Spazi Aule con Framework GWT

Ferrari Nico

August 14, 2016

Indice

List of Figures

Chapter 1

Introduzione

Il lavoro di tesi svolto si pone come obiettivo lo sviluppo di una applicazione web che mette a disposizione degli utenti una visione chiara e completa dell'occupazione delle aule negli spazi universitari. L'applicazione si propone quindi di rendere un servizio sia basato sulla semplice e veloce consultazione delle informazioni riguardanti lo stato delle aule universitarie, sia la manipolazione di tali informazioni.

Chapter 2

Strumenti utilizzati

2.1 JAVA

Il linguaggio Java è un linguaggio di programmazione orientato agli oggetti, creato da James Gosling e altri ingegneri di Sun Microsystems. Il gruppo iniziò a lavorare nel 1991, il linguaggio inizialmente si chiamava Oak. Il nome fu successivamente cambiato in Java a causa di un problema di copyright (il linguaggio di programmazione Oak esisteva già nel 1991). Java fu annunciato ufficialmente il 23 maggio 1995 a SunWorld. La piattaforma di programmazione Java è fondata sul linguaggio stesso, sulla Java Virtual Machine (JVM)e sulle API. Java è un marchio registrato di Sun Microsystems.

2.1.1 Panoramica del linguaggio JAVA

Java venne creato per soddisfare quattro scopi:

- essere orientato agli oggetti
- essere indipendente dalla piattaforma
- contenere strumenti e librerie per il networking
- essere progettato per eseguire codice da sorgenti remote in modo sicuro

Orientamento agli oggetti

Java è un linguaggio object-oriented. La programmazione orientata agli oggetti (OOP, Object Oriented Programming) è un paradigma di programmazione che permette di definire oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi. Questi oggetti, come nella vita pratica hanno proprietà rappresentate da valori, e qualità o meglio metodi: ciò che sanno fare questi oggetti.

La programmazione ad oggetti è particolarmente adatta nei contesti in cui si possono definire delle relazioni di interdipendenza tra i concetti da modellare (contenimento, uso, specializzazione).

Tra i vantaggi della programmazione orientata agli oggetti abbiamo:

- fornisce un supporto naturale alla modellazione software degli oggetti del mondo reale o del modello astratto da riprodurre
- permette una più facile gestione e manutenzione di progetti di grandi dimensioni
- permette una più facile gestione e manutenzione di progetti di grandi dimensioni
- l'organizzazione del codice sotto forma di classi favorisce la modularità e il riuso di codice

Indipendenza dalla piattaforma

L'indipendenza dalla piattaforma significa che lâesecuzione di programmi scritti in Java deve avere un comportamento simile in contesti di esecuzione diversi. Per raggiungere questo obiettivo il codice Java viene compilato in un linguaggio intermedio bytecode. Il bytecode è un insieme di istruzioni altamente ottimizzate eseguibili dalla Java Virtual Machine(JVM), disegnata inizialmente come interprete per il bytecode.

La traduzione di un programma Java in bytecode rende molto piu semplice l'esecuzione di un programma in una vasta varietà di ambienti perchè solo la JVM deve essere implementata per ogni piattaforma.

Esecuzione sicura del codice remoto

La piattaforma Java ha caratteristiche progettate per aumentare la sicurezza delle applicazioni Java:

- **La JVM:**
  La JVM esegue una verifica del bytecode prima di eseguirlo per prevenire l'esecuzione di operazioni non sicure e per far rispettare vincoli a runtime. La piattaforma non permette ai programmi di eseguire alcune operazioni insicure e controlli manuali sull'allocazione e deallocazione della memoria.

- **Il Security Manager:**
  La piattaforma fornisce un security manager che permette agli utenti di eseguire codice bytecode inaffidabile in un ambiente "sandbox" progettato per proteggerli da software pericoloso o scritto male evitando che il tale codice abbia accesso ad alcune caratteristiche e API della piattaforma. Il security manager permette di assegnare ai programmi Java una firma digitale in modo tale che l'utente possa scegliere di dare i privilegi a software con firma digitale valida proveniente da entitÃ di fiducia.

- **Le API:**
  Viene fornita una serie di API orientate alla sicurezza, come algoritmi standard di crittografia,autenticazione e protocolli di comunicazione sicuri.

2.1.2 La piattaforma JAVA

La piattaforma Java è una piattaforma solo software che gira in cima ad una piattaforma hardware di base che può essere un computer, una tv, un telefono cellulare, una smart card, ecc.. La piattaforma Java è composta da due blocchi:

- la Java Virtual Machine (JVM)
- la Java Application Program Interface (API)

La JVM è la base della piattaforma Java, mentre la Java API è una collezione di componenti software pronti allâuso per losvolgimento dei più disparati compiti.

Java Virtual Machine

La JVM consiste di:

- class loader
- class verifier
- l'interprete Java

Il class loader carica le classi che formano il bytecode, sia dell'applicazione Java, sia delle API Java necessarie per l'esecuzione da parte dell'interprete Java.

Subito dopo il class verifier controlla che il bytecode sia valido, che non superi i limiti superiori o inferiori dello stack, assicura non esegua aritmetica dei puntatori (che potrebbe potenzialmente portare ad una violazione di memoria). Se il bytecode passa tutti questi controlli, può essere eseguito dall'interprete.

L'interprete può essere di varie forme: può essere un modulo software che interpreta il bytecode in una sola volta oppure può fare uso di un compilatore just-in-time (JIT, o Just-In-Time compiler) che traduce a runt-time il bytecode in codice nativo della macchina ospitante e lo salva in memoria durante l'esecuzione. È anche possibile utilizzare un sistema "misto", in cui il JIT viene applicato solo alle porzioni di codice del programma utilizzate più frequentemente, mentre il resto viene interpretato.

API Java

Le API Java raccolgono una gran quantità di componenti disponibili per scrivere applicazioni di qualsiasi genere. Per questo motivo la piattaforma Java è disponibile in tre configurazioni a seconda dell'uso che se ne vuole fare:

- **Standard Edition.** Fornisce API per le esigenze più comuni, che permette di scrivere applicazioni stand-alone, applicazioni client e server in un contesto di reti di computer, applicazioni per accesso a database, applicazioni per il calcolo scientifico e di altro tipo.
- **Enterprise Edition.** Permette di scrivere applicazioni distribuite.
- **Micro Edition.** Permette di scrivere applicazioni per i terminali mobili e, più in generale, per i dispositivi dotati di poche risorse computazionali (telefoni cellulari, palmari, smart cards ed altri).

2.2 Framework Hibernate

La gestione della persistenza è un argomento delicato nella progettazione di applicazioni software. Lo sviluppo di applicazioni Object-Oriented che utilizzano un DBMS relazionale per immagazinare i dati in memoria secondaria è uno scenario molto comune. Ciò che rende complessa la costruzione dello strato di persistenza è il problema dell'object/relational impedence mismatch, la discrepanza tra il paradigma Object-Oriented e il paradigma relazionale. Fondamentalmente, gli oggetti fanno riferimento ad altri oggetti e perciò danno forma ad un grafo, a differenza degli gli schemi relazionali, che hanno invece una struttura tabellare e basati sull'algebra relazionale che definisce insiemi di tuple. La conversione di tuple in strutture a grafo spesso richiede molto tempo e può risultare difficile, infatti, viene anche descritta "Vietnam of Computer Science". Questo problema può esser diviso in più parti:

- **Struttura, Ereditarietà, Interfaccia e Polimorfismo**:
  Una classe specifica gli attributi e i metodi che devono avere gli oggetti che appartengono ad essa. Una classe può anche far parte di una gerarchia (Concetto di ereditarietà). Il modello relazionale non contempla il concetto di âclasseâ di oggetti e non fornisce una analogia per le gerarchie, interfacce e polimorfismo.

- **Granularità dei tipi di dato**:
  Tipi di dati composti sono tipicamente rappresentati nei linguaggi OO mediante classi di oggetti, mentre nel modello relazionale non si prevede alcun meccanismo per la definizione di tipi di dato composti.

- **Incapsulamento**:
  Lo stato di un oggetto è incapsulato ed è accessibile attraverso i metodi. Lo stato di una riga di una tabella non contempla questo aspetto e può essere modificato in maniera diretta.

- **Identità**:
  Gli oggetti esistono indipendentemente dal loro stato. Essi possono essere identici o uguali. Se due oggetti sono identici, essi sono lo stesso oggetto. Se sono uguali, essi contengono gli stessi valori. Nel modello relazionale,

invece, le righe sono identificate solo dai valori che esse contengono.

- **Associazioni**:
  Il modello relazionale contempla solo un tipo di associazione: l'utilizzo di una chiave esterna, che si riferisce ad una chiave primaria di un'altra tabella. Il paradigma Object-Oriented contempla diverse tipologie di associazione: uno a uno, uno a molti, molti a molti.

- **Coesione**:
  Tutte le proprietà di un oggetto sono contenuto all'interno di esso. Invece, i dati relazioni che corrispondono ad una stessa entità possono essere stati suddivisi in più tabelle (fase di ristrutturazione dello schema logico)

Il problema dell'impedence mismatch deve essere analizzato e risolto nel modo opportuno in fase di progettazione.

Un altro aspetto molto importante nella gestione della persistenza è l'interazione tra lo strato di business logic e lo strato di persistenza. In applicazioni Java la comunicazione con la base di dati avviene utilizzando API specifiche come ODB (Open Database Connectivity) o JDBC(Java Database Connectivity). Una delle strategie più valide e utilizzate è l'uso del pattern Data Access Object (DAO) per la creazione di oggetti che costituiscono uno strato di comunicazione con la base di dati. La realizzazione dei DAO avviene usando API JDBC per rendere disponibili le operazioni CRUD(Create Read Update Delete) allo strato di logica applicativa. I DAO incapsulano l'accesso alla base di dati e permettono di gestire la maggior parte degli scenari ma la loro realizzazione necessita di una conoscenza dettagliata della base di dati. Altri strategie fanno uso della serializzazione, di object oriented database systems oppure dell'Object-Relational Mapping che consiste nel mappare le classi di dominio dell'applicazione su una base di dati relazionale.

2.2.1 Object/Relational Mapping

L'Object/Relational Mapping (ORM) è una tecnica di programmazione che favorisce l'integrazione di sistemi software aderenti al paradigma della programmazione orientata agli oggetti con sistemi RDBMS. Attraverso questo mezzo 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 srumento per la specifica del mapping mediante metadata.
4. Una tecnica per l'implementazione di ORM per interagire con oggeti transazionali al fine di eseguire funzioni di ottimizzazione.

L'associazione fisica tra la classe e la tabella viene ottenuta mediante l'utilizzo di file di descrizione (detti file di mapping), in cui si specificano le modalità di conversione tra gli attributi dell'oggetto e i campi della tabella. In definitiva 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

Uno degli esempi più noti di ORM per il linguaggio Java è Hibernate.

2.2.2 Cos'è Hibernate

Hibernate è un framework open source con servizi ORM in Java. Hibernate ORM(fino alla versione 4.0 conosciuto come Hibernate Core), composto dalle API native di Hibernate e il suo motore, è disponibile alla versione 4.1, utilizzata in questa trattazione. Il software è composto da API Java che permettono di gestire la persistenza con il mapping delle classi sulla base di dati e forniscono interfacce per l'accesso ai dati persistenti.

2.2.3 Architettura Hibernate

Facciamo riferimento alla figura sottostante per trattare i componenti principali dell'architettura Hibernate.

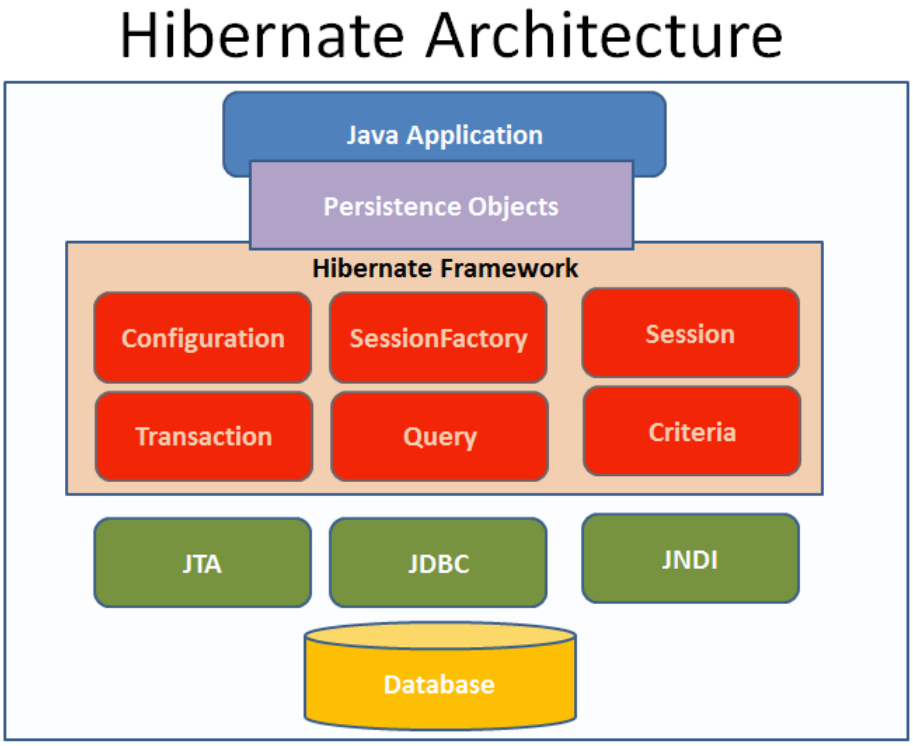

Figure 2.1: Struttura di Hibernate

L'oggetto Configuration è di fondamentale importanza per il funzionamento del framework, in quanto contiene informazioni che riguardano:

- la connessione al database
- il class mapping

la prima viene gestita mediante il file di configurazione hibernate.cfg.xml, mentre il mapping tra le classi java e le tabelle del database, può avvenire mediante file di configurazione xml, oppure con il meccanismo delle annotation. Questo oggetto viene creato una sola volta all'atto della inizializzazione della applicazione,

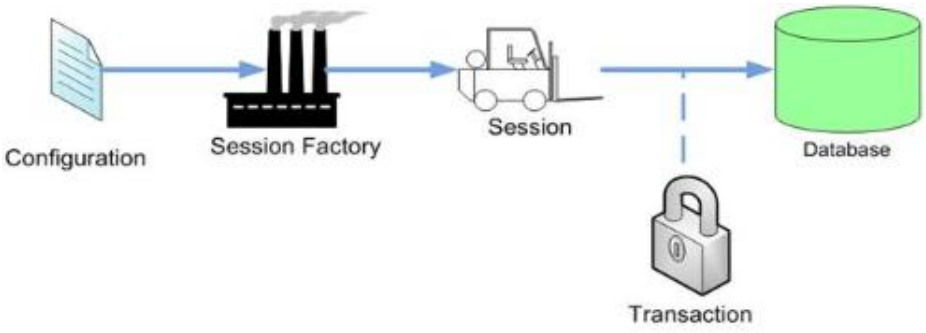

Figure 2.2: Workflow Hibernate

```
Configuration config = new
    Configuration().configure("/it/resources/hibernate.cfg.xml");
```

e dopo che è stato utilizzato per la creazione di un altro oggetto, il SessionFactory, resta inutilizzato.

```
SessionFactory sessionFactory = config.buildSessionFactory();
```

Quest'ultimo quindi, viene anch'esso creato una sola volta allo startup, ma a differenza di Configuration, viene utilizzato in tutta l'applicazione. L'oggetto SessionFactory è thread safe, ed è utilizzato da tutti i thread dell'applicazione, inoltre siccome dipende da Configuration, che a sua volta si riferisce ad uno specifico database, potrà puntare solo a quel database. Nel caso di connessioni a database multipli, avremmo mulipli Configuration e quindi multipli SessionFactory, uno per ogni Configuration. Il SessionFactory, come si evince dal nome, risulta essere uno stampo da cui creare gli oggetti Session, necessari ogni qualvolta occorre effettuare una interazione con il database. Quando con un oggetto di questo tipo viene aperta una sessione di lavoro,

```
Session session = sessionFactory.openSession();
```

viene stabilita una connessione fisica con il database, e siccome non è thread safe, dopo aver effettuato le operazioni necessarie, occorre poi chiuderla manualmente, in modo da non mantenere connessioni aperte inutilmente. Se durante una sessione di lavoro, occorre fare operazioni di modifica sul database, come inserimenti, aggiornamenti o cancellazioni, occorre gestirle come transazioni: a tal proposito l'oggetto Session viene a sua volta utilizzato per la creazione di

oggetti Transaction. In Hibernate le transazioni sono gestite dal TransactionManager che permette allo sviluppatore di astrarsi dal livello sottostante (JDBC, JTA, ecc.) evitando di scrivere codice specifico. Infine Query e Criteria sono utilizzati per recuperare oggetti persistenti. Gli oggetti Query utilizzano SQL oppure HQL (Hibernate Query Language) per il recupero di dati dal database e per la creazione di oggetti, mentre Criteria utilizza oggetti per la costruzione e la esecuzione di una richiesta di recupero dati.

2.3 Framework GWT

La necessità di creare un'applicazione Web nasce dai molteplici vantaggi che le applicazioni RIA (Rich Internet Application) possiedono nei confronti delle tecnologie alternative. Infatti rispetto alle applicazioni desktop, non richiedono installazione, gli aggiornamenti sono automatici, sono indipendenti dalla piattaforma utilizzata, più sicure in quanto girano nel ristretto ambiente del Web browser e maggiormente scalabili perchè la maggior parte del lavoro computazionale viene eseguito dal server.
Con l'avvento della tecnologia Ajax (Asynchronous JavaScript and XML), lo sviluppo di applicazioni Web si basa su uno scambio di dati in background fra Web browser e server, che consente l'aggiornamento dinamico di una pagina Web senza eplicito ricaricamento da parte dell'utente. Purtroppo, scrivere applicazioni Ajax è molto complicato e perciò particolarmente esposto ad errori e bug; questo perchè JavaScript è un linguaggio piuttosto differente da Java e richiede molta pratica per lo sviluppo; il tutto è peggiorato dal fatto che JavaScript tende ad avere differenze in base al browser Web utilizzato, concentrando gli sforzi ed il tempo degli sviluppatori più sulla parte grafica che sulla logica applicativa. Google Web Toolkit (GWT) nasce proprio per risolvere questi problemi, fornendo un vero e proprio livello di astrazione che nasconde il codice JavaScript e provvede automaticamente ad uniformare le differenze tra i browser.
Rilasciato da Google nell'estate 2006 sotto licenza Apache, Google Web Toolkit è un set di tool open source che permette agli sviluppatori Web di creare e gestire complesse applicazioni fronted JavaScript scritte in Java. Il codice sorgente

Java può essere compilato su qualsiasi piattaforma con i file Ant inclusi. I punti di forza di GWT sono la riusabilità del codice, la possibilità di realizzare pagine Web dinamiche mediante le chiamate asincrone di Ajax, la gestione delle modifiche, il bookmarking, l'internazionalizzazione e la portabilità fra i differenti browser.

2.3.1 Architettura GWT 2.7

GWT è caratterizzato da tre componenti base:

1. Un compilatore Java-JavaScript di alta qualità.
2. La Java Runtime Enviroment (JRE) Emulation library.
3. Interfaccia utente

Il compilatore

Il compilatore Java-Javascript è il componente principale di GWT. Si occupa di prendere il codice Java 1.5 e produrre una versione equivalente in JavaScript ed incapsulare la varie differenze tra i browser. Il compilatore esegue numerosi processi di ottimizzazione del codice, debugging, logging e generazione del codice. La generazione del codice JavaScript può essere fatta con uno di questi tre stili:

-Offuscato il codice è illeggibile, compatto e di piccole dimensioni, quindi è raccomandato solamente quando pubblichiamo in produzione la nostra applicazione.

-Formattato codice leggibile

-Dettagliato codice ricco di specificatori di package e signature complete: questo stile è molto comodo per ricavare informazioni del codice Java dalla console degli errori JavaScript.

JRE Emulation library

Contiene le implementazioni in linguaggio JavaScript delle librerie Java standard maggiormente utilizzate(package java.lang.*,java.sql e java.util.*). Gli altri

package(ad esempio java.io.*)sono molto limitati, includendo solo alcune interfaccie. Questa limitazione deriva dal fatto che il codice JavaScript generato da GWT viene eseguito su una sandbox del browser e stampanti.

libreria UI

È una libreria User Interface contenente un insieme di interfacce e classi che permettono di disegnare le pagine web (ad es. bottoni, text boxes, immagini, ecc.). Questa è la libreria standard principale per creare applicazioni web-based basate su GWT.

2.3.2 Comunicazione Client-Server in GWT

È noto che quando si sviluppa una RIA, cosa fondamentale è la comunicazione tra il browser(client) e il server. GWT fornisce differenti strade per comunicare con un server e il formato dei dati da utilizzare dipende dal server con cui si interagisce.

Utilizzando le Remote Procedure Calls(GWT RPC)

GWT RPC è un framework che permette di facilitare il passaggio di oggetti Java tra client e server (e anche viceversa) attraverso il protocollo HTTP. E' possibile utilizzare il framework GWT RPC per rendere trasparenti le chiamate alle servlet Java e lasciare a GWT il compito di prendersi cura dei dettagli di basso livello come la serializzazione degli oggeti. Il meccanismo GWT RPC può esser diviso in tre parti:

1. Il servizio eseguito sul server
2. Il codice client che invoca il servizio
3. Gli oggetti Java trasmessi tra client e server

Figure 2.3: GWT RPC: interazione tra server e client mediante data object

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

- Tipi primitivi Java.
- I wrapper dei tipi primitivi
- Un sottoinsieme degli oggetti Java Runtime Environemnt (JRE)
- Qualsiasi tipo definito dall'utente a patto che sia serializzabile[1] (implementi l'interfaccia Serializable o IsSerializable di GWT)

Ricevendo dati JSON via HTTP

Se l'applicazione comunica con un server che non può ospitare servlet Java, oppure con uno che utilizza già un'altro formato di dati come JSON o XML, si possono eseguire richieste HTTP per ottenere i dati. GWT fornisce classi HTTP generiche che possono essere utilizzate per fare le richieste, e classi XML e JSON client utilizzabili per processare le risposte.

[1] Le chiamate alle GWT-RPC sono tra codice Javascript e Java e GWT prevede la serializzazione come parte del meccanismo RPC

Facendo richieste Cross-Site per JSONP

Se si crea un'applicazione che richiede dati da uno o più web server remoti bisogna evitare le restrizioni SOP(Same Origin Policy)[2].

2.4 Librerie

Apache Batik

Vectomatic lib-gwt-svg

[2]Same Origin Policy è una misura di sicurezza del browser che limita il codice JavaScript client-side nell'interagire con risorse originate da nomi di dominio, porte e protocolli differenti.

Chapter 3

La Webapp GESTIONE AULE

software figo

3.1 Pattern architetturale MVP

Un' importante fase nella progettazione di applicazioni software è quella della scelta di un'opportuna architettura, che definisce le linee guida allo sviluppo del progetto. Definire bene tale processo è utile sia per standardizzare il modello di sviluppo del progetto corrente, sia per le applicazioni future. L'impiego di un adeguato pattern porta numerosi vantaggi tra cui:

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

Il Model-View-Presenter (MVP) è un pattern architetturale usato principalmente quando si vogliono creare User Interfaces. Il pattern MVP Separa la parte di gestione dei dati di un'applicazione dalla loro visualizzazione e manipolazione attraverso l'interfaccia utente.

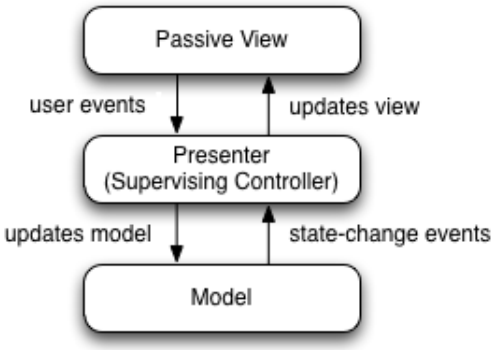

Figure 3.1: Diagramma del Model-View-Presenter MVP

Questo pattern è composto da tre elementi:

Model è un'interfaccia che definisce i dati da visualizzare ed incapsula lo stato dell'applicazione.

View è un'interfaccia responsabile della visualizzazione dei dati e delle informazioni, raccoglie gli input dell'utente e mai la manipolazione dei dati avviene in maniera diretta ma sempre attraverso un'interfaccia. Questo tipo di approccio consente di gestire facilmente eventuali modifiche alla GUI (Graphical User Interface) che non richiederanno mai l'aggiornamento del presenter.

Presenter è colui che, oltre ad aggiornare la vista, interagisce con il model,che può essere identificato sia come lo stato di un oggetto che come dati persistenti in un'applicazione, in base alle richieste ricevute dalla view.

Nelle viste vi è una relazione uno a uno con i presenters e ciò permette a questi di osservare le loro viste e reagire agli eventi. Siccome le viste non possono interagire direttamente con altre viste, i presenters devono scambiare i dati con gli altri presenters. Per ottenere tale comunicazione fra presenters, si fa uso dell' Event Bus, un oggetto in grado di trasmettere e filtrare le notifiche(figura: 3.2).

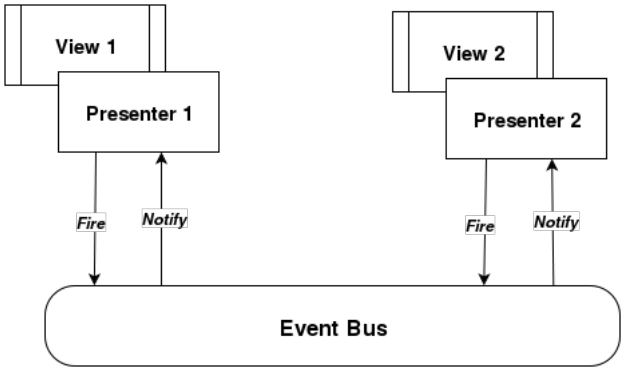

Figure 3.2: Comunicazione tra unità del software che non sono direttamente collegate tra loro. Ognuna di esse può inviare eventi all'Event Bus ed essere in "ascolto" di particolari eventi.

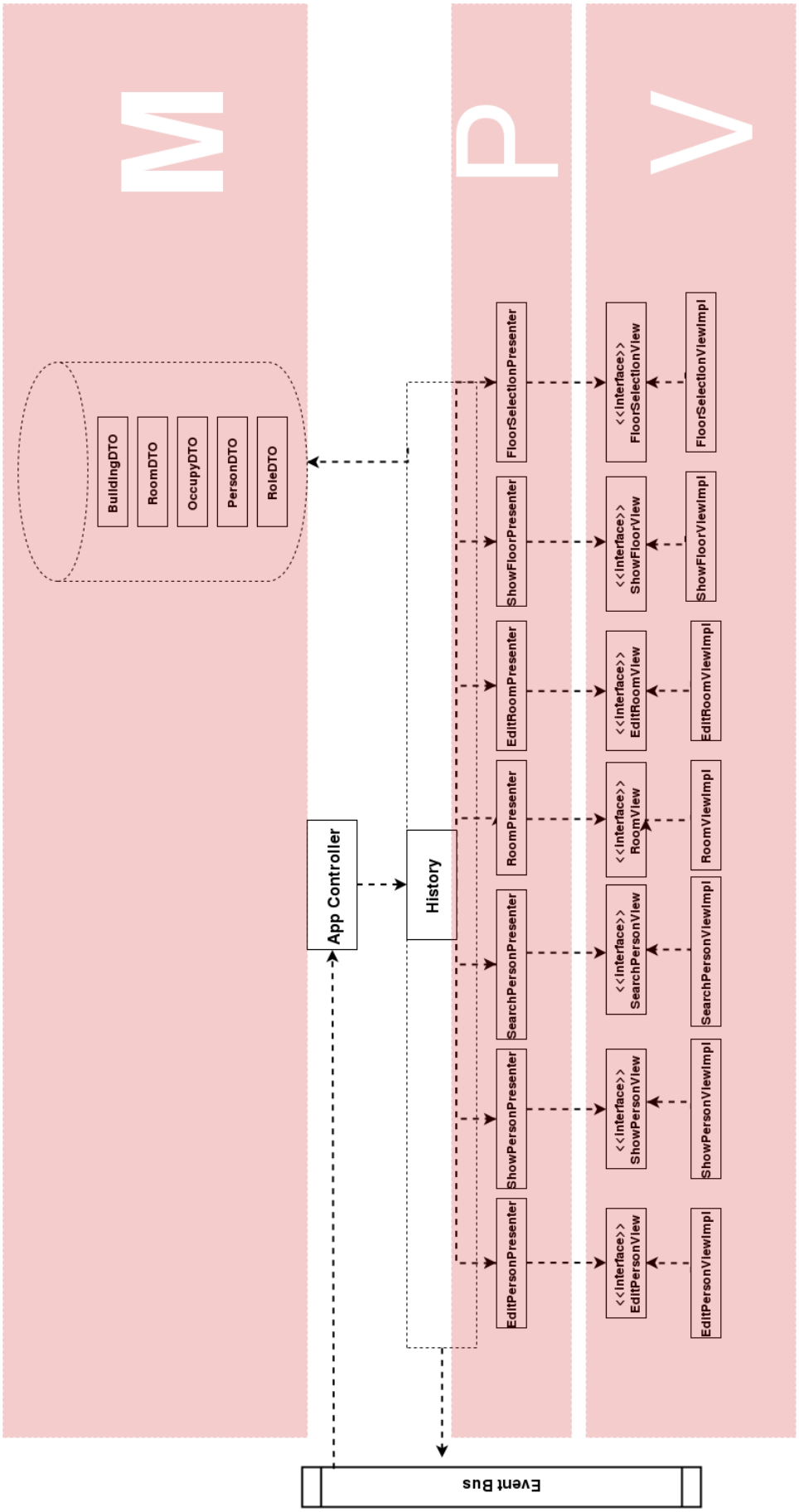

Figure 3.3: Implementazione client seguendo pattern MVP

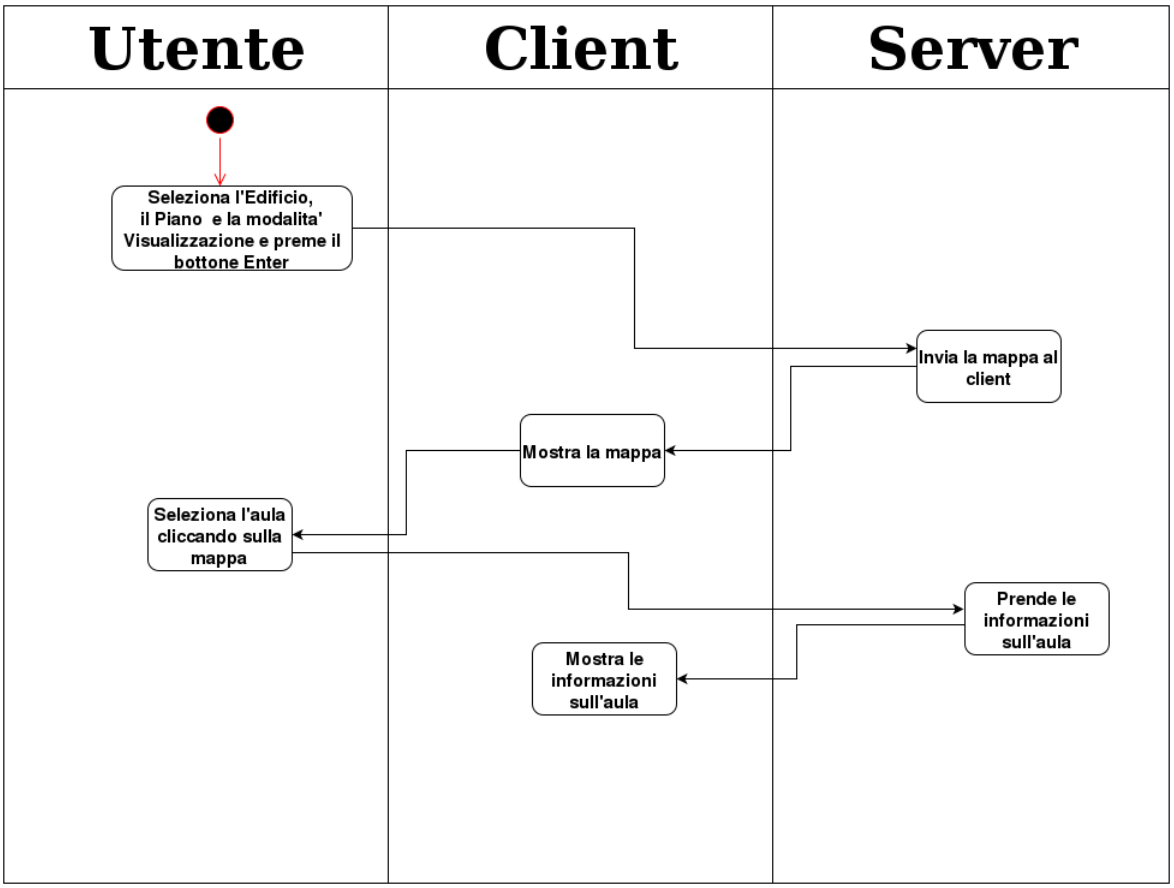

Figure 3.4: Activity diagram della funzione di visualizzazione di un'aula.

Chapter 4

L'applicazione MONITORAGGIO AULE

4.1 Stile architetturale REST

Il Web Ã¨ popolato da articoli e trattazioni relative alle architetture REST che tristemente ripetono le medesime battute. Pertanto, al fine di minimizzare la monotonia, in questo paragrafo ci limiteremo a ricordare brevemente i concetti fondamentali, riducendo al minimo la ripetizione delle storielle ormai usurate. Lâacronimo REST, REpresentational State Transfer (âtrasferimento dello stato di rappresentazioneâ) deriva dalla tesi di dottorato di Roy Fielding intitolata âArchitectural Styles and the Design of Network-based Software Architecturesâ (âstili architetturali e progettazione di architetture software basate sul networkingâ) [2]; concediamoci una battuta ricorrente: per essere una tesi di dottorato, non Ã¨ poi cosÃ¬ $difficile da leggere. Fielding è uno dei principali autori di del protocollo HTTP, HyperText Tra$

REST non Ã¨ unâarchitettura bensÃ¬ $un o stile architetturale formato da vincoli, linee guida e best practice.$

Con questo assunto in mente, Fielding ha iniziato la sua esplorazione nel dominio degli stili delle architetture distribuite partendo dal limite inferiore da lui definito âspazio nulloâ (null space), ossia il âFar Westâ, rappresentato da organizzazioni dotate di sistemi a basso grado di maturitÃ in cui tutte le risorse tecnologiche sono disponibili, tutti gli stili sono ammessi, senza regole neâ limiti. Continuando lungo la direttrice evolutiva, Fielding ha poi definito lo

stato di totale maturitÃ caratterizzato da sistemi che rispettano le regole da lui definite e che quindi possono essere definiti compatibili con le guideline REST (RESTful). Queste regole sono condensate nei seguenti sei vincoli, di cui i primi cinque sono obbligatori, mentre lâultimo Ã¨ facoltativo. Vediamo quindi i 5+1 vincoli cui un sistema deve sottostare per essere definito RESTful.

Chapter 5

Conclusioni

conclusioni LITERATURE CITED

Bibliography

[1] Christian Bauer, Gavin King; *Java Persistence with Hibernate*

[2] Adam Tacy, Robert Hanson, Jason Essington, Anna Tokke; *GWT in Action*