UNIVERSITÀ DEGLI STUDI
DI MODENA E REGGIO EMILIA

Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Progettazione e Sviluppo di un Sistema di Risposta Automatico per la Richiesta di Informazioni Riguardanti i Servizi Ferroviari

Bedogni Davide
Tesi di Laurea

Relatore:
Prof. Riccardo Martoglia

Anno Accademico 2013/2014

RINGRAZIAMENTI

Ringrazio la mia famiglia e tutti coloro che in questi anni mi hanno sostenuto in questo percorso.
Inoltre ringrazio Marcello Pellacani e l'Ing Riccardo Martoglia per la disponibilità e professionalità con cui mi hanno seguito nella realizzazione della tesi.

Indice

Introduzione **p.7**

1 Basi di NLP **p.9**
1.1 Processo di disambiguazione p.10

2 Cogito Studio **p.11**

2.1 Introduzione p.11
2.2 Potenzialità p.12
2.3 Sensigrafo p.12
2.4 Processo di analisi p.13
2.5 Categorizzazione p.14
2.5.1 Applicazioni di regole e porzioni di testo p.14
2.5.2 Linguaggio C p.15
2.5.3 Scope, estensioni e dominio p.15
2.5.4 Espressione p.17
2.6 Estrazione p.20
2.6.1 Definizione di un template p.21
2.6.2 Individuazione ed estrazione dei dati p.22
2.6.3 Trasformazioni p.23
2.7 Linguaggio “D” p.25

3 ESSEX **p.27**

3.1 Introduzione p.27
3.2 Componenti p.28
3.3 Richiesta di analisi p.30

4 Analisi del progetto **p.33**

4.1 Obiettivo del progetto p.33
4.2 Idea di risoluzione p.34
5 Creazione ambiente in COGITO **p.37**

5.1 Fase di Categorizzazione p.37
5.2 Fase di Estrazione p.44
5.3 Risultati di elaborazioni di esempio su Cogito Studio p.47

6 Implementazione p.49

6.1 Introduzione p.49
6.2 Apache e Tomcat p.50
6.2.1 Perché Apache e Tomcat p.50
6.2.2 Configurazione di Tomcat p.51
6.3 Classi Java p.52
6.3.1 Classe “BaseServlet” p.52
6.3.2 Classe “LeggiInput” p.53
6.3.3 Classe “CallFormNew” p.57
6.3.4 Classe “SingleThreadEx” p.61
6.3.5 Classe “Scheda” p.61
6.3.6 Classi del Package Beans p.62

7 Template con Velocity p.65

7.1 Perché Velocity? Principali caratteristiche p.65
7.2 Configurazione Velocity p.66

8 Test e statistiche p.67

8.1 Test p.67
8.2 Statistiche p-69

9 Conclusione e sviluppi futuri p.71

APPENDICE A : Codice delle classi in Java p.73
APPENDICE B : Template Velocity p.90
APPENDICE C : File web.xml per la configurazione di Tomcat p.93
APPENDICE D : Domande utilizzate per il test dell’applicazione p.94

SOFTWARE UTILIZZATO p.95
BIBLIOGRAFIA p.96

Introduzione

Questa tesi prende in analisi lo sviluppo di un'applicazione che risponda a delle domande poste dagli utenti in linguaggio naturale. Le domande riguarderanno, in particolare, i servizi ferroviari.
Sarà trattata con attenzione la parte relativa all'implementazione del software e i relativi strumenti utilizzati allo scopo, ma verrà anche dedicata una sezione più teorica che riguarderà le basi dell'NLP (Natural Language Processing).

Per raggiungere gli obiettivi fissati ci si è serviti della collaborazione con l'azienda Expert System SPA (www.expertsystem.com), leader nel campo della tecnologia semantica, che ha fornito alcuni strumenti indispensabili alla riuscita del progetto tra cui il motore semantico "ESSEX" ed il relativo tool di sviluppo "COGITO Studio".

La prima parte della tesi tratterà le basi del NLP e verranno analizzati gli strumenti forniti da Expert System da un punto di vista teorico, in modo da avere poi un'idea più chiara quando verrà descritta la fase implementativa.
La tesi prosegue con la descrizione dello sviluppo del software in tutte le sue fasi.
La terza parte invece va a concludere il discorso con l'analisi dei risultati ottenuti dopo dei test su un campione di utenti e le ipotesi sui possibili sviluppi futuri dell'applicazione.

La struttura del documento è la seguente:

- Il capitolo 1 tratta le basi del NLP.
- I capitoli 2 e 3 trattano, in maniera teorica, gli strumenti forniti da Expert System, ovvero COGITO Studio ed ESSEX.
- Il capitolo 4 introduce alla risoluzione vera e propria del problema tracciando le caratteristiche che dovrà avere l'applicazione e le fasi che portano allo sviluppo finale del software.
- I capitoli 5, 6 e 7 descrivono l'implementazione dell'applicazione, in particolare si vedranno nel capitolo 5 l'utilizzo di COGITO Studio finalizzato all'obiettivo, nel capitolo 6 lo sviluppo dell'applicazione vera e propria e di come si interfaccia sul web, infine nel capitolo 7 sarà descritto il template web che permetterà di visualizzare le risposte.
- I capitoli 8 e 9 concluderanno la tesi con le analisi dei risultati dei test eseguiti sull' applicazione da un campione di utenti ed alcune ipotesi sui possibili sviluppi futuri.

Capitolo 1
Basi di NLP

Avere un sistema di riconoscimento del linguaggio naturale è molto importante perché consente di automatizzare dei processi sui documenti che altrimenti dovrebbero essere svolti dall' uomo. Permette quindi di porsi, nei confronti di testi e documenti, come un vero e proprio essere umano in grado non solo di tradurre il documento in una serie di keyword, come fanno ad esempio i motori di ricerca, ma anche di capire il contenuto del documento.
La difficoltà principale di questo processo è la disambiguazione, ovvero il riuscire a riconoscere il diverso significato di parole identiche posizionate però all' interno di contesti diversi, oppure riconoscere se, ad esempio, un verbo si riferisce alla proposizione principale o ad una secondaria.

Un esempio di disambiguazione di termini può essere notato nell' uso del termine "rombo" nelle due frasi:

1. *Ho acquistato un rombo al mercato*
2. *Il rombo è una figura geometrica*

"rombo" assume un significato diverso: un pesce nella prima, una figura geometrica nella seconda.

1.1 Processo di Disambiguazione

Il processo mediante il quale si compie la disambiguazione è composto di più fasi:

1. **Analisi lessicale:** In questa fase si scompone un'espressione linguistica in token. Con token si intende, in analisi lessicale, la più piccola porzione di testo che ha un senso anche presa singolarmente.
2. **Analisi grammaticale:** Si associa ad ogni parola del testo il relativo discorso a cui è collegata semanticamente.
3. **Analisi sintattica:** Si strutturano i token ottenuti dall' analisi lessicale in una struttura ad albero (detta *parse tree*).
4. **Analisi semantica:** Ad ogni elemento della struttura sintattica ricavata nel punto 3 viene associato il significato del singolo elemento. Così facendo ottengo il significato dell'intera espressione linguistica.

Oggigiorno è sempre più importante avere un sistema di riconoscimento del linguaggio naturale perché le informazioni, nelle aziende, sul web, sono sempre in aumento e uno strumento che permette di automatizzare, categorizzare e comprendere il contenuto di queste informazioni può fare la differenza riducendo i costi e anche aumentando i profitti se utilizzato, ad esempio, come strumento per indagare il mercato e il livello di soddisfazione dei clienti.

Capitolo 2
Cogito Studio

2.1 Introduzione

Cogito Studio è un sistema software realizzato dall'azienda Expert System che permette di risolvere molti problemi relativi all' analisi dei testi.
Il suo funzionamento è basato sull' analisi semantica e permette di definire una serie di regole che consentiranno di analizzare ed elaborare il testo fornito ed infine ottenere le risposte cercate.
Le regole sono scritte in tre linguaggi differenti, ognuno con un compito diverso:

- **Linguaggio C (file *.efo):** Permette di associare parti del documento ad una specifica categoria di appartenenza.
- **Linguaggio D (file *.efd):** Effettua una manipolazione sui domini (categorie appartenenti all' albero dei domini) o su un set di domini.
- **Linguaggio E (file *.efe):** Tramite questo linguaggio si effettuano le estrazioni di informazioni contenute nei documenti.

2.2 Potenzialità

Cogito Studio fornisce una base per effettuare svariati tipi di operazioni. Alcuni esempi delle attività che si possono svolgere tramite Cogito Studio sono:

- **Categorizzazione di documenti:** Quando si hanno molti documenti può essere utile suddividerli per argomento al fine di trovare più facilmente le informazioni ricercate.
- **Ricerca informazioni in un testo:** Tramite Cogito Studio posso analizzare un documento nella sua completezza e quindi suddividerlo in categorie, cercare delle informazioni al suo interno, analizzare la presenza di parole chiave e anche schematizzare le informazioni ottenute in diagrammi.
- **Percepire il "Sentiment":** È possibile analizzare un documento e capire il tono con cui ci si riferisce ai temi presenti nel testo. Ad esempio si può verificare se un documento è scritto in tono dispregiativo, minaccioso, di apprezzamento o scherzoso. Il tono è anche detto *Sentiment* del documento.
- **Risposta automatica a domande poste in linguaggio naturale:** Questa caratteristica è quella che verrà più analizzata nella tesi e riguarda la capacità di Cogito Studio di riconoscere una domanda posta in linguaggio naturale e comunicare la conoscenza ad un sistema esterno che elaborerà le informazioni in modo tale da poter restituire una risposta esauriente alla domanda posta dall' utente.

2.3 Sensigrafo

Il Sensigrafo è un componente di capitale importanza in Cogito, in quanto rappresenta la maggior parte della sua "conoscenza". Il Sensigrafo è un insieme di reti semantiche interconnesse tramite una struttura a grafo in cui ogni nodo rappresenta un concetto.
Tramite il Sensigrafo posso avere informazioni riguardanti: interconnessione tra concetti, specifiche sull' appartenenza o meno di un certo elemento ad un certo dominio lessicale ed infine posso avere informazioni sulla frequenza di utilizzo di un certo concetto.
I concetti all' interno del Sensigrafo sono rappresentati univocamente da un numero intero detto *syncon*. Va specificato che *syncon* diversi rappresentano concetti diversi, ma possono avere la stessa successione di caratteri. Ad esempio la parola "stato" può riferirsi sia a "stato" inteso come nazione, ma anche a "stato" in senso come participio

passato del verbo stare; in questo caso le due parole saranno identiche, ma avranno *syncon* diversi.

2.4 Processo di Analisi

Il processo di analisi mediante il quale viene analizzato un documento è composto da specifiche operazioni:

1. **Attivazione:** Il motore software viene inizializzato utilizzando, inizialmente, una specifica gerarchia di domini. Successivamente il motore viene attivato caricando dinamicamente alcuni strumenti semantici (disambiguatore e Sensigrafo)
2. **Compilazione delle regole:** Dopo aver attivato il motore semantico e caricato gli strumenti necessari posso compilare le regole.
3. **Analisi del documento:** Si sottopone il testo da analizzare al motore semantico.
4. **Trasformazione di formato:** Se è stato utilizzato un formato di testo particolare, ad esempio word, questo è trasformato in testo ASCII che sarà più semplice da elaborare.
5. **Trasformazione in blocchi logici:** Il testo viene suddiviso in blocchi logici, ad esempio: titolo, paragrafo, frase, eccetera…
6. **Normalizzazione:** Il testo viene normalizzato in modo da eliminare ad esempio i caratteri superflui (spazi ripetuti) oppure si trasformano le maiuscole in minuscole. Questo permette di avere un formato più facilmente lavorabile.
7. **Disambiguazione:** Il testo viene trasformato in una struttura che consenta l'applicazione di regole di categorizzazione ed estrazione.
8. **Implementazione regole:** Le regole vengono applicate al testo e vengono assegnati i punteggi ai domini a seconda di quanto una certa regola sia rispettata o meno.
9. **Selezione domini:** In questa fase vengono elaborate le risposte e si applicano le regole del linguaggio "D".

10. **Estrazione di informazioni:** Si estraggono le informazioni specificate nelle regole poste tramite il linguaggio “E”.

2.5 Categorizzazione

La categorizzazione è l’operazione mediante la quale vengono selezionate porzioni di testo e raggruppate secondo una serie di domini definiti tramite le regole in linguaggio “C”.

2.5.1 APPLICAZIONI DI REGOLE E PORZIONI DI TESTO

Prima di procedere nel descrivere le basi del linguaggio C e le operazioni che si possono svolgere mediante esso è bene individuare le porzioni di testo su cui le regole sono applicate.
Il testo è infatti suddiviso in diverse porzioni a differenti livelli gerarchici che saranno poi connesse tra loro tramite delle relazioni. La porzione di codice su cui si applicano le regole prende il nome di *scope*.

Le porzioni di testo possono essere:

- **Documento:** L’intero testo da analizzare.
- **Sezione:** Definite nel file di configurazione.
- **Paragrafo:** Porzione di testo terminata da un punto e a capo.
- **Frase:** Porzione di testo terminato con un punto, punto interrogativo o punto esclamativo.
- **Periodo:** Porzione di una frase che ha di per sé un senso compiuto; non deve necessariamente essere continua e ci dev’essere la presenza di almeno un predicato.
- **Gruppo:** Porzione di periodo con un ruolo logico, ovvero che comprende un soggetto, un predicato e un complemento.

- **Parola:** Singolo elemento di una frase che corrisponde ad un lemma del Sensigrafo.
- **Atomo:** Porzione di testo che non può essere ulteriormente suddivisa. Non è necessario che abbia significato logico.

2.5.2 LINGUAGGIO C

Il linguaggio C di Cogito permette di scrivere le regole per la categorizzazione dei documenti, in modo che ad ogni determinato segmento di testo sia assegnato un punteggio che indichi l'appartenenza o meno ad un determinato dominio.

Una regola generica scritta in linguaggio C deve avere una forma del tipo:

```
SCOPE scope { DOMAIN (nomedominio[:score]){ EXPRESSION}}
```

Ora andremo ad analizzare ognuna delle parti fondamentali di questa generica regola.

2.5.3 SCOPE, ESTENSIONI E DOMINIO

Con la definizione dello *scope* definisco l'ambiente entro il quale la regola può essere inizializzata.
Oltre allo *scope* potrei inserire anche un'estensione che indica invece la porzione del documento in cui la regola deve essere rispettata per produrre un'istanza valida (la dimensione dell'estensione dev'essere minore o uguale a quella dello *scope*).

La regola con l'aggiunta dell'estensione si scrive in questo modo:

```
SCOPE scope IN estensione{ ... }
```

Lo *scope* e le estensioni come abbiamo detto indicano sostanzialmente una certa porzione di testo, in particolare posso indicare come *scope* o estensioni:

- **Sentence:** Ricerca nella frase.
- **Paragraph:** Ricerca nel paragrafo.
- **Group:** Ricerca in un gruppo.
- **Clause:** Ricerca in una proposizione.

Affinché la regola svolga il suo compito di categorizzazione devo indicare in quale dominio voglio caratterizzare la porzione di testo che rispetta la regola. Indico quindi dopo "DOMAIN" il nome del dominio a cui voglio assegnare la regola.
Facoltativamente è possibile assegnare anche un peso alla regola, ovvero il punteggio che assegno al dominio al verificarsi della regola. Il punteggio assegnato dipenderà dal parametro utilizzato, che potrà essere:

- **Normal:** Si assegna punteggio 10 al dominio
- **High:** Si assegna punteggio 15 al dominio
- **Low:** Si assegna punteggio 3 al dominio
- **Select:** Si assegna punteggio infinito al dominio. In pratica se la regola è verificata allora la porzione di testo sarà caratterizzata automaticamente nel dominio.
- **Discard:** Si assegna punteggio meno infinito al dominio. Se la regola quindi è rispettata scarto il dominio a prescindere.

La categorizzazione avviene sommando i punteggi di tutte le regole in modo tale da avere, per ogni dominio, un punteggio che indica se la porzione di testo a cui sono state applicate appartiene in misura maggiore o minore ad una certa categoria.

2.5.4 ESPRESSIONE

Le regole in linguaggio C solitamente descrivono dei prototipi di sequenza di testo che devono coincidere con il testo che si vuole verificare. Per scrivere una sequenza si

ricorre a degli attributi, a degli operatori di sequenza che permettono di stabilire un collegamento tra gli attributi ed infine si può anche ricorrere a degli operatori logici.

Gli attributi che permettono di verificare il matching tra la regola e una porzione di codice e sono i seguenti:

Attributo	Descrizione
KEYWORD	Identifica un *token* tramite una sequenza precisa di caratteri. Viene confrontato quindi un pezzo di testo con una sequenza di caratteri compresi tra apici. Una *keyword* può essere ad esempio "casa","albero","gomma", ecc… Se i caratteri della *keyword* sono tutti minuscoli non verrà data importanza alla presenza di sequenze eterogenee di caratteri maiuscoli e minuscoli nel testo da controllare.
LEMMA	Identifica un *token* in base alla sua forma base, quindi nel caso di un verbo si utilizzerà la sua forma infinita, mentre nel caso di un nome si utilizzerà il singolare. Se, ad esempio, utilizzo la regola LEMMA("casa") verrà considerato valido il match con "case" anche se "case" è al plurale.
ANCESTOR	Viene indicato il numero identificativo di un certo concetto all' interno del Sensigrafo. Ad esempio con ANCESTOR(1400) si individua il concetto di "casa","abitazione", quindi il match sarà corretto se nel testo comparirà una parola che farà riferimento concettualmente a quell' ambito: "dimora" apparterrà a quel concetto e il matching avrà esito positivo, se invece si sta parlando di baseball ad esempio, il termine "casa" farà riferimento a un concetto diverso e quindi il matching avrà esito negativo.
LIST	Il testo viene confrontato con una lista di termini personalizzata. Se la porzione di testo da analizzare appartiene alla lista allora il matching è positivo.
TYPE	Il matching sarà positivo se una porzione di testo sarà del tipo identificato dal TYPE. Ad esempio TYPE(ART) darà esito

	positivo se la porzione di testo sarà un articolo. Posso cercare in particolare: o ART: Articolo; o PRE: Preposizione; o AGG: Aggettivo; o SOST: Sostantivo; o NPR: Nome proprio; o VER: Verbo; o AUX: Verbo ausiliario; o AVV: Avverbio; o PRO: Pronome; o CON: Congiunzione; o NPH: Nome proprio; o PNT: Segno di punteggiatura; o ENT: Sostantivo associato ad una categoria definita in fase di configurazione.
SYNCON	Indentifica un *token* in base al significato che gli è stato assegnato dal disambiguatore semantico.
ROLE	Identifica un *token* in base al suo ruolo logico nella frase. I ruoli possono essere: o OBJECT: Oggetto diretto; o SUBJECT: Soggetto; o NOMINAL_P: Predicato nominale; o VERBAL_P: Predicato verbale; o COPULA: Elemento di congiunzione; o INDIRECT: Oggetto indiretto; o OTHER: Altro ruolo;
POSITION	Indentifica un *token* in base alla sua posizione all' interno del documento. I valori possibili sono: o BEGIN SENTENCE: Il token è all'inizio della frase; o END SENTENCE: Il token è alla fine della frase; o BEGIN PARAGRAPH: Il token è all'inizio di un paragrafo;

	o END PARAGRAPH: Il token è alla fine di un paragrafo; o BEGIN SECTION: Il token è all'inizio del documento; o END SECTION: Il token è alla fine del documento;

Tabella 2.1 – Attributi nel Linguaggio C

Gli attributi da soli però non bastano a creare regole efficaci: è necessario utilizzare degli altri operatori, ovvero le sequenze, che permettono di collegare tra loro gli attributi.

Le sequenze permettono di verificare che gli attributi si susseguano l'un l'altro in diversi modi: ad esempio se voglio accertarmi che una frase sia del tipo "La casa è…" posso verificare che ad un TYPE(ART) faccia seguito un ANCESTOR(1400) a cui a sua volta faccia seguito un TYPE(VER). Queste sequenze in linguaggio C si indicano tramite dei simboli particolari:

Operatore	**Descrizione**
>> <<	Indica una sequenza diretta (gli attributi non devono avere nulla che li separi)
> <	Sequenza debole. Gli attributi possono essere separati da segni "dal debole valore semantico" come avverbi e virgole.
< >	Sequenza con qualsiasi numero di elementi tra due attributi. È anche possibile specificare un limite superiore ed inferiore di elementi compresi tra i due attributi in questo modo: <min:max>
&SV **&VS**	Sequenza soggetto-verbo.
&SO **&OS**	Sequenza soggetto-oggetto.
&VO **&OV**	Sequenza verbo-oggetto.
&SS	Sequenza soggetto-soggetto.
&> **&<**	Legame di collegamento logico

Tabella 2.2 - Operatori di sequenza in Linguaggio C

È bene ricordare per chiarezza che i simboli di sequenza sono da intendere applicati ad una coppia di attributi.

Gli ultimi elementi utili per definire le regole per le espressioni sono gli operatori logici. Essi permettono di definire una regola valida al verificarsi di specifiche combinazioni di più espressioni.
Gli operatori logici sono:

Operatore	Descrizione
NOT	Non presenza di un attributo.
AND	Presenza contemporanea di più attributi validi o di più espressioni valide.
OR	Presenza alternativa o contemporanea di più attributi o espressioni valide.
XOR	Presenza alternativa e non contemporanea di più attributi o espressioni valide.

Tabella 2.3 - Operatori logici in Linguaggio C

2.6 Estrazione

L'estrazione è l'operazione che consente di ricavare informazioni contenute nel testo e memorizzarle in strutture dati. Il processo di estrazione generalmente consiste, in realtà, di 4 sotto processi che insieme prendono il nome di **operazioni ETL**. Questi sotto processi sono:

1. **Identificazione (fase I)**: Si individuano i dati non strutturati presenti nei documenti che sono sottoposti ad analisi.
2. **Estrazione (fase E)**: Si riconoscono e si estraggono i dati.
3. **Trasformazione (fase T**): I dati estratti possono essere eventualmente manipolati e quindi posso decidere di memorizzarli in diversi modi.
4. **Caricamento (fase L)**: I dati vengono trasferiti al client e memorizzati in strutture apposite.

In questo processo le prime due fasi (identificazione ed estrazione) sono regolate dal linguaggio “E”; in particolare il processo definito dalle regole in linguaggio “E” è così strutturato:

1. Definizione dei template: Si definiscono i template, ovvero, le strutture date designate alla memorizzazione delle informazioni estratte.
2. Individuazione dei dati e classificazione: Tramite apposite regole scritte tramite estensioni del linguaggio “C”, visto in precedenza, si riconoscono i dati di interesse nel testo e li classifico.
3. Estrazione dei dati: Vengono riempiti i campi definiti al passo 1 con i dati estratti al punto 2.

Vediamo ora più in dettaglio come il linguaggio “E” agisce sulle tre fasi sopra descritte.

2.6.1 DEFINIZIONE DI UN TEMPLATE

Definire un template significa definire una struttura dati che può contenere uno o più valori. Può essere visto come una tabella avente una o più righe e colonne dove le colonne sono i tipi di informazioni estratte, mentre le righe possono essere viste come record di un database.

La sintassi di definizione di un template è la seguente:

```
TEMPLATE ( nome )
{
    @nome-field1
    @nome-field2
    [regole di attrazione]
    [regole di merging]
}
```

che creerà una tabella del tipo:

nome	
nome-field1	nome-field2

Inoltre dopo i nomi dei campi che si andranno a creare posso inserire tra parentesi tonde degli attributi opzionali, che sono:

- **Attributo N (Necessary):** Un' estrazione è valida solo se tutti i campi del suo template sono associati ad un valore estratto.
- **Attributo C (Cardinal):** Se due campi con attributo C hanno lo stesso valore posso fonderli tra loro.
- **Attributo V (Validating):** Al termine dell'elaborazione i record che appariranno saranno quelli che per ogni campo con attributo V avranno un valore non nullo.
- **Attributo S (Solitary):** Il campo è considerato solo se corrispondono anche le posizioni.

Le regole di attrazione sono regole che indicano come e quando due campi separati possono essere aggregati in un record unico.
Le regole di merging invece definiscono se e quando il corrispondente template può generare record aggregati.

2.6.2 INDIVIDUAZIONE ED ESTRAZIONE DEI DATI

Le regole che determinano l'individuazione e l'estrazione dei dati sono simili alle regole di categorizzazione.

```
SCOPE scope
{
   IDENTIFY (nome)
   {
     condizioni e associazioni ai field
    }
}
```

SCOPE come visto nella categorizzazione indica l'area di testo da valutare (paragrafo, frase, periodo, eccetera…).
IDENTIFY invece, identifica il template (definito in precedenza) che si andrà a riempire con i valori dei campi estratti.

Le condizioni e le associazioni dei valori ai field rispettivi sono le parti salienti del processo di categorizzazione. Per estrarre un'informazione dal testo devo per prima cosa riuscire a riconoscerla e poi indicare il campo in cui salvarla.

Per riconoscere un'informazione utilizzo le condizioni viste nel linguaggio "C" per la categorizzazione, mentre per estrarre l'informazione riconosciuta ed inserirla nel field corrispondente utilizzo il carattere "@" seguito dal nome del field in cui voglio salvare il valore. Ad esempio *@nome-field1[TYPE(NPH)]* salverà il nome proprio di persona riscontrato all' interno del field "nome-field1".
Posso anche in questo caso utilizzare le sequenze e quindi se ho ad esempio:

```
KEYWORD("Phone") > @TELEPHONE[PATTERN("[+]?([\d]|[\s])+")]
```

inserisco nel campo Telephone i valori che rispettano il pattern dato e che seguono la parola "Phone".

2.6.3 TRASFORMAZIONI

Abbiamo prima accennato a delle possibili trasformazioni che agiscono sui dati estratti, analizziamole ora più in dettaglio.

Le trasformazioni sono opzionali nella stesura delle regole e consentono di controllare, dopo l'estrazione di un token, quali dati vengono effettivamente memorizzati nei campi del template.

La sintassi con cui si fa uso delle trasformazioni è la seguente:

```
SCOPE scope
{
 IDENTIFY(nometemplate)
 {
  @nome-field1[ ATTRIBUTO ] | [ TRASFORMAZIONE ]
 }
}
```

dove per **ATTRIBUTO** si intende una delle condizioni viste in precedenza (keyword, lemma, syncon, eccetera…), mentre per **TRASFORMAZIONE** si utilizza una delle seguenti opzioni:

Trasformazione	Descrizione
ATOM	Viene estratta la forma base dell'atomo. Quindi se viene individuato un *token* composto ad esempio "emergency team" viene estratto solo l'atomo "emergency".
BASE	In questo caso viene estratta la forma base del *token*.
ENTRY	Viene estratta la forma base ritenuta più calzante a seconda del contesto in cui si trova.
SYNCON	Di un dato trovato si estrae il relativo identificativo del Sensigrafo.
TEXT	Viene restituita la forma come riportata nel testo.
TOKEN	Viene restituita la forma del *token* le testo. Simile alla trasformazione TEXT, ma comprende un livello logico più ampio.
GROUP	Estrae l'intero gruppo di parole in cui è contenuto il *token*. Per gruppo si intende un insieme di parole che hanno un ruolo preciso all' interno di una frase.
CLAUSE	Estrae l'intero periodo in cui è contenuto il *token*.
SENTENCE	Estrae l'intera frase in cui è contenuto il *token*.
PARAGRAPH	Estrae l'intero paragrafo in cui è contenuto il *token*.
SEGMENT	Estrae l'intero segmento in cui compare il *token*. Bisogna prima definire il segmento da prendere in considerazione.
SECTION	Estrae l'intera sezione in cui è contenuto il *token*.
EXTENSION	Estrae l'estensione che comprende il *token*.
SEQUENCE	Estrae l'intera sequenza contenente il *token*. Da utilizzare ovviamente solo nel caso in cui le sequenze siano realmente utilizzate nella ricerca del *token*.
SECTOR	La sequenza è estesa per includere completamente il

	gruppo iniziale e quello finale.

Tabella 2. 4 - Estrazioni in Linguaggio E

2.7 Linguaggio “D”

Il linguaggio “D” è il linguaggio che permette di operare modifiche sui set di domini. Abbiamo visto come con il linguaggio “C” sia possibile categorizzare le informazioni estratte tramite linguaggio “E” all’ interno dei domini. Ora è possibile modificare i domini in modo da rendere il set più consono all’applicazione finale. In particolare tramite il linguaggio “D” posso agire su due tipi di variabili: i set e i domini, dove il dominio è un singolo nodo dell’albero dei domini risultanti dalla categorizzazione, mentre un set dei domini contiene una lista di domini.

Essendo linguaggio “D” non presente nel progetto che si discuterà nel seguito della tesi non si descriveranno i dettagli del linguaggio e come si utilizza. Si tenga però in considerazione il fatto che il linguaggio “D” permette di agire sui set e sui domini eseguendo su di essi alcune operazioni come ad esempio, aggregazione, eliminazione, modifica, eccetera… L’obiettivo è quello di ottenere così un output migliore per l’applicazione che dovrà utilizzare poi i dati forniti da Cogito.

Capitolo 3
ESSEX

3.1 Introduzione

ESSEX è un acronimo che sta per **Expert System Semantic Engine eXtended Server**.

Se COGITO Studio permette di scrivere delle regole e dei meccanismi per l'analisi semantica, ESSEX invece è il vero e proprio motore che consente di elaborare le informazioni e fornire un output utilizzabile poi in diversi contesti. Quindi ESSEX può essere visto come un punto di accesso alle tecnologie semantiche personalizzate con il tool COGITO Studio.

Genericamente si può dire che ESSEX prende in input file XML ed esegue le elaborazioni regolamentate dai linguaggi "C" ed "E" di Cogito e fornisce in output un file XML contenente le categorizzazioni, le estrazioni effettuate ed altre possibili informazioni.

Il file XML di ingresso contiene il testo da analizzare e la modalità di analisi che si intende eseguire su di esso.

3.2 Componenti

L’architettura su cui è basato ESSEX è formata da due componenti principali:

- **GSL Server**
- **GSL Dispatcher**

(GSL è un acronimo che significa General Semantic Layer)
Il GSL Server è il responsabile principale del lavoro svolto da ESSEX: si può dire che sia il vero e proprio “motore semantico” del sistema.
Le operazioni che si possono svolgere devono essere indicate nel file XML in input; per ogni tipologia di operazione è assegnato un comando corrispondente:

Comando	Operazione corrispondente
CRL_RESET	Si resetta il motore semantico, le regole e le librerie vengono caricate nuovamente. Si utilizza per aggiornare il motore semantico qualora siano state effettuate delle modifiche.
CRL_UNLOAD	Dealloca il motore semantico.
CRL_LOAD	Alloca il motore semantico. Si utilizza spesso dopo il comando CRL_UNLOAD per ricaricare il motore una volta che è stato de allocato.
CRL_COMPILE	Consente di compilare le regole del server senza dovere deallocare tutto il motore semantico. È utilizzato quando si effettuano modifiche solo sulle regole e si vuole renderle effettive.
CRL_ANALYZE	È il comando più frequentemente utilizzato in quanto è quello che consente di eseguire l’analisi di un documento.

Tabella 3. 1 - Elenco possibili operazioni

Ogni GSL Server dovrà avere un numero identificativo, un indirizzo e una porta a cui collegarsi per comunicare con esso.
Di seguito si trova un’immagine che rende più chiaro il processo dal documento iniziale all’ output in XML fornito da ESSEX. Nel riquadro di ESSEX agiranno sia il GSL Dispatcher che il GSL Server.

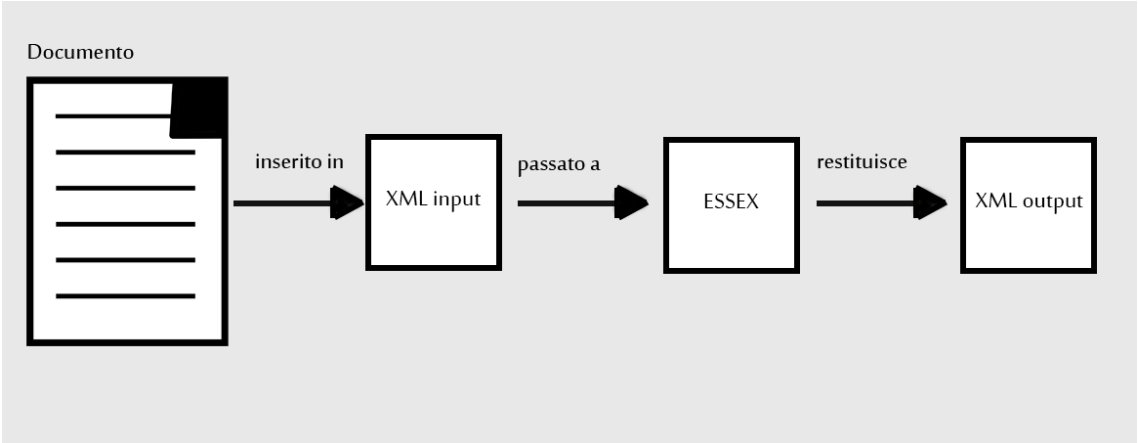

Figura 3.1 – Processo dal documento all' XML di output

Il GSL Dispatcher agisce come bilanciatore di carico per il GSL Server. È un ulteriore livello che si interpone tra l'esterno e il GSL Server. Raccoglie in una coda le richieste e le inoltra ai GSL Server agendo come una sorta di *proxy*; le richieste quindi non saranno recapitate direttamente al GSL Server, ma bisognerà comunicare prima con il GSL Dispatcher.

Nella figura successiva è descritta la comunicazione tra Utente, GSL Dispatcher e GSL Server.
Gli agenti interessati sono: un utente, il dispatcher e due GSL Server A e B.

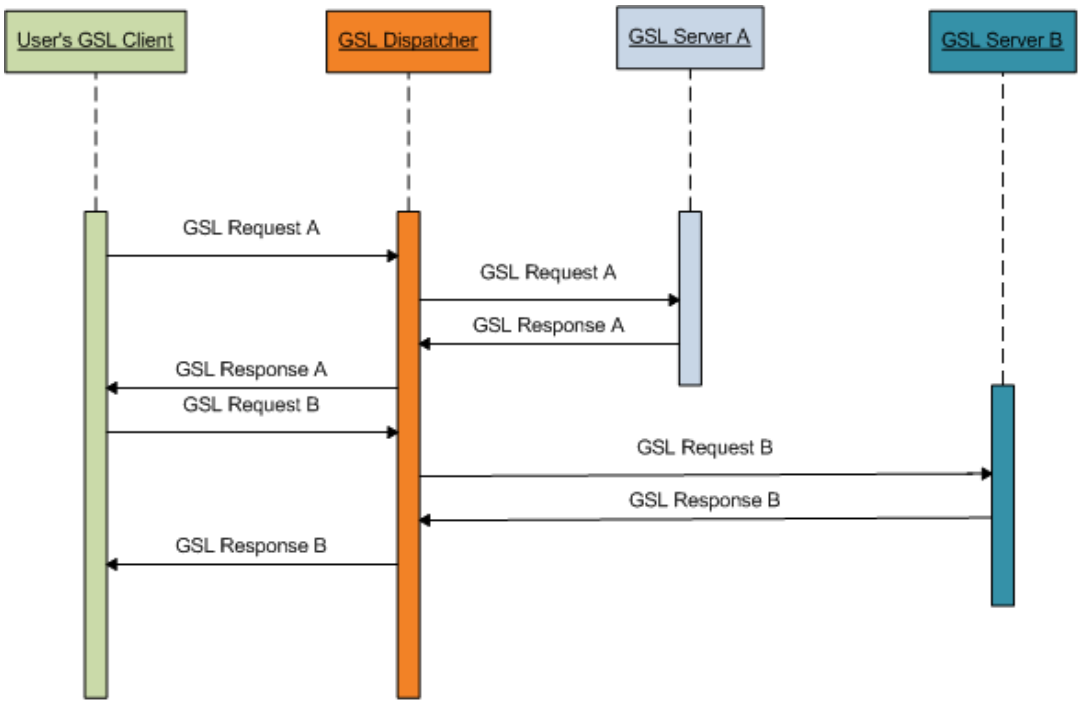

Figura 3.2 – Gestione di una richiesta da parte del GSL Dispatcher

Si nota come in presenza di due richieste A e B il GSL Dispatcher le inoltra a due GSL Server diversi che riportano in modo indipendente il responso al GSL Dispatcher che lo comunica poi all' utente.

3.3 Richiesta di analisi

Quando si utilizza il comando CRL_ANALYZE si hanno in realtà diverse possibili modalità di analisi che andranno specificate nel file XML.

Il file XML in particolare è suddiviso in 4 aree:

- **Area di testo**: In questo campo viene inserito il testo vero e proprio che si desidera analizzare.
- **Area delle richieste**: In questo campo vengono specificate le tipologie di analisi a cui si vuole sottoporre il documento. I dettagli di ogni tipologia di analisi verranno descritti in seguito.
- **Area dei flag**: Contiene opzioni di vario tipo.

- **Area dei metadati**: Informazioni che sono trasportate senza essere modificate dal flusso in entrata al flusso in uscita. Questo campo può non essere presente.

Le tipologie di analisi sono molteplici, ma nel seguito della tesi saranno trattate specialmente la CATEGORIZATION e la EXTRACTION.
In particolare CATEGORIZATION fornisce in output le categorie associate al documento utilizzando le regole in linguaggio "C", mentre EXTRACTION fornisce in output le estrazioni effettuate sul documento seguendo le regole in linguaggio "E".

È possibile fornire ulteriori parametri quando si fa uso del comando EXTRACTION; questi parametri aggiuntivi permettono di richiedere un elaborazione più dettagliata.
I parametri aggiuntivi sono:

- **Verbose**: Richiede al motore semantico di restituire in output informazioni aggiuntive rispetto all' output standard.
- **Level**: Tramite una scala di livelli numerati da 0 a 3 richiede in output un determinato livello di dettaglio. Lo 0 è considerato il minimo dettaglio (restituisce solo i record e i relativi campi), mentre il 3 è considerato il livello a dettaglio massimo (restituisce i singoli atomi che compongono le parole).

Altre operazioni utili possono essere:

- **SYNTHESIS**: estrae informazioni in sintesi sul documento, ad esempio nomi di persona, luoghi, organizzazioni, eccetera…
- **CLEANDOCUMENT**: Fornisce in uscita il documento ripulito dai caratteri superflui come ad esempio gli spazi ripetuti.

Tuttavia queste operazioni non sono state utilizzate nel progetto, ma ci si è limitati all'utilizzo di CATEGORIZATION ed EXTRACTION.

Capitolo 4
Analisi del progetto

4.1 Obbiettivo del progetto

Si vuole realizzare un'applicazione che, utilizzando gli strumenti descritti nei capitoli precedenti, fornisca risposte a domande poste in linguaggio naturale riguardanti i trasporti ferroviari.
Gli utenti potenzialmente potrebbero richiedere domande di tipologie molto diverse, si è cercato quindi di porre attenzione solamente su alcune di esse, ovvero quelle che potrebbero essere le più utili e frequenti per un utente dei servizi ferroviari.
I quesiti che l'applicazione dovrà poter risolvere sono:

1. Rispondere se il treno è in orario o meno ed in caso di ritardo indicare i minuti di ritardo.
2. Ricercare un treno da una certa stazione di partenza ad una certa stazione di arrivo per un certo giorno.
3. Ricercare una certa tipologia di treno (es. Regionale, Intercity, eccetera…)

4. Ricercare un treno in base ai servizi che può fornire ai passeggeri (servizio biciclette, disabili, bar, ristorante, cuccette)
5. Ordinare i treni in base al prezzo delle corse e quindi ricavare la corsa più economica.
6. Ricercare solamente treni diretti, senza cambi.
7. Integrare i servizi di Trenitalia con quelli di Italo Treno per la ricerca di un treno ad alta velocità su una certa tratta.

Si noti inoltre che i punti 2,3,4,5,6 possono essere combinati tra loro, ovvero con una singola richiesta posso restringere la ricerca di un treno utilizzando tutte le restrizioni descritte nei suddetti punti.
L'applicazione, oltre al vantaggio di rispondere a domande in linguaggio naturale e quindi in modo più intuitivo per gli utenti, ha anche il vantaggio di permettere delle restrizioni sulle ricerche che non sono presenti nei servizi di ricerca treni di Trenitalia: ad esempio la ricerca di un treno in base ai servizi che può offrire non è fornita da Trenitalia, così come non è fornita la ricerca per una specifica tipologia di treno.

4.2 Idea di risoluzione

Per risolvere il problema e raggiungere quindi l'obiettivo si avrà bisogno di suddividere il lavoro in diverse parti:

1. Creazione di un ambiente in Cogito in grado di fornire le informazioni necessarie sulla tipologia di domanda fornita dall' utente.
2. Scrivere un codice in un linguaggio di programmazione che sfrutti le conoscenze ottenute dal motore semantico per interrogare in modo adeguato dei siti web che contengono informazioni riguardanti i servizi ferroviari (Trenitala ed Italo). Questa fase comprenderà in particolar modo la stesura di alcuni script che estrarranno le informazioni necessarie da alcuni siti web.
3. Elaborare le informazioni ottenute effettuando delle restrizioni sui risultati.

4. Utilizzando dei template realizzare un’interfaccia grafica che consenta all’ utente di porre le domande in linguaggio naturale e visualizzare l’output in modo semplice.

Nei prossimi capitoli si discuteranno in modo approfondito tutte le fase del lavoro.

Capitolo 5
Creazione ambiente in COGITO

Il primo passaggio per la realizzazione del progetto è riconoscere, tramite gli strumenti forniti da Cogito, le tipologie di domande effettuate dall' utente ed estrarre le informazioni contenute nelle domande.
Ad esempio devo saper discernere se una domanda richiede il ritardo di un treno oppure richiede di sapere le corse per un Frecciarossa da Milano a Roma; inoltre devo anche estrarre le informazioni utili, ad esempio le città di partenza e arrivo o la data o altre informazioni ancora.
Per eseguire queste operazioni è evidente la necessità di utilizzare gli strumenti di Categorizzazione e di Estrazione descritti nei capitoli precedenti.

5.1 Fase di Categorizzazione

Tenendo conto delle tipologie di domande che si vogliono prendere in considerazione si crea l'albero dei domini in modo tale che ogni tipologia di domanda sia "coperta" da un dominio e quindi da una categoria.

Nel progetto l'albero dei domini è il seguente:

Figura 5. 1 – Albero dei domini

Ogni categoria descrive una tipologia di domanda e possono essere attivate più categorie contemporaneamente a seconda se la domanda pone uno o più vincoli. In tabella sono descritte per ogni dominio la tipologia di domanda a cui è assegnato.

Dominio	**Descrizione**
CercaT	Si attiva ogni volta che ricerco un treno.
Diretto	Si attiva quando nella domanda è specificato che si cerca un treno diretto.
Migliorprezzo	Si attiva quando nella domanda si fa riferimento al miglior prezzo o al prezzo più basso.
Ritardo	Si attiva quando l'utente desidera sapere lo stato di ritardo di un treno specifico.
Servizio	Si attiva quando nella domanda è richiesto una tipologia di servizio precisa (es. Disabili, Ristorante, eccetera…)
TAV	Si attiva quando si effettua la ricerca di un treno ad alta velocità (Quindi senza badare se la compagnia è Trenitalia o Italo)
TipoT	Si attiva quando nella domanda è specificato un tipo di treno particolare. (es. Frecciabianca, Regionale, eccetera…)

Tabella 5. 1 – Elenco dei domini presenti nel progetto

Ora che si sono messe le basi per la categorizzazione bisogna procedere con il fornire a Cogito le regole mediante le quali le domande possano essere categorizzate. Per fare questo devo utilizzare le regole scritte in linguaggio “C” e dei file di estensione “.efo”. Nel progetto per chiarezza è stata suddivisa ogni categoria in un file “.efo” che contiene le regole specifiche per quella singola categoria, ma sarebbe anche possibile aggregare le regole in un solo file.

I file “.efo” contenenti le regole di categorizzazione sono i seguenti:

- **cercatreno.efo**

```
SCOPE SENTENCE
{
  DOMAIN(CercaT:NORMAL)
  {
    KEYWORD("da")
    <>
    TYPE(NPR)
    <>
    KEYWORD("a") OR KEYWORD("per") OR KEYWORD("ad")
    <>
    TYPE(NPR)
  }

  DOMAIN(CercaT:NORMAL)
  {
    KEYWORD("per")
    <>
    TYPE(NPR)
    <>
    KEYWORD("da")
    <>
    TYPE(NPR)
  }

  DOMAIN(CercaT:NORMAL)
  {
    KEYWORD("tra") OR KEYWORD("TRA")
    <>
    TYPE(NPR)
    <>
    KEYWORD("e") OR KEYWORD("E")
    <>
    TYPE(NPR)
  }
}
```

Il file cercatreno.efo inserisce la generica ricerca del treno nella categoria CercaT ed avendo impostato come *scope* SENTENCE verrà valutata ogni frase (che sarà solo una nel caso in cui l'utente ponga una sola domanda).

Le regole create sono tre per la categorizzazione nel dominio CercaT: può essere valida una domanda del tipo "Treno da Milano a Roma", oppure "Treno per Roma da Milano", ma anche "Treno tra Milano e Roma"

- **diretto.efo**

```
SCOPE SENTENCE
{
  DOMAIN(Diretto:NORMAL)
  {
    KEYWORD("diretto")
    OR KEYWORD("senza cambi")
  }
}
```

In questo caso invece l'unica regola per cui una domanda debba attivare il dominio "Diretto" è che nella frase si faccia riferimento a "diretto" o "senza cambi".

- **migliorprezzo.efo**

```
SCOPE SENTENCE
{
  DOMAIN(Migliorprezzo:NORMAL)
  {
    KEYWORD("prezzo")
    OR KEYWORD("costo")
    <>
    KEYWORD("basso")
    OR KEYWORD("economico")
  }

  DOMAIN(Migliorprezzo:NORMAL)
  {
    ANCESTOR(317564)
    OR
    ANCESTOR(317563)
    <>
    KEYWORD("costo")
```

```
    OR KEYWORD("prezzo")
  }
}
```

Nella prima regola trovo le sequenze del tipo “prezzo più economico” o “costo più basso” e così via, mentre tramite la seconda regola utilizzo il concetto di “migliore, meglio,…” dato dai campi del Sensigrafo identificati con i numeri 317564 e 317563; in questo modo posso trovare segmenti di frase del tipo “miglior prezzo” e catalogare la domanda attivando il dominio “Migliorprezzo”.

- **ritardo.efo**

```
SCOPE SENTENCE
{
  DOMAIN(Ritardo:NORMAL)
  {
    ANCESTOR(11631)
    OR ANCESTOR(115710)
  }
}
```

Tramite questa regola la domanda sarà associata alla categoria “Ritardo” qualora al suo interno siano presenti i concetti di “ritardo, rinvio” o di “essere in ritardo, arrivare più tardi”, concetti che sono identificati nel Sensigrafo con gli identificativi 11631 e 115710.

- **servizio.efo**

```
SCOPE SENTENCE
{
  DOMAIN(Servizio:NORMAL)
  {
    KEYWORD("servizio")
    OR KEYWORD("supporta")
    OR KEYWORD(LOAD "servizi.txt")
  }
}
```

Tramite questa regola le domande che saranno categorizzate attivando il dominio “Servizio” saranno quelle che contengono la parola “servizio” o “supporta” o che

contengono uno dei servizi contenuti nel file “servizi.txt”. Tramite l’attributo LOAD è possibile caricare le parole contenute nel suddetto file ed utilizzarle come *keyword* separate.

- **tav.efo**

```
SCOPE SENTENCE
{
   DOMAIN(TAV:NORMAL)
   {
      ANCESTOR(282332)
      AND NOT
      ANCESTOR(518901)
   }
}
```

In questo caso ho categorizzato come “TAV” le domande che contengono il concetto generico di “Alta velocità” dato dall’elemento del Sensigrafo identificato con il numero 282332, ma che non contengono il concetto di “FrecciaRossa” dato dall’ elemento del Sensigrafo identificato dal numero 518901. Questo perché se dovessi ricercare un FrecciaRossa la ricerca dovrebbe limitarsi alla categoria di treno invece che ricercare un generico treno ad alta velocità.

- **tipot.efo**

```
SCOPE SENTENCE
{
   DOMAIN(TipoT:NORMAL)
   {
      ANCESTOR(518901) //Frecciarossa
      OR KEYWORD("freccia rossa")
      OR KEYWORD(“frecciarossa”)
   }

   DOMAIN(TipoT:NORMAL)
   {
      KEYWORD ("freccia bianca")
      OR KEYWORD("frecciabianca")
   }

   DOMAIN(TipoT:NORMAL)
   {
      ANCESTOR(518902) //FrecciaArgento
      OR KEYWORD("freccia argento")
```

```
    OR KEYWORD("frecciaargento")
  }

  DOMAIN(TipoT:NORMAL)
  {
    ANCESTOR(323800)//Regionale
    OR KEYWORD("regionale")
  }

  DOMAIN(TipoT:NORMAL){
    ANCESTOR(23684)//Intercity
    OR KEYWORD("Inter City")
    OR KEYWORD("inter city")
  }
}
```

Per verificare che una domanda ricerchi un tipo specifico di treno ho suddiviso le regole per ogni categoria di treno.

Nella prima regola si verifica se la domanda richiede la ricerca di un FrecciaRossa, sia tramite l'attributo KEYWORD, sia tramite il concetto del Sensigrafo.
Tramite KEYWORD verifico la ricerca di un FrecciaBianca, sia scritto con lo spazio sia senza lo spazio, in modo da prevenire le possibili diverse digitazioni degli utenti.
Il tipo Freccia Argento lo verifico con il Sensigrafo e con le KEYWORD come fatto per il FrecciaRossa.
Per il treno Regionale esiste il concetto nel Sensigrafo di Cogito, individuato dal numero 323800, quindi si è utilizzato quello in aggiunta all'attributo KEYWORD.
Infine per la tipologie di treno Intercity si è utilizzato ancora una volta sia il concetto del Sensigrafo sia le KEYWORD.

Se una di queste regole viene rispettata allora sarà attivato il dominio denominato "TipoT", il che significa che la domanda pretende una restrizione sul tipo di treno da ricercare.

- **main.efo**

```
IMPORT "servizio.efo"
IMPORT "ritardo.efo"
IMPORT "migliorprezzo.efo"
IMPORT "diretto.efo"
```

```
IMPORT "tav.efo"
IMPORT "cercatreno.efo"
IMPORT "tipot.efo"
```

Per ultimo si è lasciato il file “main.efo” che è il file da cui comincia la compilazione delle regole. In questo caso si è sfruttato solamente per importare gli altri file visti in precedenza, ma potrebbe essere utilizzato anche per scrivere regole direttamente.

5.2 Fase di Estrazione

Una volta terminata la scrittura delle regole di categorizzazione si è passati alla fase di estrazione. Analizzando il problema che si vuole risolvere (ovvero la risposta ad una domanda sui servizi ferroviari) si sono notate delle informazioni specifiche che posso estrarre dal testo della domanda e che sono necessarie per elaborare la risposta. In particolare queste informazioni sono: la città di partenza, la città di arrivo, la data della corsa, il tipo di treno, il tipo di servizio e il numero del treno. Senza conoscere queste informazioni, infatti, sarebbe impossibile elaborare qualsiasi tipo di risposta che possa soddisfare la domanda.

Come descritto nei capitoli precedenti, per procedere all’ estrazione di informazioni da un testo bisogna porre delle regole scritte in linguaggio E. Queste regole sono contenute in un solo file che, nel progetto, prende il nome di “main.efe”. Il file risulta sostanzialmente diviso in due parti: una prima parte in cui si dichiarano i campi in cui andranno memorizzate le informazioni estratte, ed una seconda parte in cui saranno scritte le regole che permetteranno ad ESSEX di estrarre le informazioni.

La prima parte del file per le estrazioni è così formata:

```
TEMPLATE(TRATTA)
{
  @PARTENZA,
  @ARRIVO
}
```

```
TEMPLATE(TIPOLOGIA)
{
   @TIPO
}

TEMPLATE(GIORNO)
{
   @DATA
}

TEMPLATE(NTRENO)
{
   @NUMERO
}

TEMPLATE(SERVIZIO)
{
   @TIPOS
}
```

Tramite TEMPLATE si definiscono cinque strutture dati, una per ogni tipologia di dato che si intende memorizzare. In particolare la struttura TRATTA sarà composta di due field: PARTENZA e ARRIVO.

La seconda parte del file invece comprende le regole per le estrazioni ed è così formata:

```
SCOPE SENTENCE
{
 IDENTIFY(TRATTA)
   {
     KEYWORD("da")>@PARTENZA[TYPE(NPR)]
     OR
     KEYWORD("tra")>@ARRIVO[TYPE(NPR)]
   }
}

SCOPE SENTENCE
{
   IDENTIFY (SERVIZIO)
   {
     @TIPOS[KEYWORD(LOAD "servizi.txt")]
   }
}
SCOPE SENTENCE
{
   IDENTIFY(TRATTA)
   {
   KEYWORD("a")>@ARRIVO[TYPE(NPR)]
   OR
```

```
   KEYWORD("per")>@ARRIVO[TYPE(NPR)]
   OR
   KEYWORD("ad")>@ARRIVO[TYPE(NPR)]
   OR
   KEYWORD("e")>@ARRIVO[TYPE(NPR)]
   }
}

SCOPE SENTENCE
{
   IDENTIFY(TIPOLOGIA){
      @TIPO[ANCESTOR(518901)]
      OR
      @TIPO[KEYWORD(LOAD "tipo.txt")]
   }
}

SCOPE SENTENCE
{
   IDENTIFY(GIORNO){
      @DATA[TYPE(DAT)]
   }
}

SCOPE SENTENCE
{
   IDENTIFY(NTRENO){
      @NUMERO[PATTERN("\d{3,5}")]
      AND NOT
      TYPE(DAT)
   }
}
```

Si nota che IDENTIFY permette di indicare la struttura dati a cui si vuole accedere, mentre i singoli field della struttura sono acceduti anteponendo al loro nome il carattere "@".

La prima struttura dati considerata è TRATTA. Per quanto riguarda la partenza si inseriscono nel field PARTENZA i nomi propri (di città) che seguono la *keyword* "da". Quindi ogni volta che si presenta una frase che contiene "… da *nomedicittà*" il valore di *nomedicittà* viene inserito nel campo PARTENZA.
Per quanto riguarda il campo ARRIVO posso memorizzare un'informazione al suo interno in più modi. Sia se *nomedicittà* fa seguito alla KEYWORD "a", sia se fa seguito a "da" o a "per". In questo modo saranno inserite in ARRIVO le città sia se scritte come "… da *nomedicittà*", ma anche come "… a *nomedicittà*" o "… per *nomedicittà*".

Con lo stesso principio sono state scritte le altre regole. Interessante l'utilizzo del pattern per ricavare il numero del treno, numero che dev'essere composto da almeno 3 e massimo 5 cifre decimali.

Per quanto riguarda i servizi e le tipologie di treno si è utilizzato, come per la categorizzazione, il comando LOAD che carica un file esterno contenente tutti i possibili servizi e le possibili categorie di treno. Se nella domanda si fa riferimento ad uno dei valori contenuti nei file esterni questo valore è memorizzato in TIPOS se si tratta di un servizio e in TIPO se si tratta di una categoria dei treni.

5.3 Risultati di elaborazioni di esempio su Cogito Studio

Vediamo ora come Cogito risponde ad alcune domande di esempio e come le regole di categorizzazione ed estrazione restituiscano effettivamente i risultati aspettati.

ESEMPIO 1

È in ritardo il treno 3456 del 06/12/2014?

L'unica regola che si attiva è quella relativo al ritardo, e quindi l'unico dominio attivato è "Ritardo".
I valori ricavati dall' estrazione sono invece:

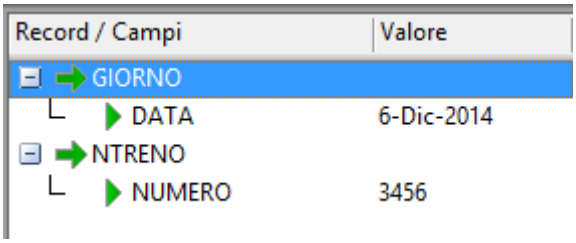

Figura 5.2 – Risultati di un' estrazione

Come ci si aspettava.

ESEMPIO 2

Treno Regionale da Modena a Carpi con servizio biciclette il 22/05/2015

Questa frase è un po' più complessa della precedente e richiede l'attivazione di più domini e l'estrazione di diversi valori.
I domini che vengono attivati sono: "TipoT" (ed in particolare viene attivato grazie al riconoscimento sul treno Regionale), "Servizio" e il generico "CercaT". La domanda può quindi essere categorizzata come: "richiesta di un particolare tipo di treno che soddisfi un particolare tipo di servizio".

Per quanto riguarda invece le estrazioni i valori ricavati sono i seguenti:

Figura 5.3 – Risultati di un' estrazione

Capitolo 6
Implementazione

6.1 Introduzione

Questo capitolo analizza le parti 2 e 3 dell'idea di risoluzione vista nel capitolo 4.

L'applicazione che si vuole sviluppare sarà una Servlet sviluppata in Java che permetta agli utenti di accedere ad un sito web e da lì inserire le domande al sistema di analisi lessicale. Le risposte che il sistema restituirà saranno visualizzate a video in forma tabellare o testuale.

Per realizzare l'applicazione ci si è dovuti servire di alcune componenti, tra cui un *web server*, un *application server* e un gestore dei template che verrà descritto nel prossimo capitolo.

6.2 Apache e Tomcat

6.2.1 PERCHÉ APACHE E TOMCAT?

Per quanto riguarda il *web server* e l'*application server* la scelta è ricaduta rispettivamente su Apache e Tomcat.

Apache è un *web server* freeware che consente di gestire le richieste di pagine web provenienti da un client (in particolare da un web browser) e di rispondere opportunamente fornendo le pagine corrispondenti.

Tomcat invece è un *application server* e fornisce una piattaforma software in grado di consentire l'esecuzione di applicazioni web sviluppate in Java. Può essere visto anche come un contenitore di servlet. Anche Tomcat è freeware e necessita solamente di essere installato e configurato.

Senza utilizzare Apache il progetto che si vuole realizzare non potrebbe esistere sul web, mentre senza utilizzare Tomcat non si potrebbe eseguire l'applicazione servlet che consente di raggiungere l'obiettivo posto all'inizio della discussione.

Di seguito un'immagine che indica come Apache e Tomcat comunicano tra loro dopo aver ricevuto una richiesta dell'utente.

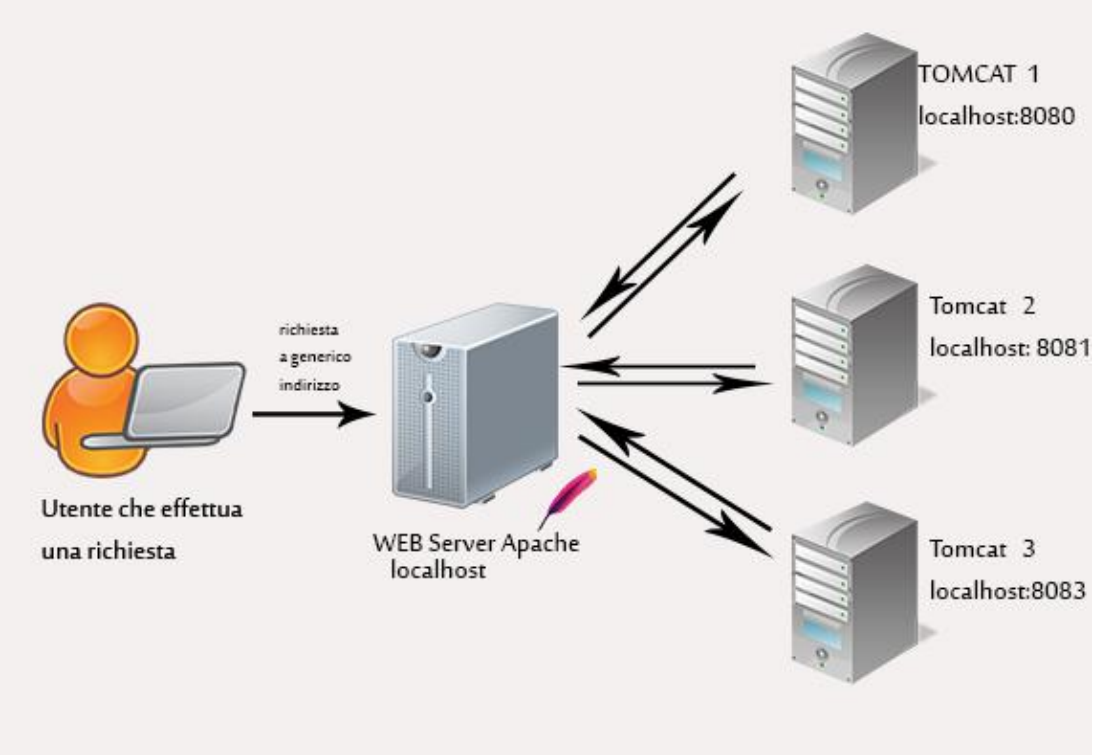

Figura 6. 1 – Schema di gestione di una richiesta utente da parte di Apache e Tomcat

In questa immagine si nota come un utente che effettua una richiesta comunica con il Server Apache che la dirotta poi sull' application server corrispondente. È possibile avere più application server Tomcat su uno stesso web server Apache, l'importante è che ognuno abbia impostato una porta diversa dall' altro.

6.2.2 CONFIGURAZIONE DI TOMCAT

La configurazione di Tomcat avviene editando i file XML contenuti in alcune specifiche cartelle.
La porta di riferimento (nel progetto è stata utilizzata la 8080) si può impostare nel file di configurazione "server.xml" contenuto nella cartella "conf" di Tomcat.

Le configurazioni specifiche dell'applicazione invece si gestiscono modificando i file XML contenuti nella cartella dell'applicazione che sarà a sua volta contenuta nella cartella "webapps". Ogni sotto directory di "webapps" rappresenta una applicazione caricata su Tomcat; nel nostro caso l'applicazione creata per il progetto si chiamerà

“orariotreni” e quindi la sua cartella avrà lo stesso nome.
Nella directory “orariotreni” è presente il file “web.xml” che contiene le informazioni relative alle azioni che Tomcat deve eseguire ogni volta che riceve un determinato tipo di richiesta. Ad ogni possibile richiesta che si intende gestire si associa una classe Servlet che conterrà un metodo “execute” la cui esecuzione consentirà di ottenere la risposta per la richiesta.
Sempre nel file “web.xml” si indicano anche i template in Velocity da utilizzare nel progetto.

Una copia del file “web.xml” sarà inserita nell’ appendice in modo da chiarire anche come è fisicamente strutturato il suddetto file.

I template in Velocity sono stati inseriti in una cartella apposita, così come le classi di estensione “.class”. Da notare che la strutturazione delle directory delle classi deve rispettare quella del progetto, quindi la cartella in Tomcat deve avere la stessa strutturazione della cartella del workspace contenente i file “.class”.

6.3 Classi Java

Per descrivere almeno in parte il codice sviluppato per generare l’applicazione si prenderanno in considerazione, una ad una, le classi che formano il progetto.

6.3.1 CLASSE “BaseServlet”

“BaseServlet” è una classe che estende la classe di Java “HttpServlet”.
Il suo scopo è quello di gestire le richieste in get o in post che arrivano, da parte dell’utente, al *web server* Apache. Questa classe, estendendo “HttpServlet”, ha la caratteristica di sovrascrivere i metodi “doGet” e “doPost” e fa in modo che la richiesta venga eseguita richiamando il metodo “execute” di cui si tratterà più avanti.

Le richieste in get attiveranno il metodo “doGet” che richiamerà a sua volta il metodo “doPost”. Si può quindi dire che le richieste, sia le get, sia le post, vengono tutte gestite come delle post.

Inoltre la classe “BaseServlet” implementa anche il metodo “done” che viene richiamato dopo che una richiesta è stata esaudita.

6.3.2 CLASSE “LeggiInput”

Questa è una delle classi più importanti del progetto, in quanto si implementa il metodo “execute” che è il metodo che di fatto gestisce le richieste e fornisce la risposta all’utente. Il metodo “execute” sovrascrive quello contenuto in “BaseServlet” perché la classe “LeggiInput” estende proprio la classe “BaseServlet”.

Per ricavare la risposta alla domanda posta dall’ utente in linguaggio naturale, per prima cosa devo formare l’XML di input da passare poi a Cogito. L’XML deve contenere la domanda dell’ utente e le tipologie di analisi da effettuare sul testo. La domanda verrà posta dall’utente in un form che sarà poi soggetto ad una submit che genererà la richiesta. Il testo nel form sarà il testo che Cogito andrà ad analizzare.

Il contenuto dell’ XML di input viene creato in questo modo:

```
String contenutoxml = new String("<?xml version='1.0'
encoding='utf-8'?><REQUEST VERSION='13.6.1'><DOCUMENT
TYPE='SECTIONS'><SECTION
NAME='BODY'><![CDATA[##DOMANDA##]]></SECTION></DOCUMENT><ANALYS
IS NAME='CATEGORIZATION'><PARAM NAME='TYPE'
VALUE='ALL'/></ANALYSIS><ANALYSIS
NAME='EXTRACTION'/></REQUEST>");
```

Si può subito notare che le tipologie di analisi alle quali sarà sottoposto il testo sono di tipo “CATEGORIZATION” ed “EXTRACTION” inserite nel tag “ANALYSIS NAME”.

Inoltre occorre notare che è presente il pattern "##DOMANDA##"; questo pattern verrà sostituito in seconda battuta dalla vera e propria domanda che sarà ricavata dal form compilato dall' utente.

Per analizzare il testo devo però collegarmi con Cogito, in particolare devo collegarmi con il GSL Dispatcher che a sua volta comunicherà con il GSL Server che eseguirà le analisi. Il collegamento si effettua tramite un metodo di ExpertSystem che viene eseguito come thread e a cui viene passato in input il file XML, l'indirizzo a cui si trova il GSL Dispatcher (nel progetto essendo in locale sarà ovviamente "localhost", ma può anche risiedere su un server esterno, in quel caso si utilizzerà l'indirizzo corrispondente), la porta con cui ci si collega al GSL Dispatcher (nel progetto è quella di default, ovvero la porta 5000) e l'id del GSL Dispatcher. È necessario che quando si compie la connessione al GSL Dispatcher esso sia effettivamente attivo e connesso sulla porta corretta.

Una volta che Cogito ha analizzato il testo restituisce un risultato, sempre in XML, da cui verranno estratte le informazioni utili tramite i metodi "populateResponse" e "populateExtraction" che appunto estraggono le informazioni e le categorie del testo. I suddetti metodi verranno analizzati in seguito in modo più dettagliato.

Se una categoria è attivata si imposta il flag relativo a quella categoria al valore 1; questo servirà poi per eseguire i controlli sulle categorie in maniera più veloce.

Procedo quindi con i controlli sulle categorie per dare per ogni tipologia di domanda la risposta che possa soddisfarla. Essendo che, come indicato nell' analisi del problema, le domande poste dall' utente possono richiedere anche più restrizioni, le tipologie di domande finali non si limiteranno alle singole categorie, ma saranno il risultato delle singole categorie e la composizione reciproca di alcune categorie. Riporto di seguito l'elenco delle possibili domande e le relative composizioni.

- Ritardo
- Ricerca treno generico (diretto o non diretto)
    - Ricerca treno generico con servizio specifico

    - Ricerca treno generico con servizio specifico e con una certa categoria di treno
    - Ricerca treno generico con servizio specifico e al miglior prezzo
    - Ricerca treno generico con servizio specifico, con una certa categoria di treno e al miglior prezzo
    - Ricerca treno generico con una certa categoria di treno
    - Ricerca treno generico al miglior prezzo
    - Ricerca treno generico con una certa categoria di treno e al miglior prezzo
- Ricerca Treno Alta Velocità

Ogni categoria di domanda richiama il metodo della classe "CallFormNew" (che sarà trattata in seguito) designato ad operare un' interrogazione ad un form di un sito web che fornisce le necessarie informazioni riguardanti le corse dei treni. In particolare, per la domanda sul ritardo di un treno si interrogherà il sito http://www.uniquevisitor.it/magazine/trenitalia-orari.php che ha un form specifico per la ricerca del ritardo di un treno. Lo stesso sito sarà utilizzato anche per la ricerca di un generico treno utilizzando però un altro form, mentre per la ricerca del treno ad alta velocità si utilizzano siti web differenti, ma anche questo sarà trattato in modo più specifico quando si descriverà la classe "CallFormNew". Questi metodi restituiranno, nel caso della ricerca di un treno generico o di un treno ad alta velocità, una lista contenente le corse che soddisfano i parametri di Partenza, Arrivo e Data forniti dall' utente; dopodiché a questa lista saranno applicate le restrizioni necessarie che rimuoveranno delle corse fino ad ottenere la lista contenente le sole corse che soddisfano appieno la domanda posta dall' utente all'inizio.

Una volta terminati questi procedimenti posso fornire la lista ed altri parametri (a seconda del tipo di domanda) al template in modo che possano essere impaginati a video e visualizzati dall' utente.

Infine si reimpostano i flag a zero e si settano i parametri in modo da preparare il sistema alla prossima interrogazione. Il procedimento si conclude richiamando il

metodo “done” di “BaseServlet” che termina l’operazione e rende il sistema pronto per la prossima richiesta.

Rimangono da descrivere i metodi “populateResponse” e “populateExtraction” che sono stati sopra menzionati.
Il metodo “populateResponse” restituisce un oggetto della classe “Response” che ha il compito di contenere tutte le informazione di una richiesta, nel caso di “populateResponse” si agisce sulle categorie di una richiesta. Per rendere più chiaro il procedimento allego il codice del metodo “populateResponse”.

```
private Response populateResponse(Document doc, Response res) {
   final List<String> categorie = new ArrayList<String>();
   final NodeList nList = doc.getElementsByTagName("CATEGORIZATION");
   for (int temp = 0; temp < nList.getLength(); temp++) {
      final Node nNode = nList.item(temp);
      if (nNode.getNodeType() == Node.ELEMENT_NODE) {
         final Element eElement = (Element) nNode;
         for (temp = 0; temp < eElement.getChildNodes().getLength();
temp++) {
           final Node nNodeC = eElement.getChildNodes().item(temp);
           if (nNodeC.getNodeType() == Node.ELEMENT_NODE)
{
              final Element eElementC = (Element) nNodeC;
              categorie.add(eElementC.getAttribute("NAME"));
           }
         }
         res.setCategoire(categorie);
      }
   }
   return res;
}
```

Si noti innanzitutto che l’oggetto “doc” passato come parametro è l’XML fornito in output da Cogito.
Viene creata una lista di categorie in cui verranno inserite le categorie rilevate. Poi si selezionano gli elementi del documento che hanno il tag CATEGORIZATION e per ognuno di questi si estraggono i valori dei suoi figli che hanno tag uguale a NAME. Questo valore sarà il nome della categoria e sarà inserito nella lista “categorie”. Infine la lista sarà inserita tramite il metodo “set” nell’ oggetto “res” di tipo “Response” e verrà ritornato al metodo chiamante.

L'estrazione delle informazioni invece è concettualmente uguale all' estrazione delle categorie, bisogna solamente modificare i parametri dei TAG dell'XML e considerare che i TAG di EXTRACTION sono strutturati in modo diverso rispetto ai tag di CATEGORIZATION. Il codice del metodo "populateExtraction" sarà comunque presente nel codice inserito nell' appendice al termine dei capitoli della tesi.

6.3.3 CLASSE "CallFormNew"

La classe "CallFormNew" è importante almeno quanto la "LeggiInput" in quanto è la classe che implementa i metodi per interrogare i siti web che contengono le informazioni sulle corse dei treni; informazioni che sono indispensabili per la riuscita dell'applicazione. Le interrogazioni ai siti avvengono mediante delle richieste in post o in get ai siti stessi. I risultati saranno poi elaborati ed inseriti in oggetti specifici per poter poi essere utilizzati nelle successive elaborazioni.

Il primo metodo della classe "CallFormNew" è il metodo "sendPost" che effettua una richiesta in post al sito http://m.viaggiatreno.it/vt_pax_internet/mobile/numero per verificare lo stato di ritardo di un certo treno. Il metodo "sendPost" è richiamato nella classe "LeggiInput" nel blocco di codice che gestisce appunto la richiesta relativa al ritardo di un treno.
A "sendPost" viene passato come parametro l'indirizzo della pagina su cui effettuare la request in post e il numero del treno che si vuole controllare. Si impostano i parametri da utilizzare nella post in questo modo:

```
List<NameValuePair> paramList = new ArrayList<NameValuePair>();
     paramList.add(new BasicNameValuePair("numeroTreno", num));
     paramList.add(new BasicNameValuePair("tipoRicerca", "numero"));
     paramList.add(new BasicNameValuePair("lang", "IT"));
```

Dopodiché si crea un oggetto "post" di tipo "HttpPost" a cui setto la lista dei parametri paramList tramite l'istruzione:

```
post.setEntity(new UrlEncodedFormEntity(paramList));
```

E si esegue la richiesta in post tramite l'istruzione:

```
HttpResponse response = client.execute(post);
```

Se la richiesta ha avuto esito positivo (quindi il codice di risposta è 200) estraggo il codice html della risposta in una variabile di tipo String. Dal codice di uscita estraggo poi le informazioni necessarie per dare una risposta alla richiesta dell'utente, ovvero lo stato del treno. Se il treno è in ritardo ricavo i minuti di ritardo e li restituisco, mentre se il treno è in orario o non è ancora partito restituisco il relativo messaggio.

Il metodo designato invece alla richiesta di informazioni su una corsa di un treno è il "getPageContent". Questo metodo effettua una richiesta in get compilando il form per la ricerca di un treno contenuto all' indirizzo http://www.uniquevisitor.it/magazine/trenitalia-orari.php.
Il metodo accetta in input la città di partenza e quella di arrivo, la data e l'indirizzo su cui effettuare la get (se la data non è presente viene utilizzata quella odierna). Anche in questo metodo si inseriscono i parametri che compongono la request in get e si esegue la richiesta. La risposta che si otterrà sarà una pagina HTML che verrà poi inserita in un oggetto di tipo stringa.
Il procedimento con cui verranno estratte le informazioni dal documento HTML di risposta sarà adattato al tipo di risposta dello specifico sito a cui effettuo la richiesta. Se si decide in futuro di utilizzare siti differenti bisognerà modificare il procedimento di estrazione dei dati in modo che si adatti al documento HTML di risposta dei siti che saranno utilizzati.

Nel progetto il codice utilizzato per l'estrazione delle informazioni sulle corse dei treni è il seguente:

```
for (Element table : doc.select("table")) {
  for (Element row : table.select("tr")) {
    Elements tds = row.select("td");
    if (table.attr("class").equals("detagglio") == false) {
      if (tds.size() == 8) {
        ResocontoOrari resOr = new ResocontoOrari(
  resOr.setStazionepartenza(tds.get(0).text().substring(6,
tds.get(0).text().length()));
```

```
   resOr.setOrapartenza(tds.get(0).text().substring(0, 5));
   resOr.setStazionearrivo(tds.get(1).text().substring(6,
tds.get(1).text().length()));
   resOr.setOrarrivo(tds.get(1).text().substring(0, 5));
   resOr.setDurata(tds.get(2).text());
   resOr.setNumerotreno(tds.get(3).text());
   resOr.setCategoria(tds.get(4).select("img").attr("alt"));
   if (resOr.getCategoria().contains("Regionale")) {
     resOr.setCategoria("Regionale");
     }
   if (tds.get(5).text().contains("ND") == false) {
   resOr.setPrezzoprima(tds.get(5).text().replace(",",
".").substring(0, tds.get(5).text().length() - 2));
   resOr.setPrezzoseconda(tds.get(6).text().replace(",",
".").substring(0, tds.get(6).text().length() - 2));
     } else {
   resOr.setPrezzoprima(tds.get(5).text());
   resOr.setPrezzoseconda(tds.get(6).text());
     }
     resocontro.add(resOr);
   }
   if (tds.size() == 9) {
       ResocontoOrari resOr = new ResocontoOrari();
           resOr.setStazionepartenza(tds.get(0).text().substring(6,
tds.get(0).text().length()));
           resOr.setOrapartenza(tds.get(0).text().substring(0, 5));
           resOr.setStazionearrivo(tds.get(1).text().substring(6,
tds.get(1).text().length()));
           resOr.setOrarrivo(tds.get(1).text().substring(0, 5));
       resOr.setDurata(tds.get(2).text());
       resOr.setCambio("");
       resOr.setNumerotreno(tds.get(4).text());
           resOr.setCategoria(tds.get(5).select("img").attr("alt"));
// Non voglio distinzioni tra Regionale e Regionale Veloce
   if (resOr.getCategoria().contains("Regionale")) {
     resOr.setCategoria("Regionale");
   }
   if (tds.get(7).text().contains("ND") == false) {
     resOr.setPrezzoprima(tds.get(6).text().replace(",",
".").substring(0, tds.get(6).text().length() - 2));
     resOr.setPrezzoseconda(tds.get(7).text().replace(",",
".").substring(0, tds.get(7).text().length() - 2));
   } else {
     resOr.setPrezzoprima(tds.get(6).text());
     resOr.setPrezzoseconda(tds.get(7).text());
   }
   resocontro.add(resOr);
   }
   if (tds.size() == 3) {
     resocontro.get(resocontro.size() -
1).setCambio(tds.get(0).text());
     resocontro.get(resocontro.size() -
1).setNumerotreno(resocontro.get(resocontro.size() -
1).getNumerotreno() + "    " + tds.get(1).text());
     resocontro.get(resocontro.size() -
1).setCategoria(resocontro.get(resocontro.size() - 1).getCategoria() +
"    " + tds.get(2).select("img").attr("alt"));
   }
   }
```

```
    }
}
```

Come si nota dal codice per ogni riga della tabella memorizzo le colonne in un oggetto chiamato “tds” da cui estraggo i valori dei campi e li inserisco nei campi corrispondenti di “resOr” che è un oggetto di tipo “ResocontoOrari”. Nell’ oggetto “resOr” memorizzo le informazioni relative alla corsa di un treno (ad esempio la stazione di partenza, l’arrivo, l’ora di partenza, la tipologia di treno, eccetera…). Per ogni corsa inserisco “resOr” in una lista di corse chiamata “resocontro” che sarà poi il valore di ritorno del metodo.

È da notare nel codice che sono presenti delle condizioni sulla dimensione di “tds”, infatti se “tds” ha dimensione 8 significa che la tabella avrà 8 colonne e vorrà dire che il treno non ha cambi, mentre se “tds” ha dimensione 9 vuol dire che la tabella avrà 9 colonne e quindi sarà presente una colonna che descrive il cambio del treno. Se “tds” ha dimensione 3 invece significa che sto leggendo una colonna che descrive il treno che effettua il cambio.

Di seguito un esempio:

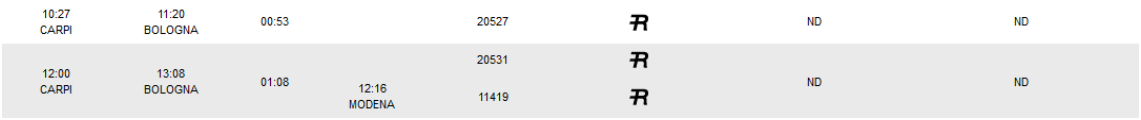

Figura 6. 2 – Esempio di visualizzazione di una corsa sul sito di Trenitalia

Nella prima riga della tabella il treno da Carpi a Bologna è diretto, quindi “tds” avrà dimensione 8. Nella seconda riga invece il treno per la tratta da Carpi a Bologna deve effettuare un cambio a Modena, quindi “tds” avrà dimensione 9, mentre per l’ultima riga (che contiene appunto l’ora e la stazione dove verrà effettuato il cambio, il numero del treno da cambiare e il tipo di treno) “tds” avrà dimensione 3.

Quando agisco sulla riga di dimensione 3 i valori che estraggo non vanno scritti in un nuovo campo della lista, ma semplicemente vengono aggiunti ai valori già presenti nell’ ultimo campo inserito nella lista.

Gli ultimi due metodi di “CallFormNew” sono “sendPostMigliorTreno” e “GetPageContentTav”, che in realtà svolgono lo stesso compito, ovvero ottenere

informazioni sui treni ad alta velocità (compreso Italo), ma "GetPageContentTav" si utilizza solo nel caso in cui, per qualche ragione, "sendPostMigliorTreno" da esito negativo. Infatti "sendPostMigliorTreno" è richiamato dallo stesso metodo "GetPageContentTav" qualora la risposta alla get dia un esito inaspettato.

Anche in questi metodi il procedimento per l'estrazione dei dati è specifica per la tipologia di risposta che forniscono i siti su cui effettuo la request. Il procedimento comunque concettualmente è identico rispetto a quello del metodo "getPageContent" quindi non verranno analizzati nel dettaglio.
È bene però specificare che il metodo "sendPostMigliorTreno" fa riferimento al sito http://www.migliortreno.it/ per effettuare la request in get, mentre "GetPageContentTav" fa riferimento a http://www.easytreno.it/ . Se il servizio di "easytreno" è disponibile si preferisce rispetto a "migliortreno" perché fornisce già i treni ordinati in base al prezzo, che può essere utile per una migliore impaginazione dei risultati.

6.3.4 CLASSE "SingleThreadEx"

È la classe che permette il collegamento con ESSEX utilizzando dei metodi forniti dalle librerie di ExpertSystem. Viene richiamata nella classe "LeggiInput".

6.3.5 CLASSE "Scheda"

È la classe designata a rispondere per la richiesta della homepage dell'applicazione. Estende "BaseServlet" ed ha al suo interno il metodo "execute" che è quello eseguito da Tomcat quando viene ricevuta una richiesta per la homepage, ovvero quando viene inserito nel browser un indirizzo del tipo http://localhost:8080/orariotreni/Scheda.
È già stato descritto perché nel progetto si è usato localhost e la porta 8080, mentre "orariotreni" rappresenta il nome dell'applicazione su Tomcat. Se l'indirizzo termina con Scheda verrà eseguito il metodo "execute" della classe "Scheda".

Il metodo “execute” non fa altro che caricare il template “scheda.vm” che è quello dove compare il form per l’inserimento della domanda da parte dell’utente nonché il tasto di “Submit” che fa partire il processo di analisi.

6.3.6 CLASSI DEL PACKAGE “BEANS”

Le classi del Package “Beans” sono in realtà classi di supporto, ma la loro descrizione è comunque utile per avere una più facile idea di come è stato organizzato il lavoro.

La classe “**ResocontoOrari**” contiene i campi che descrivono la corsa di un treno e i relativi metodi “get” e “set”. I campi della classe “ResocontoOrari” sono:

- stazionepartenza: Stazione di Partenza
- stazionearrivo: Stazione di Arrivo
- orapartenza: Orario di partenza
- orarrivo: Orario d’arrivo
- durata: Durata del viaggio
- cambio: Stazione in cui dev’essere effettuato il cambio del treno
- numerotreno: Numero del treno
- categoria: Categoria del treno (regionale, intercity, eccetera…)
- prezzoprima: Prezzo di prima classe
- prezzoseconda: Prezzo di seconda classe

compagnia: Nome della compagnia (o Trenitalia o Italo)

La classe “**Response**” serve a memorizzare i dati che COGITO ha estratto dalla domanda dell’ utente, sia per quanto riguarda le categorie, sia per quanto riguarda le estrazioni vere e proprie. I campi di cui si compone sono:

- categorie: Lista di categorie. Ogni valore della lista corrisponderà ad una categoria a cui appartiene la domanda.

Data: Contiene la data richiesta dall’ utente per la richiesta del treno.

- Numero: È il numero del treno del quale si vuole ottenere informazioni sul ritardo.
- Arrivo: Contiene la città di arrivo richiesta dall' utente.
- Partenza: Contiene la città di partenza richiesta dall' utente.
- Tipo: Contiene la tipologia di treno richiesta dall' utente.

  Servizio: Contiene il tipo di servizio richiesto dall' utente.

Oltre ai campi sopra indicati ovviamente conterrà anche i metodi "get" e "set" per agire su di essi. Se un campo non è utile alla richiesta dell'utente assume il valore di stringa vuota o di null nel caso della lista di categorie.

Infine la classe "**Comparators**" fornisce un comparator per l'ordinamento dei prezzi delle corse, qualora l'utente abbia richiesto quella con il prezzo più economico.

Capitolo 7
Template con Velocity

7.1 Perché Velocity? Principali Caratteristiche

Velocity è uno strumento sviluppato dall' Apache Foundation che agisce da motore template. Se si utilizza il modello "Model-View-Controller" si può dire che Velocity agisca a livello View.

Nel progetto i file relativi ai template hanno estensione ".vm" e sono contenuti in una sottodirectory della cartella del progetto.
La loro strutturazione è identica a quella dei file HTML, ma hanno la caratteristica di poter comunicare valori con l'applicazione; in questo modo posso visualizzare a video i risultati delle elaborazioni generati dall' applicazione "orariotreni". I valori da comunicare vengono inseriti come attributo nella request ed avranno lo stesso nome usato nel template. Nel template posso poi accedere a questi valori utilizzando il carattere "$" prima del nome.

L'applicazione apre e visualizza i template quando vengono eseguiti i metodi "done" a cui è passato, oltre alla request, anche il nome del template da visualizzare.
Ad esempio, l'istruzione

```
done(request, response, "trovatreni");
```

richiamerà il metodo "done" che avrà al suo interno le istruzioni

```
RequestDispatcher requestDispatcher =
request.getRequestDispatcher("WEB-INF/template/" + templateName
               + ".vm");
requestDispatcher.forward(request, response);
```

La prima istruzione indica il percorso vero e proprio del template il cui nome era stato passato al metodo done e in questo caso era "trovatreni". La seconda istruzione invece esegue la request e quindi visualizza anche il template indicato in precedenza.

7.2 Configurazione Velocity

Per configurare Velocity sufficiente inserire le sue librerie in formato .jar insieme alle altre librerie del progetto e alle librerie ti Tomcat.

Un'altra configurazione da effettuare è nel file web.xml di Tomcat, qui infatti bisogna specificare che le richieste ai file .vm fanno riferimento a Velocity. La righe del file web.xml che configurano Velocity sono le seguenti:

```
<servlet-mapping>
     <servlet-name>velocity</servlet-name>
     <url-pattern>*.vm</url-pattern>
</servlet-mapping>
```

Si nota infatti che tutti gli url che terminano con .vm saranno gestiti da Velocity.

Capitolo 8

Test e Statistiche

In questo capitolo verranno mostrati dei test di utilizzo dell' applicazione, in particolare test relativi alla verifica del ritardo di un treno e alla richiesta di un generico treno. Inoltre saranno mostrate anche alcune statistiche che indicano se l'applicazione riesce effettivamente a dare risposte soddisfacenti alle domande di un campione di utenti.

8.1 Test

Il primo test che si effettuerà sarà quello relativo alla ricerca di un generico treno. Si potrà così vedere che tipo di input può essere inserito e come il sistema risponde.

Innanzitutto vediamo la pagina iniziale dell'applicazione che è la seguente:

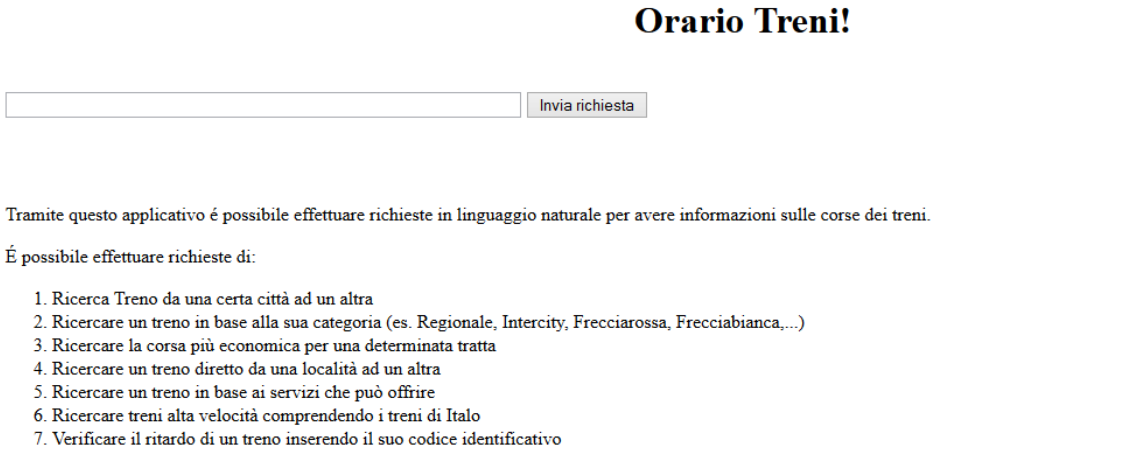

Figura 8. 1 – Schermata di Homepage dell'applicazione

Si può notare la form per la richiesta da parte dell'utente e il bottone "Invia Richiesta" che permette di effettuare la submit e quindi eseguire la richiesta.
Sono state inserite anche delle istruzioni sulle tipologie di domande a cui l'applicazione può rispondere.

TEST N° 1

Effettuiamo il test sulla seguente richiesta inserita dall' utente:

Ricerca di un treno diretto da Carpi a Bologna Regionale in data 12/03/2015

La richiesta è abbastanza restrittiva, infatti si vuole cercare un treno che sia Regionale e senza cambi in una certa data.
Ecco il risultato:

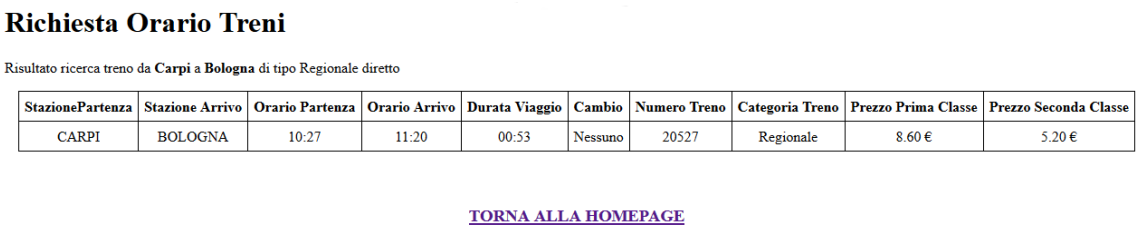

Figura 8.2 – Risultato di un elaborazione sulla richiesta di un treno

Si può notare che è stato trovato un solo treno con le caratteristiche richieste, infatti quasi tutti i treni da Carpi a Bologna fanno cambio a Modena e quindi questo li ha esclusi dai risultati.

TEST N° 2

Effettuiamo ora il test sull'altra tipologia di domande che l'utente può richiedere all' applicazione, ovvero le domande riguardanti la situazione di ritardo di un determinato treno. La richiesta è la seguente

È in ritardo il treno 551?

Il treno 551 è stato scelto perché al momento del test era segnalato sul portale di Trenitalia come uno dei treni in ritardo. E infatti l'applicazione restituisce il risultato:

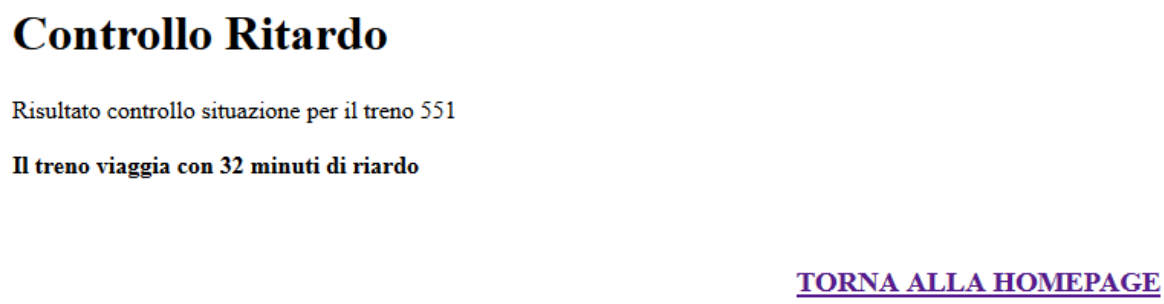

Figura 8.3 – Risultato di un elaborazione sulla situazione di ritardo di un treno

che è conforme al dato riportato dal sito ufficiale di Trenitalia.

8.2 Statistiche

La statistica è stilata su un campione di 35 domande che ricoprivano tutte le categorie di domande che l'applicazione può gestire. Alcune domande erano complesse e richiedevano diverse restrizioni sui risultati.

L’applicazione “OrarioTreni” è riuscita a rispondere e a restituire un risultato soddisfacente in 32 casi sui 35 testati pari a poco più di un 91%.

Le domande che hanno generato un risultato negativo erano quelle che non rispettavano la struttura delle regole scritte in COGITO Studio, ad esempio domande come “Treno Modena Bologna” o “treno da modena a bologna” non sono state gestite correttamente; questo comportamento è dovuto alla mancanza di preposizioni nel primo tipo di domanda e alla scrittura dei nomi di città in minuscolo che non vengono più interpretati da COGITO come nomi di città.

I risultati sono comunque soddisfacenti e il 91% può arrivare, con qualche accorgimento, a crescere. I possibili miglioramenti sono trattati nel capitolo successivo relativo agli sviluppi futuri.

Le domande testate invece saranno allegate nell’ appendice al termine del documento.

Capitolo 9

Conclusioni e sviluppi futuri

Per concludere ricapitoliamo il tutto tramite un'immagine che indica fase per fase il lavoro svolto partendo dalla richiesta dell'utente fino ad arrivare all' output finale:

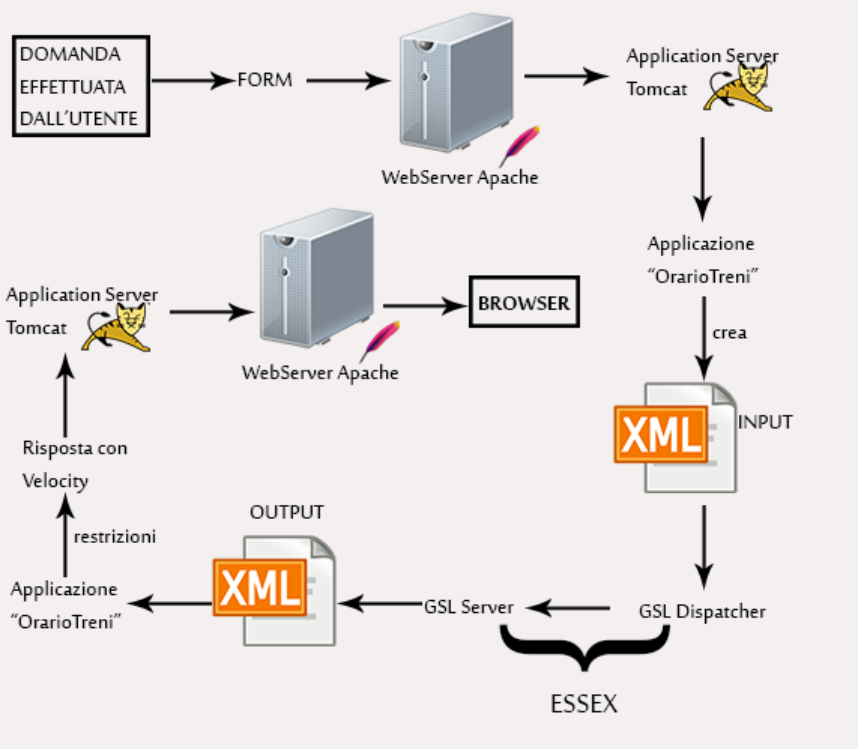

Figura 9.1 – Schema riassuntivo del processo

Considerando i risultati ottenuti ed analizzati nel capitolo precedente si può anche dire che l'applicazione comunque risulta solida e può essere una buona base per ulteriori estensioni.
Nel futuro sarebbe possibile migliorare l'applicazione migliorando quei possibili errori a cui è possibile incorrere, ad esempio sarebbe possibile ricavare i nomi delle città anche se non sono preceduti da alcuna preposizione o se sono scritti in minuscolo.

Per quanto riguarda ulteriori funzionalità dell'applicazione si potrebbe implementare una funzionalità per la quale, data una domanda dell'utente, sia possibile rispondere in modo "Affermativo" o "Negativo" o tramite una risposta diretta. Ad esempio alla domanda "qual è il prossimo treno che passa da Modena per Mantova" il sistema dovrebbe rispondere riportando il numero del treno, oppure alla domanda "il treno numero 5563 ha le cuccette" il sistema dovrebbe rispondere "sì" o "no". Il futuro migliore di questa applicazione, dopo molti sviluppi, sarebbe quello di arrivare a sostituirsi al personale del centro informazione di Trenitalia, ma anche al personale della biglietteria, magari consentendo all' applicazione di poter prenotare anche i biglietti. Si potrebbe riuscire ad arrivare dunque, dopo gli opportuni sviluppi, ad avere un'applicazione completa che gestisca tutte le problematiche tra utente dei servizi ferroviari ed un responsabile dei servizi ferroviari, salvo ovviamente i piloti ed i controllori.

Un altro sviluppo estremamente interessante ed utile sarebbe quello di rendere l'applicazione fruibile direttamente tramite la ricerca sui motori di ricerca, come ad esempio "Google".

Appendice A

Codice delle classi in Java

CLASSE “Base Servlet”

```
package com.orariotreni2;
import java.io.File;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.log4j.Logger;
import org.apache.log4j.xml.DOMConfigurator;

@SuppressWarnings("serial")
public class BaseServlet extends HttpServlet {
   static final Logger log = Logger.getLogger(BaseServlet.class);

   public void doGet(HttpServletRequest req, HttpServletResponse res)
         throws IOException, ServletException {
      doPost(req, res);
   }

   public void doPost(HttpServletRequest req, HttpServletResponse res)
         throws IOException, ServletException {
      ServletContext context = req.getServletContext();
      final String webappBase = context.getRealPath("/");
      DOMConfigurator.configure(webappBase + File.separator + "WEB-INF"
            + File.separator + "config" + File.separator + "log4j.xml");
      execute(req, res);
   }

   public void done(HttpServletRequest request, HttpServletResponse response,
         String templateName) {
      try {
         log.debug("fine");

         // get UI
         RequestDispatcher requestDispatcher = request
               .getRequestDispatcher("WEB-INF/template/" + templateName
                     + ".vm");

         requestDispatcher.forward(request, response);
      } catch (ServletException e) {
         e.printStackTrace();
      } catch (IOException e) {
         e.printStackTrace();
      }
   }

   public void execute(HttpServletRequest request, HttpServletResponse
response) {

   }
}
```

CLASSE “Scheda”

```
package com.orariotreni2;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.log4j.Logger;

@SuppressWarnings("serial")
public class Scheda extends BaseServlet {
   static final Logger log = Logger.getLogger(Scheda.class);

   public void execute(HttpServletRequest request, HttpServletResponse
response) {
      try {
         done(request, response, "scheda");
      } catch (Exception e) {
         log.error("Eccezione", e);
      }
   }

}
```

CLASSE “LeggiInput”

```
package com.orariotreni2;
import java.net.CookieHandler;
import java.net.CookieManager;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.log4j.Logger;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import com.orariotreni2.beans.Comparators;
import com.orariotreni2.beans.ResocontoOrari;
import com.orariotreni2.beans.Response;

@SuppressWarnings("serial")
public class LeggiInput extends BaseServlet {
   static final Logger log = Logger.getLogger(LeggiInput.class);
   Response res = new Response();
   int cercaTreni = 0;
   int ritardo = 0;
   int TAV = 0;
   int diretto = 0;
   int migliorPrezzo = 0;
   int tipoT = 0;
   int servizio = 0;

   public void execute(HttpServletRequest request, HttpServletResponse
response) {
      try {
         String query = request.getParameter("text1");
         String contenutoxml = new String("<?xml version='1.0' encoding='utf-
8'?>" + "<REQUEST VERSION='13.6.1'>" + "<DOCUMENT TYPE='SECTIONS'>" +
"<SECTION NAME='BODY'>"
               + "<![CDATA[##DOMANDA##]]>" + "</SECTION>" + "</DOCUMENT>" +
"<ANALYSIS NAME='CATEGORIZATION'><PARAM NAME='TYPE' VALUE='ALL'/></ANALYSIS>"
               + "<ANALYSIS NAME='EXTRACTION'/>" + "</REQUEST>");
```

```
        if (query.contains("E'") == true) {
           System.out.println("Entra nell'if");
           query = query.replaceAll("E'", "è");
        }
        contenutoxml = contenutoxml.replaceFirst("##DOMANDA##", query);
        final SingleThreadEx thread = new SingleThreadEx("localhost", 5000,
1139, contenutoxml, new Callback() {
           @Override
           public void callback(String result) {
              if (result == null)
                 return;
              final Document doc = ConvertXml.convertStringToDocument(result);
              res = populateResponse(doc, res);
              res = populateExtraction(doc, res);

              for (String categoria : res.getCategoire()) {
                 if (categoria.equals("CercaT") == true)
                    cercaTreni = 1;
                 else if (categoria.equals("Ritardo") == true)
                    ritardo = 1;
                 else if (categoria.equals("TAV") == true)
                    TAV = 1;
                 else if (categoria.equals("Diretto") == true)
                    diretto = 1;
                 else if (categoria.equals("Migliorprezzo") == true)
                    migliorPrezzo = 1;
                 else if (categoria.equals("TipoT") == true)
                    tipoT = 1;
                 else if (categoria.equals("Servizio") == true)
                    servizio = 1;
              }
              if (ritardo == 1) {
                 System.out.println("Entra nell' if");
                 try {
                    System.out.println("Entra nel try");
                    request.setAttribute("ntreno", res.getNumero());
                    request.setAttribute("result",
Richiestaritardo(res.getNumero()));
                 } catch (Exception e) {
                    System.out.println("Eccezione lanciata" + e);
                 }
              } else if (cercaTreni == 1 && TAV == 0) {
                 try {
                    System.out.println("Esegue la ricerca del treno");
                    request.setAttribute("partenza", res.getPartenza());
                    request.setAttribute("arrivo", res.getArrivo());
                    List<ResocontoOrari> resocontro = new
ArrayList<ResocontoOrari>();
                    resocontro = RichiestaTreno(res.getPartenza(),
res.getArrivo(), res.getData());
                    if (servizio == 1) {
                       if (res.getServizio().toLowerCase().equals("biciclette"))
{
                          for (int i = 0; i < resocontro.size(); i++) {
                             if
(resocontro.get(i).getCategoria().toLowerCase().contains("intercity") == true
                                   ||
resocontro.get(i).getCategoria().toLowerCase().contains("freccia") == true) {
                                resocontro.remove(i);
                                i--;
                             }
                          }
                       }
                       if (res.getServizio().toLowerCase().equals("disabili")) {
                          for (int i = 0; i < resocontro.size(); i++)
                             if
```

```
(resocontro.get(i).getCategoria().toLowerCase().contains("regionale") == true
                                 ||
resocontro.get(i).getCategoria().toLowerCase().contains("notte") == true) {
                              resocontro.remove(i);
                              i--;
                           }
                        }

                        if (res.getServizio().toLowerCase().equals("cuccette")) {
                           for (int i = 0; i < resocontro.size(); i++)
                              if
(resocontro.get(i).getCategoria().toLowerCase().contains("regionale") == true
                                 ||
resocontro.get(i).getCategoria().toLowerCase().contains("freccia") == true) {
                              resocontro.remove(i);
                              i--;
                           }
                        }

                        if (res.getServizio().toLowerCase().equals("ristorante")
|| res.getServizio().toLowerCase().equals("ristorazione")) {
                           for (int i = 0; i < resocontro.size(); i++)
                              if
(resocontro.get(i).getCategoria().toLowerCase().contains("bianca") == true
                                 ||
resocontro.get(i).getCategoria().toLowerCase().contains("regionale") == true
                                 ||
resocontro.get(i).getCategoria().toLowerCase().contains("intercity") == true)
{
                              resocontro.remove(i);
                              i--;
                           }
                        }

                        if (res.getServizio().toLowerCase().equals("bar")) {
                           System.out.println("Richiesto servizio bar");
                           for (int i = 0; i < resocontro.size(); i++)
                              if
(resocontro.get(i).getCategoria().toLowerCase().contains("intercity") == true
                                 ||
resocontro.get(i).getCategoria().toLowerCase().contains("regionale") == true)
{
                              resocontro.remove(i);
                              i = i - 1;
                           }
                        }

                     }
                     if (migliorPrezzo == 0 && tipoT == 0) {
                        request.setAttribute("dimensione", resocontro.size());
                        request.setAttribute("ElencoTreni", resocontro);
                        request.setAttribute("richiestatipo", "");
                     } else if (tipoT == 1 && migliorPrezzo == 0) {
                        System.out.println("Entra in tipoT");
                        List<ResocontoOrari> resocontroTipo = new
ArrayList<ResocontoOrari>();
                        String tipologiarichiesta = res.getTipo();
                        for (int i = 0; i < resocontro.size(); i++) {
                           if
(resocontro.get(i).getCategoria().toLowerCase().contains(tipologiarichiesta.to
LowerCase()) == true) {
                              resocontroTipo.add(resocontro.get(i));
                           }
                        }
                        request.setAttribute("dimensione",
resocontroTipo.size());
```

```
                    request.setAttribute("ElencoTreni", resocontroTipo);
                    request.setAttribute("richiestatipo", "di tipo " +
res.getTipo());
                } else if (migliorPrezzo == 1 && tipoT == 0) {
                    int i;
                    double min = 0.0;
                    Collections.sort(resocontro, Comparators.PREZZO);
                    List<ResocontoOrari> resocontroOrdinato = new
ArrayList<ResocontoOrari>();

                    for (i = 0; i < resocontro.size(); i++) {
                       if (resocontro.get(i).getPrezzoseconda().contains("ND")
== false) {
                           System.out.println("Entra nell' inserimento del
minimo ND");
                           if (min == 0.0
                                 ||
Double.parseDouble(resocontro.get(i).getPrezzoseconda().substring(0,
resocontro.get(i).getPrezzoseconda().length() - 3)) < min) {
                              System.out.println("Trovato un nuovo minimo");
                              min =
Double.parseDouble(resocontro.get(i).getPrezzoseconda().substring(0,
resocontro.get(i).getPrezzoseconda().length() - 3));
                              resocontroOrdinato.clear();
                              resocontroOrdinato.add(resocontro.get(i));
                           } else if (min == 0.0
                                 ||
Double.parseDouble(resocontro.get(i).getPrezzoseconda().substring(0,
resocontro.get(i).getPrezzoseconda().length() - 3)) == min) {
                              System.out.println("Trovato un minimo uguale");
                              resocontroOrdinato.add(resocontro.get(i));
                           }
                       }
                    }
                    request.setAttribute("ElencoTreni", resocontroOrdinato);
                    request.setAttribute("dimensione",
resocontroOrdinato.size());
                    request.setAttribute("richiestatipo", "in base al treno
più economico");
                } else {
                    System.out.println("Entra in tipoT Ultimo");
                    String tipologiarichiesta = res.getTipo();
                    int i;
                    double min = 0.0;
                    List<ResocontoOrari> resocontroCategoria = new
ArrayList<ResocontoOrari>();
                    for (i = 0; i < resocontro.size(); i++) {
                       if
(resocontro.get(i).getCategoria().toLowerCase().contains(tipologiarichiesta.to
LowerCase()) == true)
                           resocontroCategoria.add(resocontro.get(i));
                    }
                    List<ResocontoOrari> resocontroOrdinato = new
ArrayList<ResocontoOrari>();
                    for (i = 0; i < resocontroCategoria.size(); i++) {
                       if
(resocontroCategoria.get(i).getPrezzoseconda().contains("ND") == false) {
                           System.out.println("Entra nell' inserimento del
minimo ND");
                           if (min == 0.0
                                 ||
Double.parseDouble(resocontroCategoria.get(i).getPrezzoseconda()
                                       .substring(0,
resocontroCategoria.get(i).getPrezzoseconda().length() - 3)) < min) {
                              System.out.println("Trovato un nuovo minimo");
                              min =
```

```
Double.parseDouble(resocontroCategoria.get(i).getPrezzoseconda()
                                    .substring(0,
resocontroCategoria.get(i).getPrezzoseconda().length() - 3));
                                resocontroOrdinato.clear();

   resocontroOrdinato.add(resocontroCategoria.get(i));
                            } else if (min == 0.0
                                    ||
Double.parseDouble(resocontroCategoria.get(i).getPrezzoseconda()
                                        .substring(0,
resocontroCategoria.get(i).getPrezzoseconda().length() - 3)) == min) {
                                System.out.println("Trovato un minimo uguale");

   resocontroOrdinato.add(resocontroCategoria.get(i));
                                }
                            }
                        }
                        request.setAttribute("ElencoTreni", resocontroOrdinato);
                        request.setAttribute("dimensione",
resocontroOrdinato.size());
                        request.setAttribute("richiestatipo", "in base al treno
più economico");
                    }
                } catch (Exception e) {
                    System.out.println("Eccezione lanciata" + e);
                }
            } else if (cercaTreni == 1 && TAV == 1) {
                try {
                    System.out.println("Esegue la ricerca del treno alta
velocità");
                    List<ResocontoOrari> resocontro = new
ArrayList<ResocontoOrari>();
                    request.setAttribute("partenza", res.getPartenza());
                    request.setAttribute("arrivo", res.getArrivo());
                    resocontro = RichiestaTAV(res.getPartenza(),
res.getArrivo(), res.getData());
                    request.setAttribute("dimensione", resocontro.size());
                    request.setAttribute("ElencoTreni", resocontro);
                    request.setAttribute("richiestatipo", "di tipo Alta
Velocità");
                    request.setAttribute("migliorTreno", true);
                } catch (Exception e) {
                    System.out.println("Eccezione in CERCATAV" + e);
                }
            }
        }

        private String Richiestaritardo(String num) throws Exception {
            String url =
"http://m.viaggiatreno.it/vt_pax_internet/mobile/numero";
            CookieHandler.setDefault(new CookieManager());
            CallFormNew http = new CallFormNew();
            System.out.println("Chiamata CallFormNew effettuata");
            return (http.sendPost(url, num));
        }

        private List<ResocontoOrari> RichiestaTreno(String partenza, String
arrivo, String data) throws Exception {
            CookieHandler.setDefault(new CookieManager());
            CallFormNew http = new CallFormNew();
            System.out.println("Chiamata CallFormNew effettuata");
            return
(http.getPageContent("http://orario.trenitalia.com/b2c/nppPriceTravelSolutions
.do?", partenza, arrivo, data));
        }
```

```
            private List<ResocontoOrari> RichiestaTAV(String partenza, String
arrivo, String data) throws Exception {
                CookieHandler.setDefault(new CookieManager());
                CallFormNew http = new CallFormNew();
                System.out.println("Chiamata CallFormNew effettuata");
                List<ResocontoOrari> risultatoEasyTreno =
http.GetPageContentTAV("http://trainsws-
liannario.rhcloud.com/api/trains/search?", partenza, arrivo, data);
                return risultatoEasyTreno;
            }

        });
        thread.run();
      } catch (Exception e) {
        log.error("Eccezione", e);
      } finally {
        if (ritardo == 1) {
          ritardo = 0;
          done(request, response, "ritardo");
        } else if (migliorPrezzo == 1 && cercaTreni == 1)
          request.setAttribute("operazione", "Richiesta Treni in Base al
prezzo migliore");
        else if (migliorPrezzo == 0 && cercaTreni == 1 && TAV == 0) {
          request.setAttribute("op", 0);
          request.setAttribute("operazione", "Richiesta Orario Treni");
        } else if (cercaTreni == 1 && TAV == 1) {
          request.setAttribute("operazione", "Richiesta Orario TAV");
          request.setAttribute("op", 1);
        }
        migliorPrezzo = 0;
        cercaTreni = 0;
        TAV = 0;
        if (diretto == 1 && tipoT == 1) {
          request.setAttribute("diretto", 1);
          request.setAttribute("operazione2", "diretto");
        } else if (diretto == 1 && tipoT == 0) {
          request.setAttribute("diretto", 1);
          request.setAttribute("operazione2", "diretto");
          request.setAttribute("richiestatipo", "");
        } else {
          request.setAttribute("diretto", 0);
          request.setAttribute("operazione2", "");
        }
        diretto = 0;
        done(request, response, "trovatreni");
      }
    }

    private Response populateResponse(Document doc, Response res) {
      final List<String> categorie = new ArrayList<String>();
      final NodeList nList = doc.getElementsByTagName("CATEGORIZATION");
      for (int temp = 0; temp < nList.getLength(); temp++) {
        final Node nNode = nList.item(temp);
        if (nNode.getNodeType() == Node.ELEMENT_NODE) {
          final Element eElement = (Element) nNode;
          for (temp = 0; temp < eElement.getChildNodes().getLength(); temp++)
{
            final Node nNodeC = eElement.getChildNodes().item(temp);
            if (nNodeC.getNodeType() == Node.ELEMENT_NODE) {
              final Element eElementC = (Element) nNodeC;
              categorie.add(eElementC.getAttribute("NAME"));
            }
          }
          res.setCategoire(categorie);
        }
      }
```

```
      return res;
   }

   private Response populateExtraction(Document doc, Response res) {

      final NodeList nList2 = doc.getElementsByTagName("EXTRACTION");
      for (int temp = 0; temp < nList2.getLength(); temp++) {
         Node nNode = nList2.item(temp);
         if (nNode.getNodeType() == Node.ELEMENT_NODE) {
            Element eElement = (Element) nNode;
            for (int temp2 = 0; temp2 < eElement.getChildNodes().getLength();
temp2++) {
               Node nNodeC = eElement.getChildNodes().item(temp2);
               if (nNodeC.getNodeType() == Node.ELEMENT_NODE) {
                  Element eElementC = (Element) nNodeC;
                  for (int temp3 = 0; temp3 <
eElementC.getChildNodes().getLength(); temp3++) {
                     Node nNodeD = eElementC.getChildNodes().item(temp3);
                     if (nNodeD.getNodeType() == Node.ELEMENT_NODE) {
                        Element eElementD = (Element) nNodeD;
                        System.out.println("aa" +
eElementD.getAttribute("NAME"));
                        if (eElementD.getAttribute("NAME").equals("DATA"))
                           res.setData(eElementD.getAttribute("BASE"));
                        else if (eElementD.getAttribute("NAME").equals("NUMERO"))
                           res.setNumero(eElementD.getAttribute("BASE"));
                        else if
(eElementD.getAttribute("NAME").equals("PARTENZA"))
                           res.setPartenza(eElementD.getAttribute("BASE"));
                        else if (eElementD.getAttribute("NAME").equals("ARRIVO"))
                           res.setArrivo(eElementD.getAttribute("BASE"));
                        else if (eElementD.getAttribute("NAME").equals("TIPO"))
                           res.setTipo(eElementD.getAttribute("BASE"));
                        else if (eElementD.getAttribute("NAME").equals("TIPOS"))
                           res.setServizio(eElementD.getAttribute("BASE"));
                     }
                  }
               }
            }
         }
      }
      return res;
   }
}
```

CLASSE “CallFormNew”

```
package com.orariotreni2;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.CookieHandler;
import java.net.CookieManager;
import java.net.CookiePolicy;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import org.apache.http.Header;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
```

```
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.http.message.BasicNameValuePair;
import org.jsoup.Jsoup;
import org.jsoup.nodes.Document;
import org.jsoup.nodes.Element;
import org.jsoup.select.Elements;
import com.orariotreni2.beans.ResocontoOrari;

public class CallFormNew {
   private String cookies;
   private Map<String, String> mappaCookie = new HashMap<String, String>();
   private HttpClient client = HttpClientBuilder.create().build();
   private final String USER_AGENT = "Mozilla/5.0";
   private String responso = "";

   public String sendPost(String url, String num) throws Exception {
      System.out.println("Entra in sendPost");
      HttpPost post = new HttpPost(url);


      post.setHeader("User-Agent", USER_AGENT);
      post.setHeader("Accept-Language", "en-US,en;q=0.5");
      post.setHeader("Cookie", getCookies());
      post.setHeader("Connection", "keep-alive");

      List<NameValuePair> paramList = new ArrayList<NameValuePair>();
      paramList.add(new BasicNameValuePair("numeroTreno", num));
      paramList.add(new BasicNameValuePair("tipoRicerca", "numero"));
      paramList.add(new BasicNameValuePair("lang", "IT"));

      post.setEntity(new UrlEncodedFormEntity(paramList));
      HttpResponse response = client.execute(post);

      int responseCode = response.getStatusLine().getStatusCode();

      System.out.println("\nSending 'POST' request to URL : " + url);
      System.out.println("Response Code : " + responseCode);

      BufferedReader rd = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));

      StringBuffer result = new StringBuffer();
      String line = "";
      while ((line = rd.readLine()) != null) {
         result.append(line);
      }

      // set cookies
      setCookies(response.getFirstHeader("Set-Cookie") == null ? "" :
response.getFirstHeader("Set-Cookie").toString());

      Header[] headers = response.getHeaders("Set-Cookie");
      for (Header h : headers) {
         if (h.getValue().indexOf("cFID") != -1) {
            mappaCookie.put("cFID",
h.getValue().substring(h.getValue().indexOf("cFID=") + 5));
         }
      }
      String risultato = result.toString();
      risultato = risultato.replaceAll("^\\s+", " ");
      if (risultato.contains("treno non valido") == true) {
         responso = "Il numero del treno non è valido o non è univoco";
      } else if (risultato.contains("non e&#039; ancora partito")) {
```

```
         responso = "Il treno non è ancora partito";
      } else if (risultato.contains("minuti di ritardo") == true) {
         int n = risultato.indexOf("minuti di ritardo");
         responso = "Il treno viaggia con " + risultato.subSequence(n - 3, n) +
"minuti di riardo";
      } else {
         responso = "Il treno è in orario";
      }

   System.out.println("_____________________________________________________________
_______");
      return responso;
   }

   @SuppressWarnings("deprecation")
   public List<ResocontoOrari> getPageContent(String url, String partenza,
String arrivo, String data) throws Exception {
      String param1 = "car=0";
      String param2 = "&stazin=" + partenza.replace(" ", "+");
      String param3 = "&stazout=" + arrivo.replace(" ", "+");
      String param4 = "";
      String param5 = "";
      String param6 = "";
      if (data == "") {
         Calendar now = Calendar.getInstance();
         param4 = "&datag=" + now.get(Calendar.DAY_OF_MONTH);
         param5 = "&datam=" + (now.get(Calendar.MONTH) + 1);
         param6 = "&dataa=" + now.get(Calendar.YEAR);
      } else {
         try {
            if (data.length() == 10 || data.length() == 5)
               data = "0" + data;
            else if (data.indexOf("-") == data.lastIndexOf("-")) {
               Calendar now = Calendar.getInstance();
               data = data + "-" + now.get(Calendar.YEAR);
               System.out.println(data);
            }
            String dataaaa = data;
            SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy",
Locale.ITALIAN);
            SimpleDateFormat sdfRev = new SimpleDateFormat("dd-MM-yyyy");
            Date dataDate = sdf.parse(dataaaa);
            param4 = "&datag=" + sdfRev.format(dataDate).substring(0, 2);
            param5 = "&datam=" + sdfRev.format(dataDate).substring(3, 5);
            param6 = "&dataa=" + sdfRev.format(dataDate).substring(6, 10);
         } catch (ParseException e) {
            e.printStackTrace();
         }
      }

      String param7 =
"&timsh=00&timsm=00&nreq=50&lang=it&channel=tcom&npag=1&lang_r=it&channel_r=tc
om&npag_r=1";

      String cFID = "";
      if (!mappaCookie.isEmpty()) {
         cFID = "&cFID=" + mappaCookie.get("cFID");
      }

      HttpGet request = new HttpGet(url + cFID + param1 + param2 + param3 +
param4 + param5 + param6 + param7);

      request.setHeader("User-Agent", USER_AGENT);
      request.setHeader("Accept",
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
      request.setHeader("Accept-Language", "en-US,en;q=0.5");
```

```
      request.setHeader("Cookie", getCookies());

      HttpResponse response = client.execute(request);
      BufferedReader rd = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));

      StringBuffer result = new StringBuffer();
      String line = "";
      while ((line = rd.readLine()) != null) {
         result.append(line);
      }

      setCookies(response.getFirstHeader("Set-Cookie") == null ? "" :
response.getFirstHeader("Set-Cookie").toString());

      Header[] headers = response.getHeaders("Set-Cookie");
      for (Header h : headers) {
         if (h.getValue().indexOf("cFID") != -1) {
            mappaCookie.put("cFID",
h.getValue().substring(h.getValue().indexOf("cFID=") + 5));
         }
      }

      org.jsoup.nodes.Document doc = Jsoup.parse(result.toString());
      List<ResocontoOrari> resocontro = new ArrayList<ResocontoOrari>();
      for (Element table : doc.select("table")) {
         for (Element row : table.select("tr")) {
            Elements tds = row.select("td");
            if (table.attr("class").equals("detagglio") == false) {
               if (tds.size() == 8) {
                  ResocontoOrari resOr = new ResocontoOrari();
                  resOr.setStazionepartenza(tds.get(0).text().substring(6,
tds.get(0).text().length()));
                  resOr.setOrapartenza(tds.get(0).text().substring(0, 5));
                  resOr.setStazionearrivo(tds.get(1).text().substring(6,
tds.get(1).text().length()));
                  resOr.setOrarrivo(tds.get(1).text().substring(0, 5));
                  resOr.setDurata(tds.get(2).text());
                  resOr.setNumerotreno(tds.get(3).text());
                  resOr.setCategoria(tds.get(4).select("img").attr("alt"));
                  if (resOr.getCategoria().contains("Regionale")) {
                     resOr.setCategoria("Regionale");
                  }
                  if (tds.get(5).text().contains("ND") == false) {
                     resOr.setPrezzoprima(tds.get(5).text().replace(",",
".").substring(0, tds.get(5).text().length() - 2));
                     resOr.setPrezzoseconda(tds.get(6).text().replace(",",
".").substring(0, tds.get(6).text().length() - 2));
                  } else {
                     resOr.setPrezzoprima(tds.get(5).text());
                     resOr.setPrezzoseconda(tds.get(6).text());
                  }

                  resocontro.add(resOr);
               }
               if (tds.size() == 9) {
                  ResocontoOrari resOr = new ResocontoOrari();

                  resOr.setStazionepartenza(tds.get(0).text().substring(6,
tds.get(0).text().length()));
                  resOr.setOrapartenza(tds.get(0).text().substring(0, 5));
                  resOr.setStazionearrivo(tds.get(1).text().substring(6,
tds.get(1).text().length()));
                  resOr.setOrarrivo(tds.get(1).text().substring(0, 5));
                  resOr.setDurata(tds.get(2).text());
                  resOr.setCambio("");
```

```
                resOr.setNumerotreno(tds.get(4).text());
                resOr.setCategoria(tds.get(5).select("img").attr("alt"));
                // Non voglio distinzioni tra Regionale e Regionale
                // Veloce
                if (resOr.getCategoria().contains("Regionale")) {
                   resOr.setCategoria("Regionale");
                }
                if (tds.get(7).text().contains("ND") == false) {
                   resOr.setPrezzoprima(tds.get(6).text().replace(",",
".").substring(0, tds.get(6).text().length() - 2));
                   resOr.setPrezzoseconda(tds.get(7).text().replace(",",
".").substring(0, tds.get(7).text().length() - 2));
                } else {
                   resOr.setPrezzoprima(tds.get(6).text());
                   resOr.setPrezzoseconda(tds.get(7).text());
                }

                resocontro.add(resOr);
             }
             if (tds.size() == 3) {
                resocontro.get(resocontro.size() -
1).setCambio(tds.get(0).text());
                resocontro.get(resocontro.size() -
1).setNumerotreno(resocontro.get(resocontro.size() - 1).getNumerotreno() + "
" + tds.get(1).text());
                resocontro.get(resocontro.size() -
1).setCategoria(resocontro.get(resocontro.size() - 1).getCategoria() + "    "
+ tds.get(2).select("img").attr("alt"));
             }
          }
       }
    }
    System.out.println("DONE Richiesta Orario");
    return resocontro;
  }

  public StringBuilder sendPostMigliorTreno(String url, String partenza,
String arrivo, String data, String fasciaOraria, StringBuilder
listaHtmlTrAllTrainAllHours)
       throws Exception {
    CookieHandler.setDefault(new CookieManager(null,
CookiePolicy.ACCEPT_ALL));
    final HttpPost post = new HttpPost(url);
    final SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    final SimpleDateFormat sdfCogito = new SimpleDateFormat("dd-MMM-yyyy",
Locale.ITALIAN);
    final Date dataCogito = sdfCogito.parse(data);
    data = sdf.format(dataCogito);
    final Map<String, String> mappaCodiceDescrizione = new HashMap<String,
String>();
    mappaCodiceDescrizione.put("ancona", "AN");
    mappaCodiceDescrizione.put("bologna", "BO");
    mappaCodiceDescrizione.put("firenze", "FI");
    mappaCodiceDescrizione.put("milano", "MI");
    mappaCodiceDescrizione.put("napoli", "NA");
    mappaCodiceDescrizione.put("padova", "PD");
    mappaCodiceDescrizione.put("pesaro", "PU");
    mappaCodiceDescrizione.put("reggio Emilia", "RE");
    mappaCodiceDescrizione.put("rimini", "RN");
    mappaCodiceDescrizione.put("roma", "RM");
    mappaCodiceDescrizione.put("salerno", "SA");
    mappaCodiceDescrizione.put("torino", "TO");

    if (mappaCodiceDescrizione.containsKey(partenza.toLowerCase())) {
       partenza = mappaCodiceDescrizione.get(partenza.toLowerCase());
    }
```

```
      if (mappaCodiceDescrizione.containsKey(arrivo.toLowerCase())) {
         arrivo = mappaCodiceDescrizione.get(arrivo.toLowerCase());
      }

      final Calendar cal = Calendar.getInstance();
      post.setHeader("Accept",
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
      post.setHeader("Accept-Language", "it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3");
      post.addHeader("Connection", "keep-alive");
      post.setHeader("Referer", "http://www.migliortreno.it/");
      post.addHeader("Referer", "http://www.migliortreno.it/");
      post.setHeader("Content-Type", "application/x-www-form-urlencoded");
      List<NameValuePair> paramList = new ArrayList<NameValuePair>();
      paramList.add(new BasicNameValuePair("MarketStructure", "OneWay"));
      paramList.add(new BasicNameValuePair("DepartureCityCode", partenza));
      paramList.add(new BasicNameValuePair("ArrivalCityCode", arrivo));
      paramList.add(new BasicNameValuePair("GoingOutDate", data));
      paramList.add(new BasicNameValuePair("GoingOutTimeInterval",
fasciaOraria));
      paramList.add(new BasicNameValuePair("PaxAdtCount", "1"));
      paramList.add(new BasicNameValuePair("PaxChdCount", "0"));
      paramList.add(new BasicNameValuePair("PromoCode", ""));
      paramList.add(new BasicNameValuePair("ButtonSubmit", ""));
      cal.setTime(sdf.parse(data));
      cal.getTime();
      cal.add(Calendar.DAY_OF_YEAR, 6);
      final String dataPartenzaFittiza = sdf.format(cal.getTime());
      paramList.add(new BasicNameValuePair("ComingBackDate",
dataPartenzaFittiza)); // obbligatorio
      post.setEntity(new UrlEncodedFormEntity(paramList));
      final HttpResponse response = client.execute(post);
      final int responseCode = response.getStatusLine().getStatusCode();
      // System.out.println("\nSending 'POST' request to URL : " + url);
      // System.out.println("Response Code : " + responseCode);
      InputStreamReader isr = new
InputStreamReader(response.getEntity().getContent());
      BufferedReader rd = new BufferedReader(isr);
      StringBuffer result = new StringBuffer();
      String line = "";
      while ((line = rd.readLine()) != null) {
         result.append(line);
      }
      rd.close();
      isr.close();

      // System.out.println("Result : \n " + result.toString());

      if (responseCode == HttpStatus.SC_OK) {
         final Document document = Jsoup.parse(result.toString());
         final Elements elements = document.select("div.box-table.box-10 >
table > tbody > tr");
         listaHtmlTrAllTrainAllHours.append(elements.toString());
         // System.out.println(elements.toString());
      }
      return listaHtmlTrAllTrainAllHours;
   }

   public List<ResocontoOrari> GetPageContentTAV(String url, String partenzaa,
String arrivoo, String dataa) throws Exception {
      String citta[] = { "Ancona", "Bologna Centrale", "Firenze S. M. Novella",
"Milano", "Milano Centrale", "Milano Porta Garibaldi", "Milano Rogoredo",
"Padova",
            "Napoli Centrale", "Reggio Emilia AV", "Rimini", "Roma", "Roma
Termini", "Roma Tiburtina", "Roma Ostiense", "Salerno", "Torino", "Torino
Porta Nuova",
```

```
            "Torino Porta Susa", "Venezia", "Venezia S. Lucia", "Venezia Mestre"
};
      List<ResocontoOrari> resocontro = new ArrayList<ResocontoOrari>();
      String partenza = partenzaa;
      String arrivo = arrivoo;
      String data = dataa;
      for (int i = 0; i < citta.length; i++) {
         if (citta[i].toLowerCase().contains(partenza.toLowerCase()) == true) {
            partenza = citta[i];
         }
      }

      for (int i = 0; i < citta.length; i++) {
         if (citta[i].toLowerCase().contains(arrivo.toLowerCase()) == true) {
            arrivo = citta[i];
         }
      }
      String param1 = "rtt=false";
      String param2 = "&from=" + partenza.replace(" ", "+");
      String param3 = "&to=" + arrivo.replace(" ", "+");
      String param4 = "";

      if (data == "") {
         Calendar now = Calendar.getInstance();
         param4 = "&goingd=" + now.get(Calendar.DAY_OF_MONTH) + "-" +
(now.get(Calendar.MONTH) + 1) + "-" + now.get(Calendar.YEAR);
      } else {
         String dataaaa = data;
         SimpleDateFormat sdf = new SimpleDateFormat("dd-MMM-yyyy",
Locale.ITALIAN);
         SimpleDateFormat sdfRev = new SimpleDateFormat("dd-MM-yyyy");
         Date dataDate = sdf.parse(dataaaa);
         param4 = "&goingd=" + sdfRev.format(dataDate).substring(0, 2) + "-" +
sdfRev.format(dataDate).substring(3, 5) + "-" +
sdfRev.format(dataDate).substring(6, 10);
      }
      String param5 = "&goingt=00&adt=1&chl=0&ordprice=true";

      String cFID = "";
      if (!mappaCookie.isEmpty()) {
         cFID = "&cFID=" + mappaCookie.get("cFID");
      }

      System.out.println(url + cFID + param1 + param2 + param3 + param4 +
param5);
      HttpGet request = new HttpGet(url + cFID + param1 + param2 + param3 +
param4 + param5);
      request.setHeader("User-Agent", USER_AGENT);
      request.setHeader("Accept",
"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
      request.setHeader("Accept-Language", "en-US,en;q=0.5");
      request.setHeader("Cookie", getCookies());

      HttpResponse response = client.execute(request);
      int responseCode2 = response.getStatusLine().getStatusCode();
      System.out.println("\nSending 'GET' request to URL : " + url);
      System.out.println("Response Code : " + responseCode2);
      BufferedReader rd = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));
      StringBuffer risposta = new StringBuffer();
      String line = "";
      while ((line = rd.readLine()) != null) {
         risposta.append(line);
      }
      String risultato = risposta.toString();
      System.out.println(risultato);
```

```
      if (risultato.contains("travelsolutions") == true) {
   setCookies(response.getFirstHeader("Set-Cookie") == null ? "" :
response.getFirstHeader("Set-Cookie").toString());

        Header[] headers = response.getHeaders("Set-Cookie");
        for (Header h : headers) {
           if (h.getValue().indexOf("cFID") != -1) {
              mappaCookie.put("cFID",
h.getValue().substring(h.getValue().indexOf("cFID=") + 5));
           }
        }
        org.jsoup.nodes.Document doc = Jsoup.parse(risultato);

        int index = 0;
        while (risultato.indexOf("fromStation", index) != -1) {
           ResocontoOrari resOr = new ResocontoOrari();
           index = risultato.indexOf("fromStation", index);
           resOr.setStazionepartenza(risultato.substring(risultato.indexOf(":",
index) + 1, risultato.indexOf(",", index)).replaceAll("\"", ""));
           index = risultato.indexOf("toStation", index);
           resOr.setStazionearrivo(risultato.substring(risultato.indexOf(":",
index) + 1, risultato.indexOf(",", index)).replaceAll("\"", ""));
           index = risultato.indexOf("startTime", index);
           resOr.setOrapartenza(risultato.substring(risultato.indexOf(":",
index) + 1, risultato.indexOf(",", index)).replaceAll("\"", ""));
           index = risultato.indexOf("endTime", index);
           resOr.setOrarrivo(risultato.substring(risultato.indexOf(":", index)
+ 1, risultato.indexOf(",", index)).replaceAll("\"", ""));
           index = risultato.indexOf("changeTrain", index);
           if (risultato.substring(risultato.indexOf(":", index) + 1,
risultato.indexOf(",", index)).equals("false") == true)
              resOr.setCambio("Nessuno");
           else
              resOr.setCambio("Previsto un cambio");
           index = risultato.indexOf("duration", index);
           resOr.setDurata(risultato.substring(risultato.indexOf(":", index) +
1, risultato.indexOf(",", index)).replaceAll("\"", ""));
           index = risultato.indexOf("train", index);
           resOr.setCategoria(risultato.substring(risultato.indexOf(":", index)
+ 1, risultato.indexOf(",", index)).replaceAll("\"", ""));
           index = risultato.indexOf("price", index);
           resOr.setPrezzoprima(risultato.substring(risultato.indexOf(":",
index) + 1, risultato.indexOf(",", index)));
           index = risultato.indexOf("company", index);
           resOr.setCompagnia(risultato.substring(risultato.indexOf(":", index)
+ 1, risultato.indexOf(",", index)).replaceAll("\"", ""));
           resocontro.add(resOr);
        }
        System.out.println("DONE Richiesta Orario");
        return resocontro;
     } else {
        StringBuilder listaHtmlTrAllTrainAllHours = new StringBuilder();
        List<ResocontoOrari> result = new ArrayList<ResocontoOrari>();
        CallFormNew http = new CallFormNew();
        String urlMigliorTreno = "http://www.migliortreno.it/Schedule/Search";
        listaHtmlTrAllTrainAllHours =
http.sendPostMigliorTreno(urlMigliorTreno, partenzaa, arrivoo, dataa,
"EarlyMorning", listaHtmlTrAllTrainAllHours);
        listaHtmlTrAllTrainAllHours =
http.sendPostMigliorTreno(urlMigliorTreno, partenzaa, arrivoo, dataa,
"Morning", listaHtmlTrAllTrainAllHours);
        listaHtmlTrAllTrainAllHours =
http.sendPostMigliorTreno(urlMigliorTreno, partenzaa, arrivoo, dataa,
"Afternoon", listaHtmlTrAllTrainAllHours);
        listaHtmlTrAllTrainAllHours =
http.sendPostMigliorTreno(urlMigliorTreno, partenzaa, arrivoo, dataa,
```

```
"Evening", listaHtmlTrAllTrainAllHours);
        listaHtmlTrAllTrainAllHours =
http.sendPostMigliorTreno(urlMigliorTreno, partenzaa, arrivoo, dataa, "Night",
listaHtmlTrAllTrainAllHours);
        listaHtmlTrAllTrainAllHours =
http.sendPostMigliorTreno(urlMigliorTreno, partenzaa, arrivoo, dataa,
"LateNight", listaHtmlTrAllTrainAllHours);
        result =
http.migliorTrenoDecode(listaHtmlTrAllTrainAllHours.toString());
        return result;
     }
   }

   public List<ResocontoOrari> migliorTrenoDecode(String result) {
      result = "<html><head></head><body><table>" + result +
"</table></body></html>";
     org.jsoup.nodes.Document doc = Jsoup.parse(result);
     List<ResocontoOrari> listaTreni = new ArrayList<ResocontoOrari>();
     for (Element tr : doc.select("table")) {
        for (Element row : tr.select("tr")) {
           final Elements tds = row.select("td");
           if (tds != null && tds.size() == 5) {
              final ResocontoOrari resOr = new ResocontoOrari();
              final String oraPartenza = tds.get(0).text();
              final String oraArrivo = tds.get(1).text();
              final String durata = tds.get(2).text();
              final String numeroTreno = tds.get(3).select("span").text();
              String prezzo = tds.get(4).select("div").text();
              prezzo = prezzo.replace("€", "");
              String compagnia = tds.get(4).select("span").toString();
              if (compagnia != null) {
                 if (compagnia.contains("frecce"))
                    compagnia = "Trenitalia";
                 else
                    compagnia = "Italo";
              }
              resOr.setNumerotreno(numeroTreno);
              resOr.setCompagnia(compagnia);
              resOr.setDurata(durata);
              resOr.setOrapartenza(oraPartenza.substring(0, 5));
              resOr.setOrarrivo(oraArrivo.substring(0, 5));
              resOr.setStazionepartenza(oraPartenza.substring(5));
              resOr.setStazionearrivo(oraArrivo.substring(5));
              resOr.setPrezzoprima(prezzo);
              listaTreni.add(resOr);
           }

        }
     }
     return listaTreni;
   }

   public String getCookies() {
      return cookies;
   }

   public void setCookies(String cookies) {
      this.cookies = cookies;
   }
}
```

CLASSI DEL PACKAGE “BEANS”

CLASSE “Comparators”

```
package com.orariotreni2.beans;
import java.util.Comparator;

public class Comparators {
   public static Comparator<ResocontoOrari> PREZZO = new
Comparator<ResocontoOrari>() {
      @Override
      public int compare(ResocontoOrari o1, ResocontoOrari o2) {
         return o1.getPrezzoseconda().compareTo(o2.getPrezzoseconda());
      }
   };
}
```

CLASSE “ResocontoOrari”

```
package com.orariotreni2.beans;

public class ResocontoOrari {
   private String stazionepartenza = "";
   private String stazionearrivo = "";
   private String orapartenza = null;
   private String orarrivo = null;
   private String durata = null;
   private String cambio = "";
   private String numerotreno = null;
   private String categoria = null;
   private String prezzoprima = null;
   private String prezzoseconda = null;
   private String compagnia = "";

  METODI SET E GET PER OGNI ELEMENTO DELLA CLASSE


   public int compareTo(ResocontoOrari o) {
      return Comparators.PREZZO.compare(this, o);
   }
}
```

CLASSE “Response”

```
package com.orariotreni2.beans;
import java.util.List;

public class Response {
   private List<String> categorie = null;
   private String data = "";
   private String Numero = "";
   private String Arrivo = "";
   private String Partenza = "";
   private String Tipo = "";
   private String Servizio = "";

  METODI GET E SET PER OGNI ELEMENTO DELLA CLASSE
}
```

Appendice B

Template Velocity

scheda.vm

```
<html>
<head> <title>Orario Treni</title> </head>
<body>
<br>
<br>
<h1><center>Orario Treni!</center></h1>
<br />
<form action="/orariotreni/LeggiInput" name="formscheda" method="post">
   <input type="text"name="text1" size="70"/>
   <input type="submit"/>
</form>
<br>
<br>
<p>Tramite questo applicativo &eacute possibile effettuare richieste in
linguaggio naturale per avere informazioni sulle corse dei treni.</p>
<p>&Eacute possibile effettuare richieste di:</p>
<ol>
   <li>Ricerca Treno da una certa citt&agrave ad un altra</li>
   <li>Ricercare un treno in base alla sua categoria (es. Regionale,
Intercity, Frecciarossa, Frecciabianca,...)
   <li>Ricercare la corsa pi&ugrave economica per una determinata tratta</li>
   <li>Ricercare un treno diretto da una localit&agrave ad un altra</li>
   <li>Ricercare un treno in base ai servizi che pu&ograve offrire</li>
   <li>Ricercare treni alta velocit&agrave comprendendo i treni di Italo</li>
   <li>Verificare il ritardo di un treno inserendo il suo codice
identificativo</li>
</ol>
</body>
</html>
```

trovatreni.vm

```
<html>
<head>
<style>
table{
   border-collapse:collapse;
}
table tr td{
   border:1px solid black;
}
td{
   padding: 6px;
}
</style><title>Richiesta Treni</title> </head>
<body>
<h1>$operazione</h1>
<p>Risultato ricerca treno da <b>$partenza</b> a <b>$arrivo</b> $richiestatipo
$operazione2</p>
#if ($dimensione > 0)
   #if ($operazione == "Richiesta Orario TAV")
```

```
      <table align="center">
      <tr>
      <td><b>Stazione Partenza</b></td>
      <td><b>Stazione Arrivo</b></td>
      <td><b>Orario Partenza</b></td>
      <td><b>Orario Arrivo</b></td>
      <td><b>Durata Viaggio</b></td>
      <td><b>Cambio</b></td>
      <td><b>#if(!$migliorTreno)Categoria - #end Numero Treno</b></td>
      <td><b>Prezzo</b></td>
      <td><b>Compagnia</b></td>
      <tr>
      #foreach ($i in $ElencoTreni)
         #if($foreach.count == $dimensione)
            #break
         #end
         <tr>
            <td>${i.getStazionepartenza()}</td>
            <td>${i.getStazionearrivo()}</td>
            <td>${i.getOrapartenza()}</td>
            <td>${i.getOrarrivo()}</p></td>
            <td>${i.getDurata()}</td>
            <td>$i.getCambio()</td>
            <td>$!{i.getCategoria()} $!i.getNumerotreno()</td>
            <td>${i.getPrezzoprima()}  €</td>
            <td>${i.getCompagnia()}</td>
         </tr>
      #end
      </table>
   #else
      <table align="center">
      <tr>
         <td><b>StazionePartenza</b></td>
         <td><b>Stazione Arrivo</b></td>
         <td><b>Orario Partenza</b></td>
         <td><b>Orario Arrivo</b></td>
         <td><b>Durata Viaggio</b></td>
         <td><b>Cambio</b></td>
         <td><b>Numero Treno</b></td>
         <td><b>Categoria Treno</b></td>
         <td><b>Prezzo Prima Classe</b></td>
         <td><b>Prezzo Seconda Classe</b></td>
      <tr>
      #if ($diretto == 0)
         #foreach ($i in $ElencoTreni)
            #if($foreach.count == $dimensione+1)
               #break
            #end
            <tr>
               <td><center>${i.getStazionepartenza()}</center></td>
               <td><center>${i.getStazionearrivo()}</center></td>
               <td><center>${i.getOrapartenza()}</center></td>
               <td><center>${i.getOrarrivo()}</center></td>
               <td><center>${i.getDurata()}</center></td>
               #if ($i.getCambio() == "")
                  <td><center>Nessuno</center></td>
               #else
                  <td><center>$i.getCambio()</center></td>
               #end
               <td><center>${i.getNumerotreno()}</center></td>
               <td><center>${i.getCategoria()}</center></td>
               #if ($i.getPrezzoprima() == "ND")
                  <td><center>${i.getPrezzoprima()}</center></td>
                  <td><center>${i.getPrezzoseconda()}</center></td>
               #else
                  <td><center>${i.getPrezzoprima()}  €</center></td>
```

```
                <td><center>${i.getPrezzoseconda()}  €</center></td>
             #end
          </tr>
       #end
     #else
       #foreach ($i in $ElencoTreni)
          #if($foreach.count == $dimensione+1)
             #break
          #end
          #if($i.getCambio() == "")
          <tr>
             <td><center>${i.getStazionepartenza()}</center></td>
             <td><center>${i.getStazionearrivo()}</center></td>
             <td><center>${i.getOrapartenza()}</center></td>
             <td><center>${i.getOrarrivo()}</center></td>
             <td><center>${i.getDurata()}</center></td>
             <td><center>Nessuno</center></td>
             <td><center>${i.getNumerotreno()}</center></td>
             <td><center>${i.getCategoria()}</center></td>
             #if ($i.getPrezzoprima() == "ND")
                <td><center>${i.getPrezzoprima()}</center></td>
                <td><center>${i.getPrezzoseconda()}</center></td>
             #else
                <td><center>${i.getPrezzoprima()}  €</center></td>
                <td><center>${i.getPrezzoseconda()}  €</center></td>
             #end
          </tr>
          #end
       #end
     #end
     </table>
   #end
#else
   <p>Non è stato trovato nessun treno che soddisfi le richieste</p>
#end
<br/>
<br>
<center><h3><a    href="http://localhost:8080/orariotreni/Scheda">TORNA    ALLA
HOMEPAGE</a></h3></center>
</body>
</html>
```

ritardo.vm

```
<html>
<head> <title>Controllo Ritardo</title> </head>
<body>
<h1>Controllo Ritardo</h1>
<p>Risultato controllo situazione per il treno $ntreno </p>
<p> <b> $result </b> </p>
<br/>
<center><h3><a    href="http://localhost:8080/orariotreni/Scheda">TORNA    ALLA
HOMEPAGE</a></h3></center>
</body>
</html>
```

Appendice C

File web.xml per la configurazione di Tomcat

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">


    <servlet>
        <servlet-name>Scheda</servlet-name>
        <servlet-class>com.orariotreni2.Scheda</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Scheda</servlet-name>
        <url-pattern>/Scheda</url-pattern>
    </servlet-mapping>
   <servlet>
        <servlet-name>LeggiInput</servlet-name>
        <servlet-class>com.orariotreni2.LeggiInput</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>LeggiInput</servlet-name>
        <url-pattern>/LeggiInput</url-pattern>
    </servlet-mapping>

       <!--    mapping all .vm files to velocity servlets  -->
    <servlet>
        <servlet-name>velocity</servlet-name>
        <servlet-
class>org.apache.velocity.tools.view.servlet.VelocityViewServlet</servlet-
class>
    </servlet>
    <servlet-mapping>
        <servlet-name>velocity</servlet-name>
        <url-pattern>*.vm</url-pattern>
    </servlet-mapping>


   <!-- CONFIG -->
   <session-config>
     <session-timeout>30</session-timeout>
   </session-config>
</web-app>
```

Appendice D

Domande utilizzate per il test dell' applicazione

1. treno con servizio disabili da Modena a Milano
2. treno più economico da Milano a Napoli
3. treno con servizio ristorante da Modena a Brindisi
4. treno da Bologna a Milano Frecciarossa
5. treno con servizio biciclette da Carpi a Modena
6. treno con servizio bar da Modena ad Alba Adriatica
7. ritardo del treno 85402
8. treno ad alta velocità da Modena a Monopoli
9. treno alta velocità da Reggio Emilia ad Ancona
10. Treno alta velocità da Reggio Emilia a Milano
11. treno regionale diretto da Modena a Mantova
12. treno diretto da Modena a Milano l'11/03/2015
13. treno da Milano a Bolzano con servizio ristorazione
14. treno con servizio Animali da Milano a Modena
15. treno Regionale da Milano a Bologna 15/03/2015
16. treno Roma Bologna
17. treno tra ROMA e BOLOGNA
18. orario treni tav DA REGGIO EMILIA A BOLOGNA
19. treno con cuccette da Roma A NAPOLI
20. RITARDO TRENO 592
21. Treno regionale da MANTOVA a VERONA
22. prezzo più basso treno regionale da Bologna a FIRENZE
23. Treno TRA ROMA E BOLOGNA
24. Treno tra ROMA e BOLOGNA
25. Da Carpi a Calalzo
26. Da Domodossola a Aviano
27. Treno Intercity da Pesaro a Ferrara
28. Treno da Crotone a Ragusa con trasporto biciclette
29. Treno da Carpi a Calalzo con trasporto bici
30. Corsa più economica da Torino a Levanto
31. Ritardo del treno numero 3875
32. TAV da Firenze a Napoli
33. treni con servizio disabili da Milano a Bressanone
34. Ritardo treno numero 9723
35. Treno da Milano a Bari il 01/01/2015 diretto

Software utilizzato

Software per il progetto

- ESSEX Web Service di Expert System
- GSL Dispatcher di Expert System
- Apache + Tomcat + Velocity

Software per lo sviluppo

- Eclipse Luna 4.4.1
- COGITO Studio 13.6.4 di Expert System

Bibliografia

Documentazione ExpertSystem

1. Da wiki di Expert System: Cogito Studio Manuals
    a. Cogito Studio User Manuals
    b. Languages Reference Manual
2. Da wiki di Expert System: Product Description