi essenziale mantenere viva l'attenzione su questo aspetto e fortunatamente, ci sono dei principi di progettazione che possono aiutare ad avere successo. Per raggiungere l'obiettivo preposto bisogna, però, controllare continuamente la nostra struttura secondo tali principi, in modo tale da scoprire e correggere le violazioni di design subito, prima che il software comincia a marcire.

Un ulteriore aspetto importante da considerare è la visualizzazione, una foto vale più di mille parole. Un buono strumento di analisi di una struttura presenta e visualizza la progettazione strutturale in un modo che sia facile da comprendere per tutti. In particolare, quando si tratta di dipendenze, un buono strumento di analisi ha bisogno di funzionalità di layout grafico avanzate, affinché possa creare grafici di dipendenza ordinatamente disposti, ed in modo tale che l'utente possa essere in grado di navigare attraverso questi grafici, per visualizzare in dettaglio. Un buono strumento di analisi deve sostenere, infine, numerose metriche. Per ogni manufatto, dovrebbero essere elencate e classificate tutte le violazioni metriche, permettendo all'utente, così, di distinguere le questioni importanti da quelle trascurabili.

Analizzando il codice Java, il tool STAN è in grado di raccoglie tutte le informazioni necessarie per costruire un modello dettagliato della struttura dell'applicazione. Il livello di dettaglio scelto, "member" o "class", consente di specificare se il modello include lo strato di membro con tutti i campi e i metodi delle classi o se deve essere limitata allo strato di classe.

Dunque, Stan4j permette di effettuare un'analisi visiva del progetto nella sua interezza, e di esplorarne i vari moduli e pacchetti presenti per vedere di cosa essi sono composti e quali e quante dipendenze possiedono. Di seguito sono riportati tutti gli aspetti esaminati dell'applicazione Space Classroom attraverso l'utilizzo del tool Stan4j:

- Composition View



La Composition View di questo progetto ci mostra che esso è composto da una serie di pacchetti, quali:

1. .space.controller
2. .space.security
3. .space.service
4. .space.form
5. .space.decorator
6. .space.dao
7. .space.entity
8. .space.factory
9. .space.utility

Essi sono contenuti all'interno dell'unica cartella it.skyit.space. Ogni package, inoltre, può essere espanso per analizzare la sua struttura interna e le dipendenze fra i suoi vari componenti. Per esempio, volendo estendere il pacchetto space.decorator si può visualizzare la classe, PrenotazioneDecorator, che compone il package selezionato:

- Coupling View

La Couplings View permette di analizzare un singolo elemento e di leggerne le dipendenze in entrata e in uscita. I numeri sulle linee esprimono il numero di volte che una classe ne richiama un'altra, indicata con la freccetta.

Consideriamo, ad esempio, il package space.dao, ed esaminiamo il singolo elemento PrenotazionePersonaleDaoImpl, le sue dipendenze sono riportate di seguito:

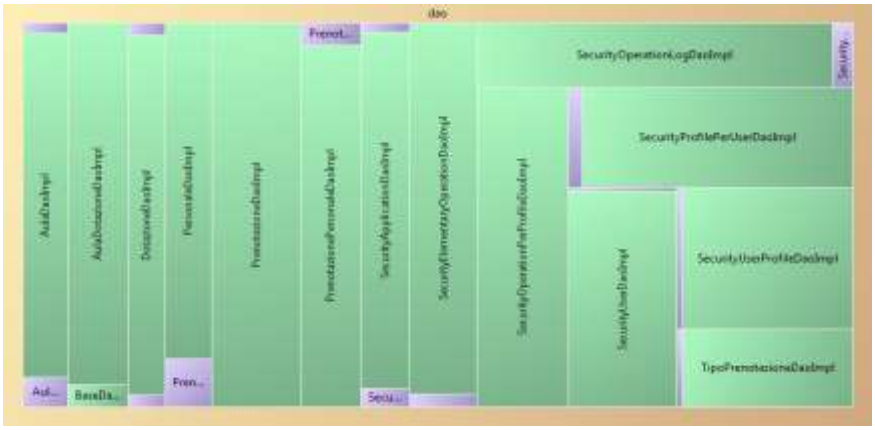
Com'è possibile vedere la classe PrenotazionePersonaleDaoImpl richiama ben 4 volte la classe PersonaleDaoImpl, che a sua volta richiama classi e interfacce appartenenti allo stesso suo package, oppure ad altri presenti nel progetto.

- Map

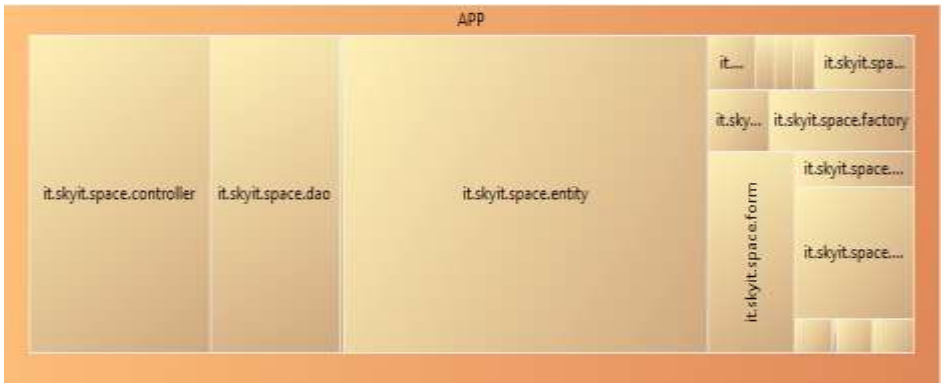
  La mappa ci consente, invece, di individuare le dipendenze attraverso un approccio visivo legato ai colori. Il codice rappresentato da uno dei blocchi, dipende dal codice rappresentato dal blocco sottostante e ha una dipendenza bidirezionale dai blocchi collocati sul suo stesso livello. Selezionando il pacchetto in esame, la divisione sarà:

  - il pacchetto selezionato (color ambra);
  - gli elementi richiesti dal pacchetto selezionato (color verde);
  - gli elementi che dipendono dal pacchetto selezionato (color blu);

In particolare, avendo selezionato il package dao, quindi di color ambra, sono colorati in verde tutte le classi chiamate dal package it.skyit.space.dao, AulaDaoImpl.java, AulaDotazioneDaoImpl.java,BaseEntityDaoImpl.java,DotazioneDaoImpl.java,PersonaleDaoImpl.java,PrenotazioneDaoImpl.java,PrenotazionePersonaleDaoImpl.java,SecurityApplicationDaoImpl.java,SecurityElementaryOperationDaoImpl.java,SecurityOperationLogDaoImpl.java,SecurityOperationPerProfileDaoImpl.java,SecurityOperationPerUserDaoImpl.java,SecurityUserDaoImpl.java,SecurityUserProfileDaoImpl.java,TipoPrenotazioneDaoImpl.java, e di blu gli elementi che dipendono da questi ultimi, come ad esempio AulaDao.java, AulaDotazioneDao.java, DotazioneDao.java, PrenotazioneDao.java, PersonaleDao.java SecurityApplicationDao.java, SecurityOperationLogDao.java, SecurityUserDao.java.



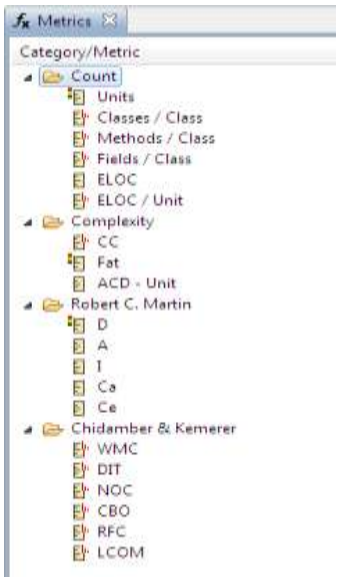
Volendo, invece, considerare l'intera applicazione Space Classroom, la mappa risulta essere la seguente:



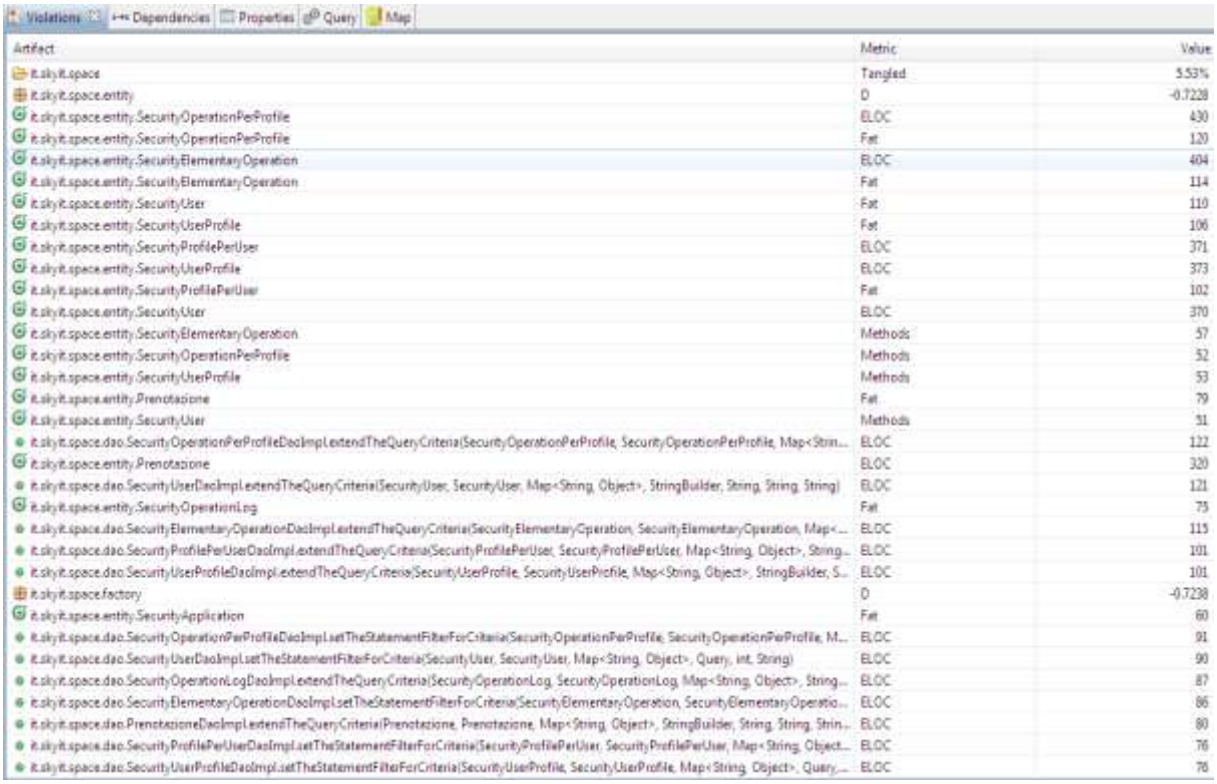
- Metriche

  Stan4j copre un elevato numero di metriche. Le più interessanti per la nostra analisi sono quelle che ci consentono di individuare eventuali violazioni.

  .

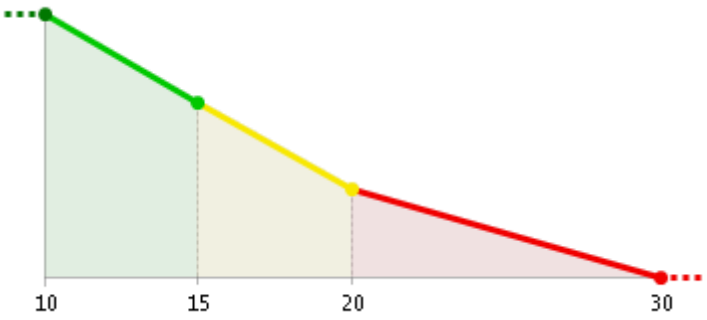


Essendo il progetto molto esteso rispetto al precedente, è presente un elevato numero di violazioni. Il report generato ci mostra le più importanti.



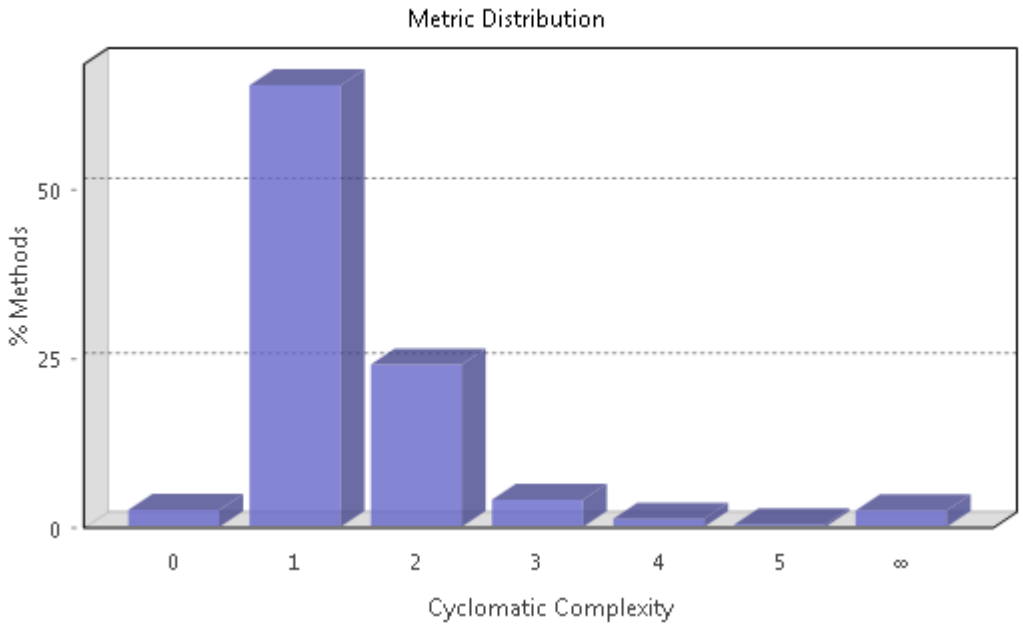
Ci sono due modi per rappresentare graficamente una metrica: il rating e la distribution.

- *Rating*. Il rating individua il numero di righe di codice entro le quali la metrica in esame non altera la qualità del codice sorgente. Per esempio, riguardo la complessità ciclomatica, le linee di codice dal 10 al 15 hanno un valore perfetto (colore verde), mentre la linea 30 rappresenta il caso peggiore (colore rosso).



Con questo intendiamo dire che la complessità ciclomatica, come qualsiasi altra metrica, ha bisogno di essere espressa attraverso dei valori partizionati che ci dicono entro quali soglie essa può essere considerata accettabile e oltre quali soglie è consigliato effettuare un refactoring.

- *Distribution*. Per ogni metrica Stan4j ci mostra anche la distribuzione in base ai vari metodi. Riguardo questo progetto notiamo, per esempio, che la colonna più alta indica che il 65.34% dei metodi ha complessità ciclomatica compresa tra 0 e 1.



- Distance

Un ultimo elemento interessante è rappresentato dalla metrica distance di Robert C.Martin che indica la distanza che il progetto, o i suoi elementi, posseggono dalla retta ideale Instabilità-Astrazione. L'Indipendenza, la Responsabilità e la Stabilità sono solo esempi delle tantissime qualità che un software dovrebbe possedere. Esse mettono in risalto un aspetto molto importante, la dipendenza. Quest'ultima può essere misurata attraverso una serie di metriche che effettuano analisi concrete e calcolabili matematicamente. Le tre principali metriche sono:

- Ca: Afferent Coupling. Numero di classi al di fuori di una categoria che dipendono dalle classi che si trovano all'interno.
- Ce: Efferent Coupling. Numero di classi all'interno di una categoria che dipendono da classi esterne.

- I: Instability: (Ce/(Ca+Ce)). Grado di stabilità di una classe. Questa metrica ha valori compresi nell'intervallo [0,1]. I=0 indica la massima stabilità e I=1 indica la massima instabilità.

Per studiare la dipendenza all'interno di una classe, è importante definire una relazione tra l'Instabilità (I) e l'Astrazione (A). Definiamo l'astrazione come A=Na/Nc, dove Na è il numero di classi astratte e Nc è il numero di classi concrete. Creiamo, quindi, un grafico con A sull'asse delle ordinate e I sull'asse delle ascisse. I due punti che rappresentano le condizioni migliori per una categoria, sono il punto (0,1), che rappresenta massima astrazione e massima stabilità, e il punto (1,0), che rappresenta massima concretezza e massima instabilità. Ovviamente, non tutte le categorie ricadono in questi due punti. Si consideri, ad esempio, una categoria con A=0 e I=0. In tal categoria si hanno massima concretezza e massima stabilità, ma una categoria con queste caratteristiche non è ottimale perché risulta rigida, non può, infatti, essere estesa perché non è astratta e non può essere modificata facilmente a causa della sua stabilità. Si consideri, invece, una categoria con A=1 e I=1. Questo comporta massima astrazione e massima instabilità. Anche questo tipo di categoria non è considerato positivamente in quanto essa diventa rigida. A causa dell'instabilità, infatti, l'astrazione non può essere utilizzata per effettuare un'estensione. Se, infine, si considerasse, invece, una categoria con A=0.5 e I=0.5, essa risulterebbe essere equilibrata. La stabilità, infatti, è in equilibrio con l'astrazione. Una categoria con queste caratteristiche, è parzialmente estendibile essendo parzialmente astratta, e le sue estensioni non sono soggette a instabilità essendo parzialmente stabili. Considerando il grafico A-I, si può tracciare una retta che unisce i punti (0,1) e (1,0). Su tale retta, chiamata "main sequence", ricadranno le categorie in cui c'è equilibrio tra astrazione e stabilità. Le categorie che si trovano sulla "main sequence" non sono né troppo astratte, né troppo instabili. Esse hanno il giusto numero di classi concrete e astratte in proporzione alle loro dipendenze afferenti ed efferenti. Le categorie con caratteristiche ideali dovrebbero trovarsi, quindi, sui punti della "main sequence" che incontrano gli assi, ma poiché questo accade molto raramente, è possibile accontentarsi di trovarle su un punto

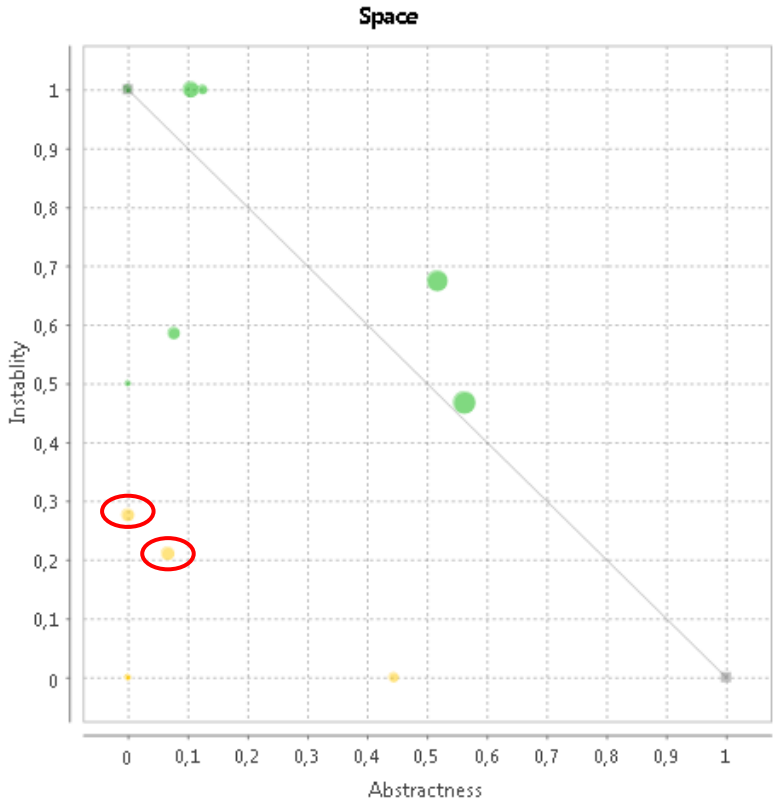
qualsiasi della retta. Quest'ultima riflessione ci permette di esaminare la metrica della distanza. Se è auspicabile per le categorie, di essere proprio sulla "main sequence", o vicino ad essa, si può creare un parametro, D, che misura la distanza di una categoria dalla retta ideale.

D: Distanza: | (A + I-1) ÷ 2 |

D è la distanza perpendicolare di una categoria dalla "main sequence", e il suo valore ricade nell'intervallo[0, ~0.707]. In realtà, è possibile considerare anche il valore normalizzato di D, Dn = | (A + I-1) |, più semplice da interpretare in quanto il suo valore ricade nell' intervallo [0,1].

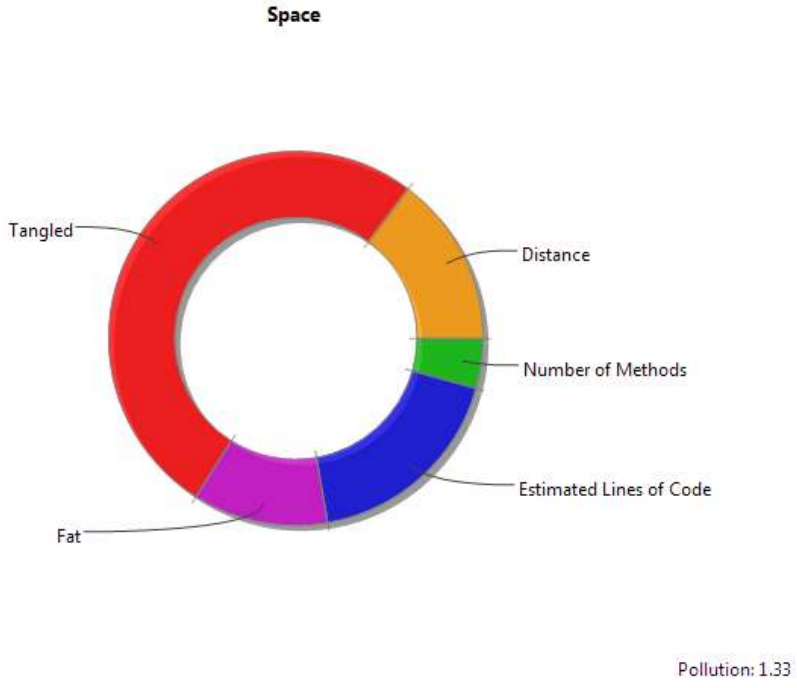
Con la metrica della distanza si può analizzare un progetto in base alla sua conformità generale rispetto alla "main sequence". La metrica D può essere, infatti, facilmente calcolata per ogni categoria, e identifica quanto una categoria è lontana dal caso ideale. Minore è la distanza del software dalla linea, maggiore è la sua qualità.

In questo caso, Stan4j evidenzia ben dieci punti nel codice in cui tale metrica è violata, ma due di essi sono quelli che incidono di più in modo negativo. In ogni caso nessuna di loro è da considerarsi una violazione critica. La prima riguarda il pacchetto 'factory', con un valore di '-0.72' e di coordinate di [0.28,0]. In questo caso essendo entrambi i valori molto vicini allo zero, la violazione intende esprimere l'elevata concretezza e rigidità dell'elemento in esame. Il secondo valore, invece, riguarda il pacchetto 'entity', con un valore di '-0.72' e di coordinate di [0.21,0,07]. Il valore della coordinata riguardante l'astrazione è più piccolo di quello del pacchetto precedente, questo indica un aumento della concretezza e dell'instabilità, che avvicina il comportamento del pacchetto in esame al valore ideale.



- Pollution Chart

La presenza di tutte queste violazioni, ci consente di ottenere la seguente Pollution Chart:



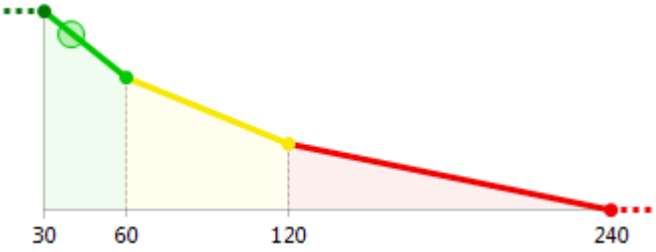
La Pollution Chart ci indica il grado di "inquinamento" del codice e la percentuale in base a questo grado, di tutte le metriche violate. Più è ampio l'anello, più alto è il grado di

inquinamento del codice sottostante. A livello di applicazione, lo spessore dell'anello può, dunque, servire come indicatore rapido per la qualità strutturale complessiva del codice di base. Inoltre, dal Pollution Chart emergono, in particolare due metriche:

- Tangled
- Fat

Tangled. Un tangle è una porzione di un grafico delle dipendenze in cui gli elementi coinvolti sono interdipendenti, presentano cioè una dipendenza ciclica. La presenza dei tangle comporta un aumento della complessità del progetto. I tangles, in Stan4j, sono calcolati in percentuale, e in questo caso il valore violato, associato al package it.skyit.space è di 5,53%. Essendo un valore abbastanza basso si può dedurre che il grafo in esame è un grafo aciclico.

Fat. È una metrica legata al grado di complessità che una libreria, un pacchetto o una classe possiede. In questo caso la figura, ci mostra la complessità delle classi relative alla sottosistema di sicurezza dell'applicazione contenute nel package it.skyit.space.entity ', il cui fat è pari a '40'.



Com'è possibile vedere dal rating, un valore di Fat pari a '40' non altera la qualità del codice sorgente. Dunque, la complessità ciclomatica può essere considerata accettabile e non è per questo richiesto un refactoring.

6.5 Quantità di lavoro risparmiato

Il motivo principale dell'utilizzo di un generatore di codice è quello di evitare la riscrittura di codice utilizzato spesso, inoltre esso fornisce gli strumenti che si occuperanno della realizzazione della struttura portante dell'applicazione, lasciando al programmatore il solo compito di implementarne i contenuti. Qualsiasi sviluppatore si sarà soffermato

almeno una volta a ragionare sull'opportunità o meno di utilizzare uno strumento di tal tipo. Spesso si pensa, come nel caso dell'azienda SkyIT, di crearne uno ad hoc, prendendo le idee migliori, o quelle che più si avvicinavano alla propria concezione di programmazione, dai vari prodotti sperimentati.

I vantaggi apportati dall'utilizzo di un generatore di codice sono molteplici. Tra tutti spicca sicuramente il risparmio di tempo e di codice. Utilizzarlo, infatti, evita la definizione, la progettazione e la riscrittura di codice utilizzato in più progetti o in più parti dello stesso progetto. Un altro vantaggio è dato dalla garanzia di funzionamento del codice. Utilizzare un framework, infatti, consente di risparmiare risorse e tempo per il testing.

Esistono, però, anche degli aspetti negativi. Molti sviluppatori hanno maturato l'idea che l'utilizzo di simili componenti comporta necessariamente di scendere a compromessi di diversa natura, ovvero la quantità delle 'limitazioni' sembra essere inversamente proporzionale alla specializzazione del prodotto in questione, con un andamento tendente a zero,ma che a zero non arriva mai. Anche il più specializzato dei generatori di codice, prevede che ci si affidi ad alcuni paradigmi e filosofie di programmazione.

Resta, quindi, indubbia l'utilità di questi programmi, specialmente in alcuni settori della produzione software, leggasi gestionali, dove le operazioni sono iper-ripetitive. Il programmatore 'puro' e 'purista' resterà sempre un po' sospettoso di fronte a questi Computer Aided Software Engineering, che in definitiva gli tolgono il controllo di alcune parti del codice, a fronte di un discreto risparmio di tempo. Ovviamente non si può sentenziare se è giusto oppure no affidarsi a prodotti del genere, ognuno ha il suo parere, fa le sue riflessioni e soprattutto fa i suoi interessi,ma sicuramente se ne può discutere.

Proprio per cercare di dedurre l'utilità o meno di questa tipologia di strumenti e in particolar modo di SkyGen, analizzeremo il codice prodotto in termini qualitativi e quantitativi. La metrica utilizzata sarà quella relativa al codice sorgente, ed in modo particolare quella delle linee di codice. Quest'ultima è una metrica dimensionale, ed ha l'obiettivo di misurare la lunghezza di un programma attraverso il numero di linee di codice.

Diverse sono le filosofie adottate per effettuare una tal misurazione, in quanto è aperto il dibattito su quali siano le linee di codice che effettivamente devono essere considerate. Alcuni sono concordi sul contare tutte le linee di codice, inclusi i commenti, altri valutano tutte le linee di codice, esclusi i commenti, infine c'è chi considera solo le istruzioni eseguibili includendo anche le istruzioni dichiarative dei dati. La definizione più accettata è, però, la seguente:

"Una linea di codice è ogni linea di testo di un programma che non sia bianca o un commento, indipendentemente dal numero di istruzioni in essa inclusa. In particolare, tali linee di codice possono contenere intestazioni di unità di programma, dichiarazioni ed istruzioni."

Tuttavia, i commenti sono una parte importante del processo di sviluppo di un programma, per cui ignorarle, e quindi non invogliare il programmatore ad inserirle, può essere deleterio; ovviamente vanno prodotti in modo consistente e non inseriti all'ultimo solo per motivi di tariffazione. I commenti possono, quindi, essere valutati in base alla loro importanza.

Il generatore SkyGen, oltre a produrre linee di codice, produce anche i commenti, quindi nell'analisi effettuata viene considerato il codice con e senza commenti. Nello specifico si è pensato di considerare il codice package per package, in modo da valutare più approfonditamente l'utilità di tale strumento, saranno contate, dunque, le linee di codice generate e le linee di codice scritte manualmente per effettuare le customizzazioni necessarie. Di seguito sono schematizzati i dati rilevati nell'analisi riportati per ogni package di ogni progetto.

- Space-core:
    1. it.skyit.space.dao

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
AulaDao.java	41	5	0	0

AulaDotazioneDao.java	63	10	0	0
DotazioneDao.java	41	5	0	0
PersonaleDao.java	41	5	0	0
PrenotazioneDao.java	64	11	10	2
PrenotazionePersonaleDao.java	65	10	0	0
SecurityApplicationDao.java	42	5	0	0
SecurityElementaryOperationDao.java	65	9	0	0
SecurityOperationLogDao.java	55	9	0	0
SecurityOperationPerProfileDao.java	70	12	0	0
SecurityOperationPerUserDao.java	67	10	0	0
SecurityUserDao.java	41	5	0	0
SecurityUserProfileDao.java	54	8	0	0
TipoPrenotazioneDao.java	41	5	0	0
Totale	**750**	**109**	**10**	**2**

Volendo esaminare i dati rilevati dall'analisi fatta, è facile vedere come la percentuale di codice scritto sia solo l'1,3% di tutto il codice del package considerato.

2. It.skyit.space.entity

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
Aula.java	482	210	0	0
AulaDotazione.java	486	216	0	0
BaseEntity.java	23	6	0	0
Dotazione.java	360	157	0	0
Personale.java	455	196	21	21
Prenotazione.java	682	310	42	13
PrenotazionePersonale.java	505	251	26	17
SecurityApplication.java	556	238	0	0
SecurityElementaryOperation.java	929	410	0	0
SecurityOperationLog.java	620	265	0	0
SecurityOperationPerProfile.java	870	390	0	0

SecurityOperationPerUser.java	758	336	0	0
SecurityUser.java	781	340	0	0
SecurityUserProfile.java	808	364	0	0
TipoPrenotazione.java	360	150	0	0
Totale	**8675**	**3839**	**89**	**51**

Nel package it.skyit.space.entity sono state contate 8764 righe di codice in totale, e di queste il solo l'1% è servito per customizzare il codice generato in modo tale da rendere l'applicazione prodotta il più corrispondente possibile ai requisiti iniziali.

3. it.skyit.space.factory

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
AulaFactory.java	77	24	0	0
AulaDotazioneFactory.java	77	24	0	0
DotazioneFactory.java	77	24	0	0
PersonaleFactory.java	77	24	0	0
PrenotazioneFactory.java	77	24	0	0
PrenotazionePersonaleFactory.java	80	25	0	0
SecurityApplicationFactory.java	79	24	0	0
SecurityElementaryOperationFactory.java	81	26	0	0
SecurityOperationLogFactory.java	79	24	0	0
SecurityOperationPerProfileFactory.java	81	26	0	0
SecurityOperationPerUserFactory.java	80	25	0	0
SecurityUserFactory.java	77	24	0	0
SecurityUserProfileFactory.java	79	24	0	0
TipoPrenotazioneFactory.java	79	24	0	0
Totale	**1100**	**342**	**0**	**0**

In tale package, invece, non vi è stato alcun bisogno di inserire codice manualmente, è stato sufficiente l'utilizzo del codice generato da SkyGen.

4. it.skyit.space.service

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
AulaService.java	43	11	0	0
AulaDotazioneService.java	23	6	0	0
BaseEntityService.java	23	5	0	0
DotazioneService.java	23	6	0	0
PersonaleService.java	23	6	15	3
PrenotazioneService.java	43	11	0	0
PrenotazionePersonaleService.java	44	11	0	0
SecurityApplicationService.java	23	6	0	0
SecurityElementaryOperationService.java	34	10	0	0
SecurityOperationLogService.java	35	10	0	0
SecurityOperationPerProfileService.java	48	13	0	0
SecurityOperationPerUserService.java	43	11	0	0
SecurityUserService.java	33	9	0	0
SecurityUserProfileService.java	23	6	0	0
TipoPrenotazioneService.java	23	6	0	0
Totale	**484**	**127**	**15**	**3**

Volendo analizzare i dati rilevati nel package it.skyit.space.service si può facilmente vedere come il 3% del codice costituente tale package sia non generato, ma dovuto a particolari customizzazioni effettuate.

5. it.skyit.space.utility

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
ApplicationProvider.java	105	37	0	0
Totale	**105**	**37**	**0**	**0**

Anche in questo package, costituito da un unico file, è bastato il codice generato dallo strumento SkyGen.

- Space-impl

    1. it.skyit.space.dao

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
AulaDaoImpl.java	189	84	0	0
AulaDotazioneDaoImpl.java	254	125	0	0
BaseEntityDaoImpl.java	24	5	0	0
DotazioneDaoImpl.java	180	75	0	0
PersonaleDaoImpl.java	189	81	0	0
PrenotazioneDaoImpl.java	276	191	115	74
PrenotazionePersonaleDaoImpl.java	265	137	0	0
SecurityApplicationDaoImpl.java	219	113	0	0
SecurityElementaryOperationDaoImpl.java	333	213	0	0
SecurityOperationLogDaoImpl.java	275	157	0	0
SecurityOperationPerProfileDaoImpl.java	370	232	0	0
SecurityOperationPerUserDaoImpl.java	320	190	0	0
SecurityUserDaoImpl.java	249	144	0	0
SecurityUserProfileDaoImpl.java	284	167	0	0
TipoPrenotazioneDaoImpl.java	184	78	0	0
Totale	**3611**	**1992**	**115**	**74**

Si è passati, a questo punto, ad esaminare il secondo progetto costituente l'applicazione generata da SkyGen. Il primo package analizzato è it.skyit.space.dao, in esso le linee di codice contate sono state 3726. Più del 3% del codice è stato scritto manualmente per apportare le giuste modifiche all'applicazione.

2. it.skyit.space.security

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
ApplicationHolderImpl.java	53	28	0	0
AuditLoggerProviderJdbcImpl.java	46	21	0	0
UserDetailsJdbcDaoImpl.java	153	94	0	0
Totale	**252**	**143**	**0**	**0**

In questo secondo package non è stata apportata alcuna modifica, infatti, è stato contato solo codice generato da SkyGen.

3. it.skyit.space.service

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
AulaServiceImpl.java	113	46	0	0
AulaDotazioneServiceImpl.java	63	23	0	0
BaseEntityServiceImpl.java	24	6	0	0
DotazioneServiceImpl.java	63	23	0	0
PersonaleServiceImpl.java	63	23	0	0
PrenotazioneServiceImpl.java	114	47	0	0
PrenotazionePersonaleServiceImpl.java	117	51	110	63
SecurityApplicationServiceImpl.java	64	24	0	0
SecurityElementaryOperationServiceImpl.java	99	42	0	0
SecurityOperationLogServiceImpl.java	100	42	0	0
SecurityOperationPerProfileServiceImpl.java	122	52	0	0
SecurityOperationPerUserServiceImpl.java	116	49	0	0
SecurityUserServiceImpl.java	99	42	0	0
SecurityUserProfileServiceImpl.java	64	24	0	0
TipoPrenotazioneServiceImpl.java	64	24	0	0
Totale	**1285**	**518**	**110**	**63**

Nel package it.skyit.space.service 1285 sono le linee di codice generato dallo strumento SkyGen, sono, invece, 110, le linee di codice scritte manualmente. Quindi, il 93% del codice risulta essere generato.

- Space-root

1. it.skyit.space.anagrafica

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
Aula.java	64	7	0	0
AulaDotazione.java	48	6	0	0
Dotazione.java	37	5	0	0
Personale.java	65	7	0	0
Prenotazione.java	103	10	0	0
PrenotazionePersonale.java	48	6	0	0
TipoPrenotazione.java	37	5	0	0
Totale	**402**	**46**	**0**	**0**

Documenti	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
Oracle-DDL.ddl	610	0	0	0
pom.xml	188	0	0	0
SkyDoc_SQLData.sql	3051	0	0	0
skyit_checks.xml	225	0	0	0
SQL_Data.sql	338	0	0	0
Totale	**4412**	**0**	**0**	**0**

In questo progetto non c’è stato bisogno di effettuare alcuna modifica, quindi tutto il codice è quello generato dallo strumento SkyGen.

- Space-web

1. it.skyit.space.controller

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
BaseController.java	118	44	0	0
BasePopupController.java	38	11	0	0
ManageAulaController.java	411	114	0	0
ManageAulaDotazioneController.java	339	157	0	0
ManageDotazioneController.java	335	156	0	0
ManagePersonaleController.java	335	156	0	0
ManagePrenotazioneController.java	422	182	45	16
ManagePrenotazionePersonaleController.java	341	150	0	0
ManageSecurityApplicationController.java	436	195	0	0
ManageSecurityElementaryOperationController.java	407	200	0	0
ManageSecurityOperationLogController.java	298	137	0	0
ManageSecurityUserController.java	373	188	0	0
ManageSecurityUserProfileController.java	446	220	0	0
ManageTipoPrenotazioneController.java	339	157	0	0
PopupAulaController.java	100	46	0	0
PopupPersonaleController.java	100	46	0	0
PopupSecurityElementaryOperationController.java	102	47	0	0
PopupSecurityUserController.java	100	46	0	0
PopupSecurityUserProfileController.java	100	46	0	0
Totale	**5140**	**2298**	**45**	**36**

Nel package it.skyit.space.controller sono state apportate modifiche al file ManagePrenotazioneController, in quanto sono stati realizzati maggiori vincoli, non esprimibili nel diagramma UML inizialmente realizzato, quindi, parte del codice è stato scritto manualmente.

2. it.skyit.space.decorator

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
PrenotazioneDecorator.java	0	0	105	52
Totale	**0**	**0**	**105**	**52**

In questo package è presente una classe decorator interamente scritta a mano, in quanto bisognava modificare una tabella già esistente. In questo caso la percentuale di codice non generato è il 100%.

3. it.skyit.space.form

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
AulaDotazioneForm.java	107	33	0	0
AulaForm.java	139	43	0	0
BaseForm.java	22	5	0	0
DotazioneForm.java	107	33	0	0
PersonaleForm.java	107	33	0	0
PrenotazioneForm.java	139	43	0	0
PrenotazionePersonaleForm.java	109	34	0	0
SecurityApplicationForm.java	141	47	0	0
SecurityElementaryOperationForm.java	148	51	0	0
SecurityOperationLogForm.java	108	34	0	0
SecurityUserForm.java	139	43	0	0
SecurityUserProfileForm.java	141	47	0	0
TipoPrenotazioneForm.java	107	33	0	0
Totale	**1514**	**479**	**0**	**0**

Analizzando tali dati raccolti, si può notare come in questo package, tutte le linee di codice risultano essere generate.

4. it.skyit.space.security

File	Linee di codice generate con commenti	Linee di codice generate senza commenti	Linee di codice non generate con commenti	Linee di codice non generate senza commenti
FilterSecurityInterceptorPostProcessor	114	81	0	0
Totale	**114**	**81**	**0**	**0**

Anche in questo caso, non vi è stato bisogno di alcun tipo di intervento sul codice.

Si consideri, ora, il contenuto della cartella webapp del progetto Space-web:

- META-INF

Documenti	Linee di codice generate	Linee di codice non generate
context.xml	31	0
Totale	**31**	**0**

- WEB-INF
  - Button

Documenti	Linee di codice generate	Linee di codice non generate
genericDeleteButton.jsp	5	0
genericEditButton.jsp	5	0
genericListButton.jsp	5	0
genericNewButton.jsp	5	0
genericPopupButton.jsp	5	0
genericViewButton.jsp	7	0
genericViewCloneButton.jsp	8	0
genericViewClonePrintButton.jsp	11	0
genericViewPrintButton.jsp	10	0
genericWizardButton.jsp	19	0
Totale	**80**	**0**

- Common

Documenti	Linee di codice generate	Linee di codice non generate
body.jsp	1	0
breadCrumb.jsp	9	0
homePage.jsp	10	0
infobar.jsp	36	0
loginPage.jsp	59	0
menuPage.jsp	9	0
message.jsp	2	0
refreshSecurityPage.jsp	23	0
Totale	**149**	**0**

- Layout

Documenti	Linee di codice generate	Linee di codice non generate
mainLayout.jsp	304	0
Totale	**304**	**0**

- Component

Documenti	Linee di codice generate	Linee di codice non generate
listAulaDotazioneForAula.jsp	72	0
listPrenotazionePersonaleForPrenotazione.jsp	121	0
listSecurityElementaryOperationForSecurityApplication.jsp	256	0
listSecurityOperationLogForSecurityElementaryOperation.jsp	43	0
listSecurityOperationPerProfileForSecurityElementaryOperation.jsp	96	0
listSecurityOperationPerProfileForSecurityUserProfile.jsp	96	0
listSecurityProfilePerUserForSecurityUser.jsp	91	0
listSecurityProfilePerUserForSecurityUserProfile.jsp	139	0
listSecurityUserProfileForSecurityApplication.jsp	160	0
Totale	**1074**	**0**

Documenti	Linee di codice generate	Linee di codice non generate
listAulaDotazioneForAula.jsp	72	0
listPrenotazionePersonaleForPrenotazione.jsp	121	0
listSecurityElementaryOperationForSecurityApplication.jsp	256	0
listSecurityOperationLogForSecurityElementaryOperation.jsp	43	0
listSecurityOperationPerProfileForSecurityElementaryOperation.jsp	96	0
listSecurityOperationPerProfileForSecurityUserProfile.jsp	96	0
listSecurityProfilePerUserForSecurityUser.jsp	91	0
listSecurityProfilePerUserForSecurityUserProfile.jsp	139	0
listSecurityUserProfileForSecurityApplication.jsp	160	0
Totale	**1074**	**0**

- Pages

File	Linee di codice generate	Linee di codice non generate
detailAula.jsp	38	16
detailAulaDotazione.jsp	33	10
detailDotazione.jsp	21	7
detailPersonale.jsp	36	15
detailPrenotazione.jsp	56	20
detailPrenotazionePersonale.jsp	35	10
detailSecurityApplication.jsp	48	0
detailSecurityElementaryOperation.jsp	95	0
detailSecurityOperationLog.jsp	57	0
detailSecurityUser.jsp	93	0
detailSecurityUserProfile.jsp	69	0
detailTipoPrenotazione.jsp	21	9
listAula.jsp	54	13
listAulaDotazione.jsp	56	21
listDotazione.jsp	44	15
listPersonale.jsp	61	17
listPrenotazione.jsp	93	22
listPrenotazionePersonale.jsp	62	19

listSecurityApplication.jsp	67	25
listSecurityElementaryOperation.jsp	117	0
listSecurityOperationLog.jsp	70	0
listSecurityUser.jsp	105	0
listSecurityUserProfile.jsp	76	0
listTipoPrenotazione.jsp	44	12
Totale	**1451**	**231**

- Popups

File	Linee di codice generate	Linee di codice non generate
popupAula.jsp	52	0
popupPersonale.jsp	63	0
popupSecurityElementaryOperation.jsp	114	0
popupSecurityUser.jsp	102	0
popupSecurityUserProfile.jsp	73	0
Totale	**404**	**0**

- Tiles

File	Linee di codice generate	Linee di codice non generate
tilesAula.xml	67	0
tilesAulaDotazione.xml	53	0
tilesDotazione.xml	53	0
tilesPersonale.xml	67	0
tilesPrenotazione.xml	53	0
tilesPrenotazionePersonale.xml	53	0
tilesSecurityApplication.xml	53	0
tilesSecurityElementaryOperation.xml	67	0
tilesSecurityOperationLog.xml	53	0
tilesSecurityUser.xml	67	0
tilesSecurityUserProfile.xml	67	0
tilesTipoPrenotazione.xml	53	0
Totale	**706**	**0**

Documenti	Linee di codice generate	Linee di codice non generate
jasper-views.xml	121	0
Servlet-context.xml	134	0
tiles-defs-base.xml	185	0
web.xml	135	0
pom.xml	65	0

Com'è possibile vedere da tutti questi dati, estrapolati dal progetto Space-web, la maggior parte del codice utilizzato è quello generato, ma c'è stato, comunque, bisogno di scrivere manualmente codice, essendo questo progetto dedicato principalmente alla view dell'applicazione. Si trovano, infatti, linee di codice scritte per customizzare, in particolare, le diverse pages. In tale sottocartella si individua un 14 % circa di codice scritto manualmente.

6.6 SkyGen vs Spring MVC

Per giustificare l'utilità di SkyGen e degli altri generatori di codice, sarà effettuata una comparazione di qualità tra lo sviluppo di una semplice web application utilizzando il solo Spring Framework, e lo sviluppo mediante l' utilizzo dello strumento SkyGen.

Avendo già esaminato nei capitoli precedenti in cosa consiste la progettazione e la realizzazione di un' applicazione web con l'utilizzo di SkyGen, verrà esposto di seguito lo sviluppo "manuale" di una semplice applicazione. Questo ci porterà a capire quanto tempo e quanto sforzo si risparmiano nell'utilizzo di un generatore di codice e come con esso si riesca ad eliminare il gap tra le fasi di modellazione e programmazione.

L'applicazione, che verrà analizzata, è un'applicazione per la gestione di un sito di inserzioni per la vendita di oggetti, che chiameremo Mercatino Spring. Il suo sviluppo sarà affrontato per gradi, adottando un procedimento che spesso viene definito "top-down", ovvero si realizzerà per prima cosa lo strato web, poi il service ed infine il DAO.

6.6.1 Configurazione

La prima cosa da fare, per lo sviluppo di questa web application, con l'utilizzo del framework Spring MVC, è, però, la preparazione dell' ambiente di lavoro. Tale preparazione consiste principalmente nel creare l'alberatura dei sorgenti e preparare il sistema di build. Per la gestione dei progetti e la build automation è utilizzato il software Maven, che consente di avere i seguenti vantaggi:

- tutte le fasi del build sono automatizzate e, se si rispettano le convenzioni, non è necessario configurare alcunché;
- l'esecuzione dei test fa parte del ciclo di build;
- la gestione delle dipendenze è molto efficace.

Inoltre, occorre creare e riempire il file web.xml all'interno della directory src/ webapp / WEB-INF. Tale file sarà utilizzato per informare il server, in cui è installata un'applicazione J2EE, riguardo alla configurazione dell'applicazione stessa.

Infine, si ha il bisogno di aggiungere al progetto le seguenti dipendenze:

- javaee-api ver. 6, jstl ver 1.2 e junit 4.8, già configurate;
- i moduli spring-core e spring-webmvc di Spring framework, oltre alla libreria spring-test necessaria per l'integrazione con JUnit.

Per aggiungere queste dipendenze è sufficiente modificare il file pom.xml nelle sezioni <dependencies> e <properties>.

6.6.2 Strato web

Finita la fase di analisi e configurazione dell'ambiente di lavoro, si può passare alla scrittura del codice. Si procede, quindi alla scrittura del test JUnit. Infatti, è prassi comune di tutte le pratiche agili scrivere i test prima dell'implementazione, in questo modo si hanno diversi vantaggi, fra i quali ne citiamo un paio di significativi:

- il codice viene utilizzato subito, questo facilita l'individuazione di eventuali problemi o difficoltà nell'utilizzo;

- si stabilisce da subito una definizione di "funzionante", rimuovendo molte incomprensioni fra clienti, sviluppatori e tester.

Per tutti i test che necessitano di accedere all'application context di Spring, file che specifica le informazioni contestuali su un thread dell'applicazione e va creato nella cartella src/main/resources, si utilizza una specializzazione di AbstractJUnit4SpringContextTests, una classe messa a disposizione da Spring e che consente di caricare a runtime un application context e di mantenerlo in memoria durante l'esecuzione dei test, velocizzando di molto l'esecuzione degli stessi. La classe, dunque, si chiamerà AbstractTest, e il sorgente deve essere memorizzato in AbstractTest.java. Terminata la fase di scrittura di test si passa al modello dei dati dell'applicazione. Esso sarà costituito da una semplice classe, Insertion, che conterrà i dati di un'unica inserzione. Implementata tale classe, si può passa ad un'ulteriore implementazione, quella della homepage. Il controller per l'homepage si chiamerà IndexController, ma prima di attivarlo, si dovrà creare il relativo test in IndexControllerTest.java, in accordo con l'approccio agile che si sta utilizzando per la realizzazione della web application.

Inizialmente, il controller sarà implementato seguendo la strategia top-down, ipotizzando, dunque, l'esistenza di servizi funzionanti, che in realtà saranno implementati solo in un secondo momento. Inoltre, l'implementazione di IndexController dovrà essere salvata in IndexController.java.

A questo punto dello sviluppo dell'applicazione, l'unico modo per poter proseguire è, quindi, realizzare un'implementazione "finta", comunemente definita "dummy", di InsertionService. Questo modo di lavorare permetterà di concentrarsi sullo sviluppo dello strato web dell'applicazione, nell'attesa che il team di sviluppo dello strato Service realizzi il suo lavoro. Come al solito, anche in questo caso si realizza prima di tutto il test, che verrà salvato in InsertionServiceTest.java. Fatto ciò si può passare alla realizzazione dummy di InsertionService, da salvare in InsertionService.java

Per completare lo strato web dell'homepage manca ora solo la realizzazione della pagina di visualizzazione, realizzeremo, quindi, le pagine JSP. Prima, però, di fare ciò, come al

solito, bisogna mettere mano ai file di configurazione. I file di configurazione coinvolta in quest' ulteriore modifica saranno :

1. applicationContext.xml
2. web.xml
3. dispatcher-servlet.xml : file di configurazione del framework Spring MVC.

Effettuate tutte le modifiche necessarie ai file di configurazione si può passare alle vere e proprie JSP. Da notare, che tutte le JSP dell'applicazione utilizzeranno almeno una delle tag library messe loro a disposizione, conviene quindi anzitutto, creare, la pagina src/main/webapp/WEB-INF/pages/common/taglibs.jsp, che verrà inclusa in ogni pagina jsp e comprendete le eventuali tag library utilizzate. L'index.jsp sarà, nell'ambito dell'applicazione che si sta realizzando, la jsp invocata come welcome page nel caso in cui nella URL non ci sia una pagina specifica.

6.6.3 DAO – Data Access Object

Nello sviluppo dell'applicazione si è percorsa fin qui molta strada : si è partiti da un' analisi poco formale fino ad arrivare ad avere un'applicazione funzionante, ma tutto ciò non basta, mancano, infatti, almeno due caratteristiche essenziali:

1. il database: i dati infatti al momento non vengono letti e scritti da nessuna parte;
2. uno stile grafico accattivante, assolutamente necessario per avere più clienti.

Per il momento ci occuperemo solo del primo punto, ovvero il salvataggio permanente e sicuro dei dati.

A questo punto della realizzazione, l'applicazione, dunque, sta cominciando a strutturarsi definitivamente nei classici tre strati: web, service e DAO (Data Access Objects). È quest'ultimo strato che si occupa effettivamente dell'interazione vera e propria con il database sottostante, "sporcandosi" le mani con query SQL o altri linguaggi.

Il primo DAO che si realizza è l'InsertionDao. Sempre nell'ottica di procedere un passo alla volta, si sviluppa dapprima una implementazione "dummy", per facilitare l'integrazione con gli altri componenti; successivamente si implementerà il collegamento

vero e proprio al database. Quindi, per ora si copiano, banalmente, i metodi getMostRecentInsertions() e getInsertion() dall' InsertionService nel file InsertionDao. Fatto ciò, bisogna modificare la classe InsertionService, facendogli utilizzare i metodi di InsertionDao appena aggiunti, affinché si possa concretizzare per l'interazione con il database. Da ciò è facilmente intuibile come InsertionService sia solo un semplice "passacarte" che si limita a girare le richieste al DAO. Questo stile di implementazione non è esente da difetti e soprattutto critiche, perché apparentemente ogni chiamata viene implementata due volte! In realtà ciò avviene solamente all'inizio di un progetto, quando i service si limitano a caricare e salvare oggetti molto semplici. Mano a mano che la complessità degli use case sale infatti, una singola chiamata ad un service avrà bisogno di più chiamate verso lo strato DAO o verso altri service. Inoltre, si deve ricordare che, per com'è stata configurata l'applicazione, ogni chiamata ad un service è una transazione a se stante. I DAO, quindi, vanno implementati dando per assodato il fatto che ci sia una transazione in corso, per evitare di doversi occupare della gestione delle stesse. Un aspetto positivo è che, dopo tutte queste modifiche, nessun client di InsertionService è stato modificato: addirittura il test di InsertionService continuerà a dare esito positivo, per il semplice fatto che si è solamente aggiunto uno strato applicativo più "in basso", mantenendo inalterato il contratto (o interfaccia) con gli strati superiori.

Siamo quasi giunti agli strati più "bassi", ma non meno interessanti, della nostra applicazione: DAO (Data Access Objects) e database. Prima di andare avanti, però, bisogna decidere come realizzare questo importantissimo strato applicativo. Lo strato di accesso ai dati è, di fatto, quello che maggiormente si "sporca le mani": deve presentare un'interfaccia semplice, efficiente e logica verso i dati, ma per farlo deve utilizzare strumenti che spesso non sono per nulla semplici, efficienti e logici. Gli strati superiori, infatti, maneggiano veri e propri oggetti, dotati di proprietà di vario tipo e con differenti semantiche di accesso come le collezioni (java.util.Set, java.util.List o java.util.Map), identificatori di vario tipo, eccetera. Se gestire tramite un database relazionale questo tipo di dati è una sfida, nascondere agli strati superiori i dettagli

implementativi lo è ancora di più. Tutte queste problematiche sono raggruppate sotto la definizione di persistenza degli oggetti, argomento molto interessante, ma anche molto complesso.

Per le prove dell'applicazione si utilizzerà il popolare database MySql. Una volta installato il database si crea uno schema denominato arteraspringtutorial, si salva il contenuto dello script di creazione in src/main/resources/create-tables.sql, si apre un client SQL ed, infine, si esegue lo script sullo schema appena creato. Lo script creerà l'unica tabella al momento utilizzata dall'applicazione. Un secondo script, invece, si occuperà di inserire alcuni dati di test. L'ultimo passo, prima dell'implementazione vera è propria, è l'aggiunta del driver JDBC di MySql, necessario per connettersi al database, si aggiunge, quindi, una dipendenza nella sezione <dependencies> del file pom.xml. Per completare l'applicazione, l'ultimo step da effettuare è fare in modo che utilizzi, per il salvataggio e la lettura dei dati, il database creato nel paragrafo precedente. Per prima cosa si deve creare il file src/main/resources/jdbc.properties. Questo file memorizza i dati di accesso al database, e verrà utilizzato anche dall'applicazione finale. Fatto ciò, si passa all'aggiunta di ulteriori righe di configurazione relative al jdbc nell' applicationContext.xml.

6.6.4 Spring JDBC

In realtà, è facile osservare come la libreria Spring JDBC sia già inclusa nelle dipendenze del progetto (spring-jdbc) e contenga diverse classi di utilità che consentono di gestire in maniera molto elegante e affidabile le complesse e decisamente poco eleganti e usabili JDBC API, consentendo al programmatore di concentrarsi sulla realizzazione delle query e non su aspetti "burocratici", ovvero apertura/chiusura connessioni e delle transazioni, corretta gestione delle eccezioni, iterazione sui risultati, e molto altro. JdbcTemplate è la classe più importante di questo package e come molte delle classi di utilità di Spring utilizza il Template Method design pattern.

Dunque, Spring mette a disposizione una classe astratta: JdbcDaoSupport per implementare i DAO utilizzando le JDBC API. Estendendo questa classe astratta, e

configurando un dataSource, si avrà a disposizione un DAO con un'istanza già pronta e funzionante di JdbcTemplate, non ci sarà, quindi, nessun bisogno di configurare alcunché, nessun bisogno di specificare transazioni gestite da Spring e nessuna gestione di eccezioni JDBC. Cambia, quindi, l'implementazione di InsertionDao. In esso, le string SQL vengono create tramite degli StringBuilder per la massima flessibilità, e ai metodi query() e queryForObject() di JdbcTemplate viene passata un'istanza diRowMapper . Quest'ultima verrà utilizzata per mappare correttamente i dati contenuti nel ResultSet nelle property delle istanze di oggettiInsertion, la sua implementazione, infatti sarà specifica per la classe considerata. Con quest'ultima implementazione la semplice applicazione web si può ritenere conclusa e funzionate.

6.6.4 Risultati ottenuti

Anche se in Spring MVC la realizzazione di un'applicazione web richiede l'utilizzo di librerie proprie di Spring e il linguaggio utilizzato risulta essere un misto tra imperativo e dichiarativo, mentre in SkyGen la situazione è diametralmente opposta, scopo di questo paragrafo sarà quello di comparare, comunque, le due tecnologie, in termini di linee di codice scritte manualmente e approcci di sviluppo utilizzati.

La realizzazione di questa semplice web application ha previsto numero fasi di sviluppo, implementazione e modifica. Inizialmente è stata effettuata una fase di configurazione dell'ambiente e del progetto stesso, successivamente si è passati allo strato web poi al service ed infine al DAO, in questo modo si è suddivisa l'applicazione in tre strati e la sua realizzazione è stata così resa più facile e schematica. In realtà, nonostante la semplicità dei passi da eseguire per creare una web application si sono scritte manualmente innumerevoli linee di codice e si e spesso molto tempo per completare l'implementazione e rendere funzionante l'applicazione.

Ben diversa sarebbe stata la situazione se si fosse utilizzato un generatore di codice. La differenza sostanziale sta nell'approccio utilizzato, infatti, con l'utilizzo dello strumento SkyGen, anziché adoperare l'approccio top-down, sarebbe stato utilizzato l'approccio

bottom-up. Per iniziare la realizzazione di un'applicazione web con SkyGen, difatti, si sarebbe partiti dall'idea di quella che sarebbe stata la struttura del database, dalla quale si ricavava il modello UML, input del generatore. L'approccio della generazione di codice, secondo la nostra esperienza, inoltre, fornisce una serie di vantaggi apprezzabili:

- Riduzione del tempo di sviluppo di componenti software ricorrenti, con una struttura generalizzabile.
- Minor probabilità di errore sulle porzioni di codice generato: una volta costruito il sistema, gli errori sulle parti generate automaticamente diventato sostanzialmente nulli.
- Uniformità del codice: la porzione di codice generata è ovviamente sempre conforme al generatore e quindi ad uno standard predefinito.
- Le modifiche e i bugfix al codice, se applicate al generatore, vengono propagate automaticamente alla successiva generazione.
- Minor frustrazione per sviluppi ripetitivi (mai da trascurare, perchè gli sviluppatori son pur sempre persone!).

Per concludere, si vogliono aggiungere anche un paio di dati di esempio che possono essere utili a dare un'idea della portata dei concetti sopra esposti. La normale implementazione di un'applicazione web molto semplice, in linguaggio Java e sviluppata tramite il framework Spring MVC, conta circa 400 linee di codice, tra configurazione e implementazione dei tre strati dell'applicazione.

Sfruttando, invece, la generazione di codice, con SkyGen, si può arrivare a generarne automaticamente lo stesso codice, ma evitando di fare manualmente tutte quelle operazioni noiose che richiedono comunque uno sforzo abbastanza notevole sia in termini di forza lavoro che di tempo, quali la connessione al database, la creazione delle tabelle del database, i metodi getter e setter delle entità del modello dati e così via. Tutto ciò permette al programmatore di concentrarsi su altri aspetti e di mettere mano al codice solo per effettuare customizzazione, in modo tale da rendere l'applicazione sempre migliore agli

occhi del cliente. Quindi, con un generatore di codice si può passare, in pratica, dall'implementare 400 linee di codice ad appena 50, circa un 88% in meno.

Un ulteriore vantaggio nell'utilizzo di SkyGen è l'implementazione automatica delle operazioni CRUD per ogni entità dell'applicazione che si sta realizzando. Si consideri, ad esempio, l'operazione elementare "save new element", per realizzarla devono essere scritti manualmente più di 300 linee di codice, e i relativi test per verificare il corretto funzionamento e la qualità del codice. Con l'utilizzo, invece, di un generatore di codice si devono solo scrivere meno di 90 linee di codice per customizzare, in quanto la funzione risulta già essere totalmente implementata e funzionante, si sono dunque scritte il 70% in meno di linee di codice. Pensare che questo vale per ogni entità dell'applicazione, sottolinea ancora una volta il grande risparmio di tempo e lavoro che può apportare l'utilizzo di SkyGen.

Un ultimo vantaggio che possiamo esaminare, nell'utilizzo di SkyGen, è la possibilità di caricare il database con gli script già scritti dal generatore. Non vi è alcun bisogno di scrivere codice SQL per la creazione di tabelle o l'inserimento di dati, tale codice, infatti, viene generato automaticamente, l'unico compito del programmatore è caricare i file nel db. Volendo passare ai numeri, nell'applicazione prima realizzata sono stati scritte linee di codice SQL, con l'utilizzo di SkyGen, invece non si sarebbe scritta alcuna linea di codice, quindi si sarebbe avuto un risparmio del 100%.

Anche per quanto riguarda la view dell'applicazione sia ha che con SkyGen essa viene implementata automaticamente, senza , invece, bisogna decidere, anzitutto, quale possa essere l'aspetto estetico e poi cercare di realizzarlo attraverso l'implementazione delle diverse jsp. In realtà, in quest'aspetto SkyGen non dimostra tutta la sua utilità, in quanto il generatore genera una view preimpostata, se si volessi, quindi, dare alle interfacce un aspetto completamente diverso bisognerebbe rifare interamente questa parte di codice.

6.7 Conclusioni

Fino a poco tempo fa, 1'ingegneria del software è stata sostanzialmente un'attività manuale, nella quale il ricorso a strumenti specializzati avveniva solo nelle ultime fasi del processo attraverso l'uso di ambienti integrati di compilazione, linkaggio e debugging. Oggi, finalmente, gli ingegneri del software possono sfoggiare il loro "primo paio di scarpe": il CASE, sigla che sta per Computer-Aided Software Engineering (ingegneria del software assistita dall'elaboratore).

Certamente questo strumento non è ancora molto sviluppato e di facile utilizzo come lo sono le applicazioni CAD/CAM, ma è uno strumento necessario nella dotazione di uno sviluppatore di software e diventerà col tempo più comodo, più agevole a più adattabile alle esigenze specifiche. Questo sia perché lo sviluppo di strumenti CASE è cominciato solo negli ultimi 25 anni, ma soprattutto perché i programmi CAD / CAM implementano pratiche ingegneristiche che sono state utilizzate manualmente per oltre 100 anni, mentre gli strumenti CASE forniscono un insieme di tool automatici e semiautomatici che implementano una cultura ingegneristica che è nuova a molte aziende. Il CASE dà la possibilità di automatizzare compiti prima svolti manualmente e di controllare con più precisione 1o sviluppo. In modo simile agli strumenti dell'ingegneria assistita dall'elaboratore e agli strumenti di progettazione utilizzati in altri settori, gli strumenti CASE contribuiscono a incorporare 1a qualità nella progettazione. L'obiettivo a cui si vuole arrivare con l'uso degli strumenti CASE è quello di avere uno sviluppo automatico del software partendo dalla definizione delle specifiche tramite schemi a blocchi o altro.

Dall'analisi svolta accuratamente sul codice, sarà semplice dedurre l'utilità dello strumento SkyGen, e quanto effettivamente abbia aiutato nello sviluppo dell'applicazione. Il 97% del codice sorgente di Space Classroom è stato scritto in modo automatico e ciò ha permesso di risparmiare tempo ed energia. Sono state, infatti, generate in modo automatico righe di codice che diversamente sarebbero state onerose e noiose da scrivere.

Tutto ciò ci porterebbe a giudicare in modo estremamente positivo in termini qualitativi e quantitativi lo strumento SkyGen, ma ci sono ancora delle ultime considerazioni da fare.

Per poter effettuare la generazione, si è disegnato, inizialmente, un diagramma delle classi, che rispecchiasse l'intera applicazione che si voleva realizzare, in termini di entità e relazioni. Il compito del generatore era quello di trasformare il digramma in codice e quindi di restituire in output una prima versione dell'applicazione funzionante. Quello che potrebbe meravigliare, in questa prima fase, è la stesura del diagramma, le entità qui riportate sono dotate, semplicemente, di attributi, è compito del generatore aggiungere a ogni classe entità i metodi getter/setter. In realtà, la sola scrittura da parte di SkyGen, non entusiasma. I metodi getter e setter sono in sé molto semplici e possono essere riprese da un progetto già esistente. La particolarità si ritrova, invece, nella scrittura delle operazioni CRUD. Nel package relativo ai controller, le operazioni elementari di creazione, modifica, ricerca ed eliminazione sono implementate per ogni singola entità costituente il diagramma UML. Per comprender meglio quale sia il vantaggio di utilizzare uno strumento che generi automaticamente operazioni CRUD, si è provata a realizzare una semplice operazione di ricerca attraverso chiave primaria nella tabella "Prenotazioni". Per implementare tale funzione sono stati modificati 5 file:

– PrenotazioneDao.java

```
68      /**
69       * findByPrimaryKeyCustom.
70       * @param thePrenotazione
71       * @return
72       */
73
74      Prenotazione findByPrimaryKeyCustom(Prenotazione thePrenotazione);
75
```

– PrenotazioneDaoImpl.java

```
277     public Prenotazione findByPrimaryKeyCustom(Prenotazione thePrenotazione) {
278          StringBuilder sql = new StringBuilder(getFromQuery());
279             sql.append("where prenotazione.prenotazioneId= ? ");
280             Query query = getCurrentSession().createQuery(sql.toString());
281             int ctr = 0;
282             query.setParameter(ctr++, thePrenotazione.getPrenotazioneId());
283             return (Prenotazione)(query.uniqueResult()) ;
284             //return super.list(query);
285         }
```

– PrenotazioneService.java

```
46      /**
47       * findByPrimaryKeyCustom.
48       * @param thePrenotazione
49       * @return
50       */
51      Prenotazione findByPrimaryKeyCustom(Prenotazione thePrenotazione);
52
```

– PrenotazioneServiceImpl.java

```
118     /**
119      * {@inheritDoc}
120      *
121      * @see it.skyit.space.service.PrenotazioneService
122      * #findByPrimaryKeyCustom(it.skyit.space.entity.Prenotazione)
123      */
124     public Prenotazione findByPrimaryKeyCustom(Prenotazione thePrenotazione) {
125         return getPrenotazioneDao().findByPrimaryKeyCustom(thePrenotazione);
126     }
```

– ManagePrenotazioneController.java

```
426     /**
427      * Prepara e lancia la pagina di visualizzazione dettaglio dell'elemento di
428      * tipo Prenotazione.
429      *
430      * @param id identificativo univoco dell'oggetto da visualizzare.
431      *           Il parametro viene automaticamento
432      *           popolato dall'estrazione del codice identificativo contenuto
433      *           direttamente nell'url path,
434      *           grazie alla presenza della paremeter annotation @PathVariable.
435      * @param form il form di gestione.
436      * @param model spring model object.
437      * @return la tiles definition name viewPrenotazione.
438      **/
439     @RequestMapping(value = "viewCustom/{id}", method = RequestMethod.GET)
440     public String viewElementCustom(@PathVariable String id, PrenotazioneForm form, Model model) {
441         Prenotazione thePrenotazione = PrenotazioneFactory.getNewEntity(id);
442         thePrenotazione = PrenotazioneFactory.getService().findByPrimaryKeyCustom(thePrenotazione);
443         if (thePrenotazione == null) {
444             throw new BusinessException(new UserMessage("message.genericNotFound", ENTITY_NAME));
445         }
446         form.setThePrenotazione(thePrenotazione);
447         form.setPageStatus(PageStatus.READ_PAGE_STATUS);
448         return finalizeModelAndView("viewPrenotazione", model, id);
449     }
450 }
```

Com'è possibile vedere, lo scrivere una semplice operazione ha portato a modificare diversi file, ha comportato la scrittura manuale di diverse righe di codice, e l'impiego di non poco tempo per rendere tale operazioni funzionante all'interno dell'applicazione. Dunque, risulta, adesso facile, apprezzare il contributo del generatore SkyGen nella realizzazione di Space Classroom. Inoltre, è possibile affermare che, nel caso particolare

dello strumento che si sta analizzando, SkyGen, le operazioni CRUD, che si possono esaminare dopo la generazione del codice sono specifiche dell'applicazione che si sta costruendo, ma, effettivamente, esistono librerie a supporto delle operazioni CRUD che restano fuori dal contesto funzionale e sono generiche per tutte le applicazioni.

Un'ulteriore considerazione da fare per valutare la qualità del code generator SkyGen, è relativa alle librerie, infatti, ci si potrebbe chiedere se le librerie utilizzate fanno parte o meno del codice generato. In caso di risposta negativa, il peso da dare alle librerie è inferiore, in quanto non risulta essere parte del codice generato, in caso di risposta positiva, invece, bisogna attribuire a SkyGen un ulteriore merito. In realtà in SkyGen Il codice generato relativo al middlelayer in fase di packging viene convertito in libreria (.jar) e potrebbero essere utilizzate anche in altre applicazioni laddove ci fossero integrazioni tra sistemi. Si può, quindi, affermare che SkyGen è davvero uno strumento utile alla realizzazione di applicazioni web gestionali. Esso, infatti, è stato costruito in modo da rispondere esattamente alle esigenze dell'azienda SkyIT e da velocizzare il processo di progettazione e realizzazione.

Un ultimo aspetto da considerare sono i test. Essi, effettivamente, non sono generati dallo strumento SkyGen, ma questo aspetto potrebbe essere semplicemente giustificato dal fatto che, essendo l'intero progetto costituito principalmente da codice generato, risulterebbe essere inutile testare un codice che per definizione è corretto. Dunque, in tal contesto la fase di test non risulta essere prioritaria.

In conclusione è possibile dire che l'utilizzo di un generatore di codice comporta numerosi vantaggi come ad esempio la generazione automatica del codice, partendo dal modello, a tempo zero, o meglio: il tempo viene speso nella progettazione del modello, inoltre il codice del modello viene creato e mantenuto in sincronia col progetto del modello automaticamente, quindi è potenzialmente ridotto a zero il rischio di scrivere codice non corretto.

Conclusioni

L'applicazione realizzata durante lo svolgimento di questa tesi non deve e non può essere paragonata ad altri prodotti commerciali presenti sul mercato. La sua realizzazione è stata, infatti, intesa come un tentativo di applicare il maggior numero possibile dei concetti della tecnologia J2EE, Spring MVC, Application Framework e Code Generation, verificando "sul campo" quali sono i problemi, le scelte di progetto, le difficoltà, le limitazioni ed i vantaggi che un team di sviluppo può incontrare.

Il lavoro che deve essere svolto per sviluppare software di questo tipo è comunque molto vasto ed è, per questo, generalmente suddiviso tra più persone, le cui conoscenze sono tra loro complementari. Per questo motivo l'azienda SkyIT introduce la suite di strumenti Sky*, ed in particolare il framework SkyGen, in quanto esso permette di offrire soluzioni software nell'ambito dell'Information & Communication Technology, in tempi relativamente brevi e sfruttando una minore forza lavoro.

Dall'analisi effettuata, inoltre, è stato possibile evidenziare quali sono i passi fondamentali da seguire per lo sviluppo di applicazioni attraverso l'utilizzo di un generatore di codice. Sono stati esaminati, in totale, tre generatori di codice e di ognuno si sono evidenziati, oltre all'utilizzo, anche i rispettivi vantaggi e svantaggi. L'attenzione, però, si è focalizzata, in modo particolare sui vantaggi che si possono avere nell'utilizzo di tali strumenti per la realizzazione di applicazioni web. Nello specifico, l'analisi fatta su

SkyGen ha interessato la sua struttura e le sue caratteristiche, ci si è, infatti, soffermati su quale sia la struttura dell'applicazione realizzata con tale strumento e quale sia il codice generato. Infine, per meglio comprendere l'effettiva utilità dell'utilizzo di uno strumento CASE, si è analizzato il codice sorgente prodotto da SkyGen. Oltre ad un'analisi qualitativa si è effettuata un'analisi quantitativa, si è, quindi, considerato il quantitativo di codice generato rispetto al codice scritto manualmente. Da questo tipo di analisi è stato possibile capire quanto effettivamente sia utile l'utilizzo di SkyGen nella realizzazione di applicazioni gestionali, essendo il 97% dell'applicazione Space composta da codice generato. Dunque, l'esperienza trattata da questa tesi e sintetizzata nei paragrafi precedenti, suggerisce che, se vengono valutati e rispettate tutte le condizioni di utilizzo degli strumenti CASE si possono ottenere grandi benefici in termini sia di architettura generale, poiché consente di realizzare vere architetture multilivello, che di qualità del servizio.

Sviluppi Futuri

SkyGen, così come molti altri strumenti, è un framework per la generazione di applicazioni basate su un modello strutturato, incentrato sul concetto di Model-Driven Development. L'idea che sta alla base di questo framework è la ricerca della fusione tra il mondo della modellazione e quello della programmazione: infatti, la funzionalità principale di SkyGen è ricevere in input un modello, sotto forma di diagramma UML e fornire come output una serie di classi Java completamente implementate, che realizzano i vincoli, le relazioni e le associazioni descritte nel modello di partenza. Dall'analisi qualitativa e quantitativa sviluppata nei capitoli precedenti, si è affermata la grande utilità, in termini di tempo e forza lavoro, nell'utilizzo del prodotto SkyGen, ma nonostante ciò la tesi non è priva di spunti per sviluppi futuri. È stata, infatti, sottolineata l'assenza di codice di test dal codice generato. Il testing è un procedimento che fa parte del ciclo di vita del software ed utilizzato per individuare le carenze di correttezza, completezza e affidabilità delle componenti software in corso di sviluppo. Consiste nell'eseguire il software da collaudare, da solo o in combinazione ad altro software di servizio, e nel valutare se il comportamento del software rispetta i requisiti. Fa parte delle procedure di assicurazione di qualità, anche se non è l'unica. Il testing, quindi, è una parte molto importante nello sviluppo di software, ma nel caso di verifica dinamica di codice generato potrebbe essere considerato non prioritario, essendo il codice generato, per definizione,

corretto. In realtà, soprattutto nel caso di realizzazioni di applicazioni web, dove il codice sorgente non è solo generato, ma sono presenti numerose customizzazioni, il testing diventa senza dubbio di fondamentale importanza. Un possibile sviluppo futuro, potrebbe, quindi, essere quello di aggiungere allo strumento SkyGen la possibilità di generare test. In questo modo si renderebbe possibile una rapida verifica dell'intera applicazione generata e customizzata, ed infine diventerebbe possibile garantire la qualità del codice sorgente del prodotto software creato.

Bibliografia

[1] Ian Sommerville, Ingegneria del software 8/Ed., Pearson, 2007, pp. 848.
[2] Carlo Ghezzi, Mehdi Jazayeri, Dino Mandrioli, Ingegneria del software, Pearson, 2004, pp. 648.
[3] Bodoff S., Green D., Haase K., Jendrock E., Pawlan M., Stearns B., The J2EE™ Turorial, Sun Microsystem, 2002.
[4] Marty Hall, Core Servlets and JavaServer Pages, Prentice Hall PTR, 2000, pp. 608.
[5] David Chappell, Tyler Jewell, Java Web Service, O'Reilly, 2002, pp. 276.
[6] Neal Ford, Art of Java Web Development, Manning, 2003, pp. 624.
[7] Roger S. Pressman, Principi di ingegneria del software, McGraw-Hill, 2008, pp. 750.
[8] Il pattern MVC, http://www.html.it/pag/18299/il-pattern-mvc/
[9] Spring Source, Spring Framework, http://www.springsource.org/
[10] Spring Tool Suite, http://www.springsource.org/sts
[11] C. Bauer,G. King, Java persistence with Hibernate, Manning Editore, 2007, pp. 600.
[12] P.Atzeni, S. Ceri, S. Paraboschi, R. Torlone, Basi di dati, modelli e linguaggi di interrogazione, McGraw-Hill Editore, 3â edizione, 2009, pp. 462.
[13] M. Fischer Jon Ellis, Jonathan Bruce, JDBC API Tutorial and Reference, Addison Wesley, 2003, pp. 1280.
[14] JDBC Tutorial, http://docs.oracle.com/javase/tutorial/jdbc/inde/
[15] EMF , http://www.eclipse.org/emf/
[16] Dave Steinberg, Frank Budinsky, Marcelo Patemostro, Ed Merks, EMF: Eclipse Modeling Framework, 2nd Edition, Addison-Wesley Professional, 2008, pp. 744.
[17] Davide Devescovi, Sviluppo di tool basati si un meta modello editabile graficamente – Panoramica e breve guida all'uso di EMF, GEF, GMF.
[18] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, Timothy J. Grose, Eclipse Modeling Framework : A Developer's Guide, Addison Wesley, 2003, pp. 720.
[19] Spring Roo, http://projects.spring.io/spring-roo/
[20] Spring Roo : un RAD per Java EE, http://www2.mokabyte.it/cms/article.run?permalink=mb172_springroo-1
[21] Spring Roo : Fast Java Application Development T ool, http://www.cubrid.org/blog/dev-platform/spring-roo-fast-java-application-development-tool/
[22] J. Arlow, I. Neustadt, UML e Unified Process, McGraw-Hill, 2003, pp. 481.
[23] Colin Atkinson, Thomas Kühne, Model-Driven Development : A Metamodeling Foundation.
[24] Maven, http://maven.apache.org/
[25] Stan4j, http://stan4j.com/

[26] Checkstyle, http://checkstyle.sourceforge.net/
[27] Guida Spring MVC, http://www.html.it/guide/guida-al-framework-spring-mvc/