

MDD Front-end

Documentazione di Progetto

Rebucini Leonardo



CdL Ingegneria Informatica, Magistrale

Dipartimento di Ingegneria Gestionale, dell'Informazione e della Produzione

Università degli Studi di Bergamo

Anno Accademico 2022-2023

Contents

1	Interazione 0 - Envisioning	1
1.1	Scopo	2
1.2	Analisi del Contesto	2
1.2.1	Feature Model	2
1.3	Casi d'Uso	3
1.3.1	Use Cases Diagram	3
1.3.2	Descrizione Casi d'Uso	4
1.4	Requisiti Funzionali	8
1.4.1	Elenco Funzionalità	8
1.5	Requisiti Non Funzionali	9
1.5.1	Affidabilità	9
1.5.2	Usabilità	9
1.5.3	Manutenibilità	9
1.5.4	Portabilità	9
1.5.5	Efficienza	9
1.6	Architettura di Sistema	10
1.6.1	Proposta tecnologie	10
1.6.2	Topology Diagram	11
1.6.3	Deployment Diagram	11
2	Iterazione 1 - Setup architettura client-server	12
2.1	Planning	13
2.2	Design	13
2.2.1	Design Diagram - Iterazione 1	13
2.3	Implementazione	14
3	Iterazione 3 - Sviluppo classe MddInstance	15
3.1	Nota iterazione 2	15
3.2	Planning	16

3.3	Design	17
3.3.1	Funzionalità	17
3.3.2	Casi d'uso	18
3.3.3	Algoritmo di ricerca informazioni Feature Model	21
3.4	Diagramma delle classi	22
3.4.1	Class Diagram	23
3.5	Implementazione	24
3.5.1	Analisi Dinamica	24
3.5.2	Copertura del codice	28
3.5.3	Analisi Statica	29
4	Iterazione 4 - Implementazione risposte REST per eccezioni	30
4.1	Planning	31
4.2	Design	31
4.2.1	Diagramma dei componenti	33
5	Bibliografia	34
5.1	Manuale Utente	35
5.1.1	Server Setup	35
5.1.2	Utilizzo client	36
5.2	Toolchain	39

Chapter 1

Interazione 0 - Envisioning

1.1 Scopo

Creazione di un'applicazione web front-end di supporto all'applicazione per la creazione di Multivalued Decision Diagrams (MDDs) per il calcolo delle possibili combinazioni di feature di un particolare sistema modellato sotto forma di Feature Model, per favorire la distribuzione del carico computazionale a dispositivi (server) più adatti.

1.2 Analisi del Contesto

Facente parte dell'analisi di linea di prodotto, la creazione di Multivalued Decision Diagram serve a rappresentare lo spazio di configurazione di una linea di prodotto, rimuovendo la necessità di utilizzo di SAT solvers per il calcolo della soddisfacibilità e fornendo un supporto fondamentale per la Knowledge Compilation. A supporto di tale contesto, l'applicazione supporta funzioni di:

1. Caricamento di Feature Model e calcolo MDD
2. Modifica del diagramma MD calcolato
3. Calcolo e display di informazioni ricavate dal MDD
4. ...

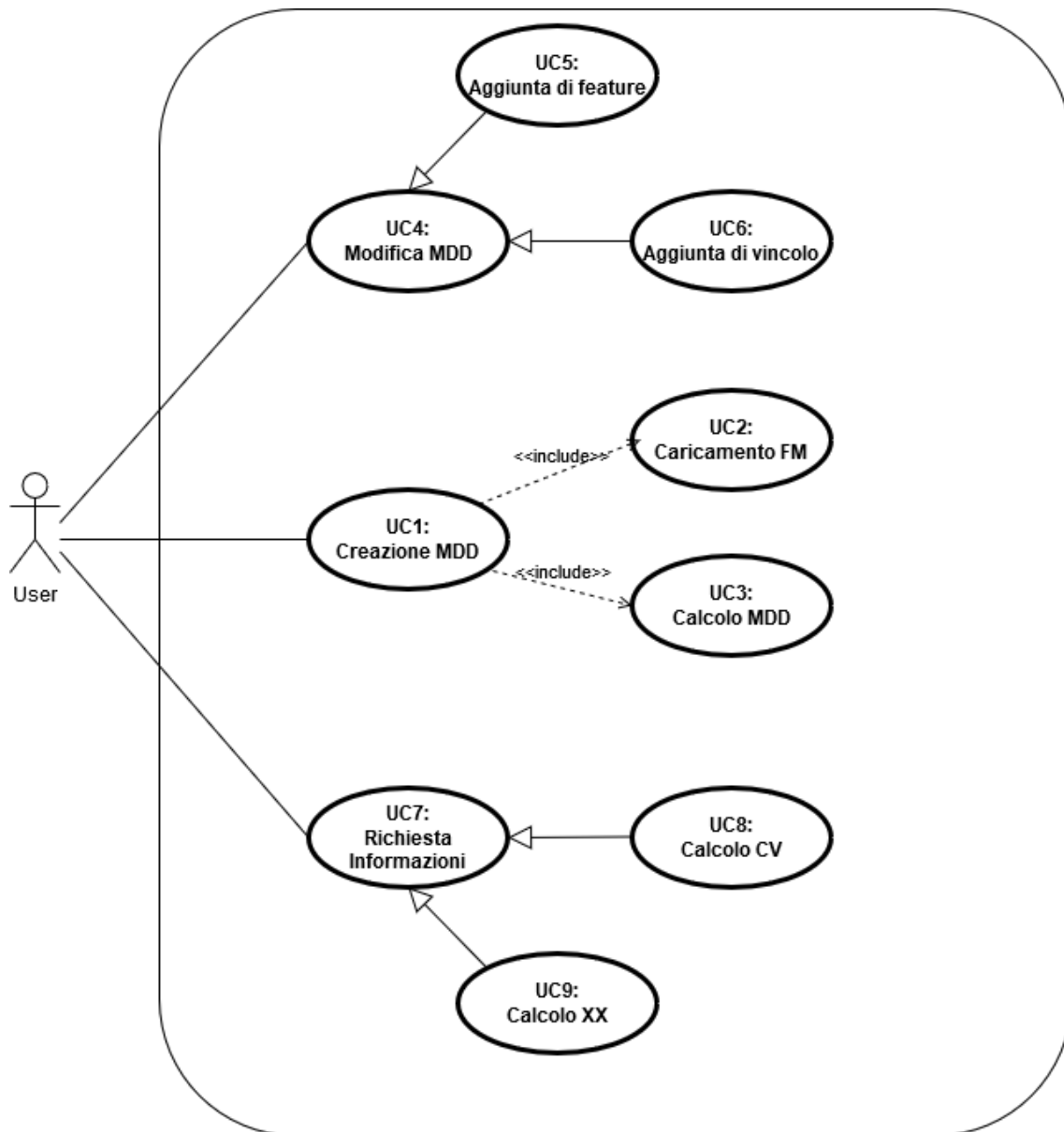
1.2.1 Feature Model

E' il modello su cui si basa l'applicativo e dal quale vengono estrapolate le informazioni necessarie per la creazione del diagramma. Composto da due elementi fondamentali strettamente legati tra loro:

- Features, ciascuna rappresentante una componente del sistema modellato.
- Constraints rappresentanti i legami esistenti tra le varie feature, a loro volta suddivisi in due tipologie:
 1. In-Line Constraint definiti in una struttura ad albero, possono presentarsi come constraint "and", "or" oppure "alt" e ciascuno definisce vincoli per i propri nodi figlio
 2. Cross-Tree Constraint sono invece definiti singolarmente e rappresentano vincoli tra non-sibling nodes

1.3 Casi d'Uso

1.3.1 Use Cases Diagram



1.3.2 Descrizione Casi d'Uso

UC01	Creazione MDD
Description	Caso d'uso astratto rappresentante la procedura completa di creazione del diagramma MD da parte del back-end
Actors	User
Pre-conditions	Utente connesso all'applicazione web
Steps	
Related U.C.	Caricamento FM Calcolo MDD
Exceptions	
Post-conditions	Il server contiene il diagramma MD del modello

UC02	Caricamento Feature Model
Description	Elemento web che permetta all'utente la ricerca, selezione e caricamento di un documento xml nel contesto dell'applicazione web
Actors	User
Pre-conditions	Utente connesso all'applicazione web
Steps	<ol style="list-style-type: none">1. L'utente preme sul pulsante per aprire l'esplora risorse2. L'utente seleziona il Feature Model da caricare3. L'utente conferma il documento4. L'applicazione web invia il documento al server MDD
Related U.C.	Creazione MDD Calcolo MDD
Exceptions	Caricamento Feature Model malformato
Post-conditions	Il server ritorna conferma di ricezione documento

UC03	Calcolo MDD
Actors	User
Description	Ricevuto il FM, l'utente richiede il calcolo dell'MDD
Pre-conditions	Feature Model ricevuto dal server
Steps	<ol style="list-style-type: none"> 1. L'utente richiede il calcolo del diagramma 2. Il server riceve la richiesta 3. Il server invia conferma di ricezione al client ed inizia il calcolo 4. Viene ritornato lo status della richiesta entro non oltre 30 secondi
Related U.C.	Caricamento FM
Exceptions	Feature Model troppo complesso per generazione diagramma
Post-conditions	Il server ha generato il diagramma MD

UC04	Modifica MDD
Actors	User
Description	Caso d'uso astratto rappresentante le funzionalità di modifica del diagramma MD fornite dall'applicativo
Pre-conditions	Il server ha creato l'MDD del FM
Steps	<ol style="list-style-type: none"> 1. Tasto mostra combinazioni feature invia la richiesta al server 2. L'informazione di ritorno dal server viene mostrata, anche in base a possibili stati d'errore
Related U.C.	Aggiunta Feature Aggiunta Constraint
Exceptions	
Post-conditions	Il diagramma MD sul server riflette le modifiche richieste

UC05	Aggiunta feature
Actors	User
Description	L'utente richiede l'aggiunta di una feature dato il nome
Pre-conditions	Il server ha calcolato l'MDD del FM
Steps	<ol style="list-style-type: none"> 1. L'utente richiede l'aggiunta di una feature 2. L'utente inserisce il nome della nuova feature nel prompt 3. La richiesta viene inviata al server che svolge le operazioni necessarie 4. Il server ritorna lo status della richiesta
Related U.C.	Modifica MDD
Exceptions	Nome della feature già utilizzato
Post-conditions	Il diagramma MD contiene la nuova feature senza vincoli

UC06	Aggiunta constraint
Actors	User
Description	L'utente richiede l'aggiunta di un constraint sotto forma di una regola
Pre-conditions	Il server ha calcolato l'MDD del FM Il vincolo è nel formato standard di cross-tree constraint
Steps	<ol style="list-style-type: none"> 1. L'utente richiede l'aggiunta di un constraint 2. L'utente inserisce il constraint nel prompt 3. La richiesta viene inviata al server che svolge le operazioni necessarie 4. Il server ritorna lo status della richiesta
Related U.C.	Modifica MDD
Exceptions	Constraint malformato
Post-conditions	Il diagramma MD contiene il nuovo vincolo

UC07	Richiesta Informazioni
Actors	User
Description	Caso d'uso astratto rappresentante le funzionalità di calcolo e ritorno di informazioni ottenute tramite MDD
Pre-conditions	Il server ha creato l'MDD del FM
Steps	
Related U.C.	Aggiunta Feature Aggiunta Constraint
Exceptions	
Post-conditions	Il client mostra le informazioni calcolate dal server

UC08	Calcolo Configurazioni Valide
Actors	User
Description	Calcola e ritorna il numero di configurazioni valide del modello in considerazione.
Pre-conditions	Il server ha creato l'MDD del FM
Steps	<ol style="list-style-type: none"> 1. Tasto mostra combinazioni feature invia la richiesta al server 2. L'informazione di ritorno dal server viene mostrata, anche in base a possibili stati d'errore
Related U.C.	Richiesta Informazioni
Exceptions	Diagramma troppo complesso per calcolo combinazioni
Post-conditions	L'applicazione web mostra a schermo le informazioni richieste

UC09	Calcolo ...
Actors	User
Description	
Pre-conditions	Il server ha creato l'MDD del FM
Steps	<ol style="list-style-type: none"> 1. ... 2. ...
Related U.C.	Calcolo MDD
Exceptions	
Post-conditions	L'applicazione web mostra a schermo le informazioni richieste

1.4 Requisiti Funzionali

1.4.1 Elenco Funzionalità

FU1	Caricamento Feature Model su server
Funz. Utente	Upload di un feature model al server

FU2	Creazione MDD
Funz. Utente	Calcolo del diagramma MD del feature model caricato precedentemente.

FU3	Aggiunta feature
Funz. Utente	Aggiornamento del MDD tramite l'aggiunta di una feature fornita dall'utente senza vincoli

FU4	Aggiunta constraint
Funz. Utente	Aggiornamento del MDD tramite il ricalcolo basato su un vincolo aggiuntivo fornito dall'utente

FU5	Calcolo configurazioni valide
Funz. Utente	Richiesta, calcolo e risposta contenente il numero di configurazioni valide sul MDD attualmente in considerazione.

1.5 Requisiti Non Funzionali

1.5.1 Affidabilità

AF01	Affidabilità – ore di funzionamento garantite **FUORI SCOPO PROGETTO
	<ul style="list-style-type: none">• n. massimo ore di non disponibilità/n.ore servizio (su base mensile): 12 ore/mese• ore totali di servizio: $30 \cdot 24 = 720$• ore totali di disponibilità del sistema: $720 - 12 = 708$• Affidabilità (%): $\text{ore totali di disponibilità del sistema} / \text{ore totali di servizio} \cdot 100 = 98.3$ n. massimo di ore consecutive di non disponibilità: n/a

1.5.2 Usabilità

US01	Caratteristica usabilità interfaccia utente
	Chiarezza nel workflow di utilizzo del client, display informazioni rilevanti di stati d'errore

1.5.3 Manutenibilità

MA02	Manutenibilità generale del software
	Commenti esplicativi sulle funzioni e documentazione

1.5.4 Portabilità

PO01	Implementazione interfaccia per permettere più client
	Disegnare l'interfaccia tra client e server in maniera tale da renderla compatibile con multiple tipologie di client.

1.5.5 Efficienza

PO01	Tempo di risposta
	Passaggio messaggi light

1.6 Architettura di Sistema

1.6.1 Proposta tecnologie

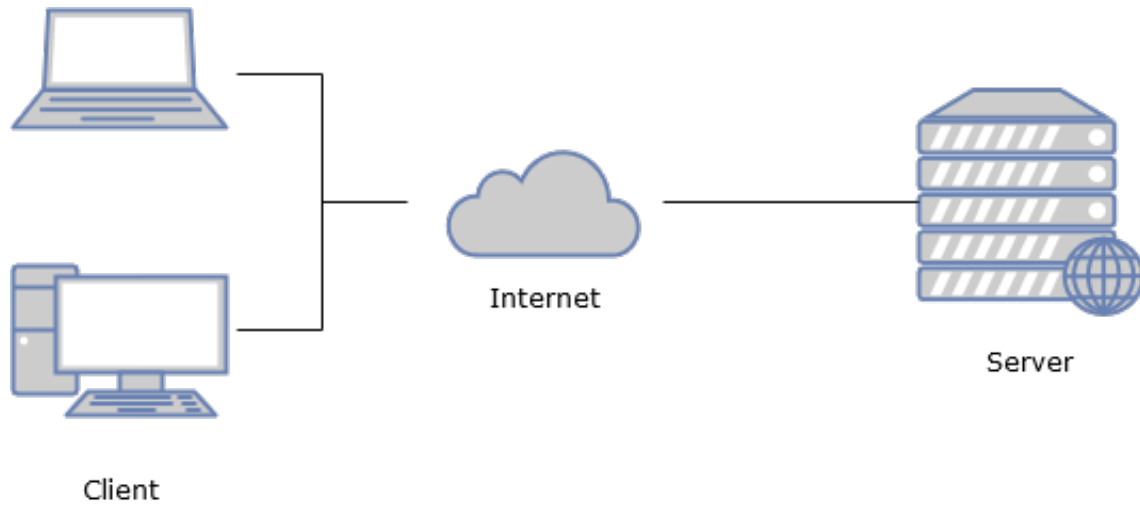
Il sistema verrà sviluppato seguendo i principi dell'architettura client-server a due livelli. A livello client saranno fornite all'utente finale le funzionalità richieste, permettendo lo scambio di informazioni con il server sul quale verranno eseguiti i calcoli computazionalmente complessi.

A livello server verrà sviluppato un componente applicativo in comunicazione con l'applicativo per la creazione di diagrammi MD tramite un'interfaccia. Suddividiamo quindi il progetto in due macro aree:

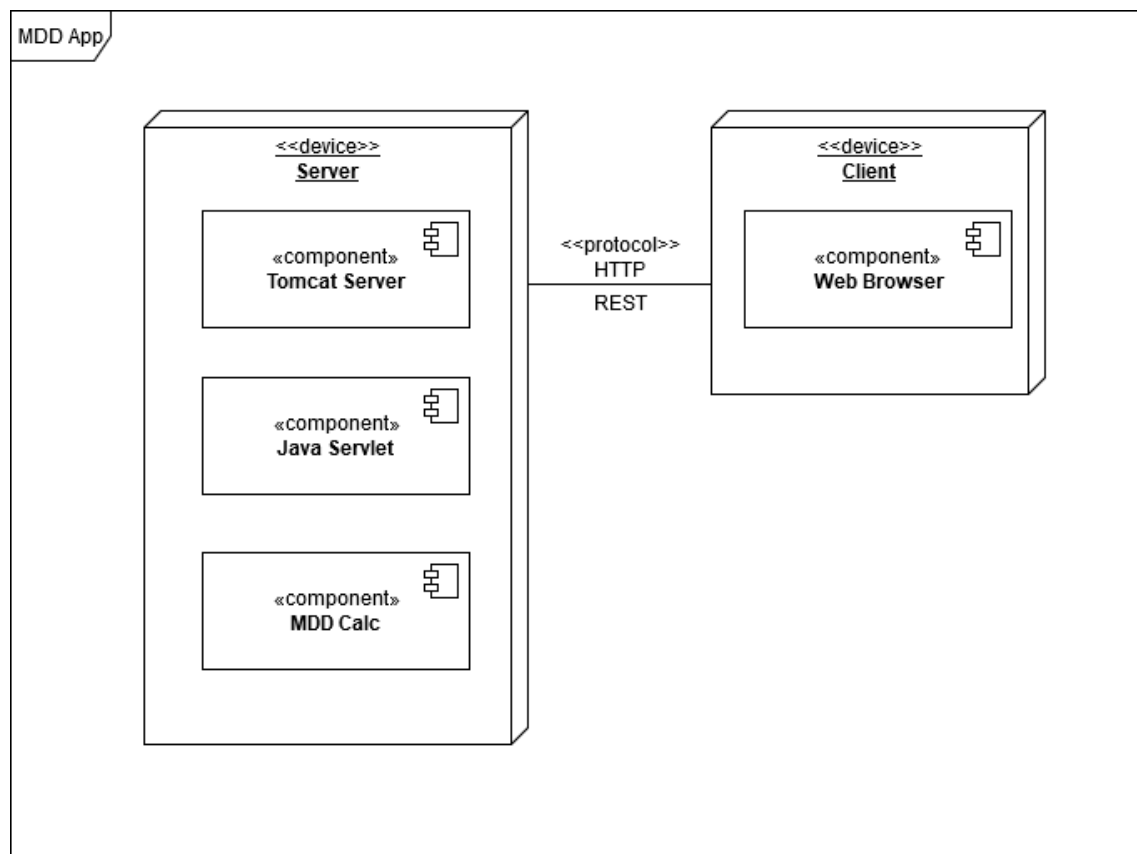
- **BackEnd**: gestione delle richieste, calcoli e algoritmi.
- **FrontEnd**: interfaccia grafica e gestione degli input-output per l'utente.

Lo scambio di messaggi tra front-end e back-end avverrà tramite messaggi REST per facilitare eventuali client di natura diversa da applicativo web/browser.

1.6.2 Topology Diagram



1.6.3 Deployment Diagram



Chapter 2

Iterazione 1 - Setup architettura client-server

2.1 Planning

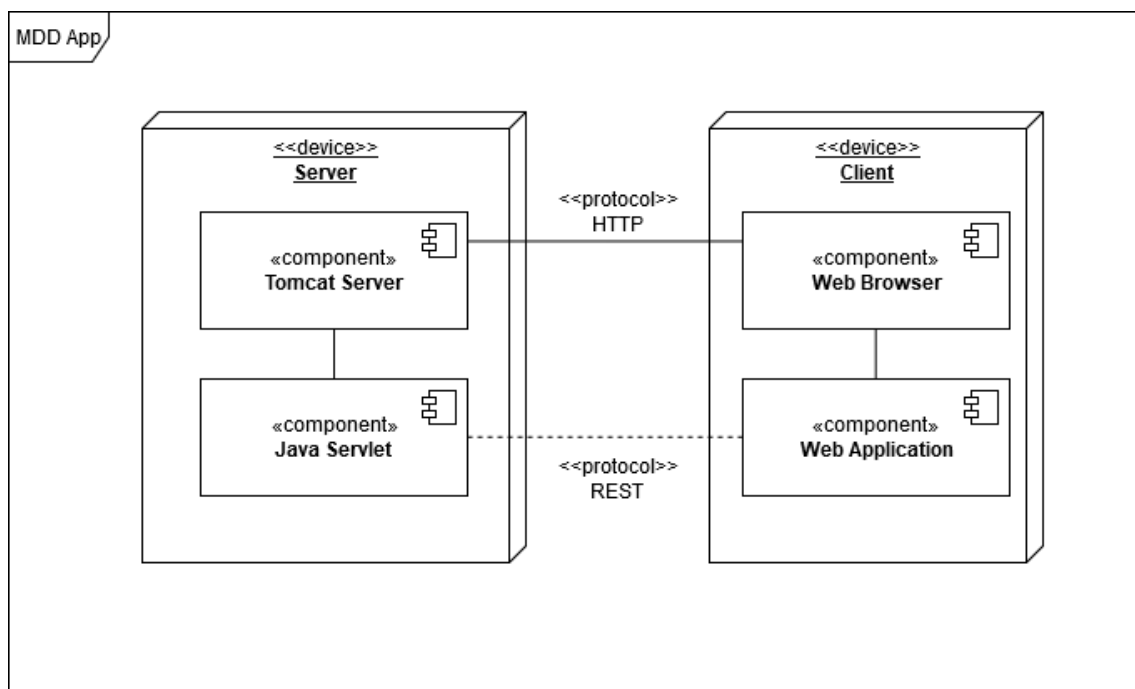
Obiettivo della prima iterazione è lo sviluppo dell'architettura base client-server sulla quale gli applicativi verranno eseguiti. L'architettura deve supportare l'applicativo client (sotto forma di web application) con lo scopo di fornire un'interfaccia d'accesso all'utilizzatore ed un applicativo server per la gestione delle richieste ricevute dai client e l'esecuzione del programma di conversione di MDD.

La creazione dell'architettura sottostante è a supporto di tutti i requisiti di progetto, per cui è stato deciso di svilupparla nella prima iterazione.

2.2 Design

Oggetto di design della prima iterazione è la componente architetturale client-server.

2.2.1 Design Diagram - Iterazione 1



2.3 Implementazione

Le tecnologie utilizzate per l'implementazione rimangono quelle scelte in fase di envisioning:

Per la gestione delle chiamate lato server, viene fatto uso di un server *Apache Tomcat*, con relativa pagina HTML, contenente la componente client del sistema, e file di configurazione *web.xml*, contenente i filtri e mapping necessari per il corretto smistamento dei messaggi in ingresso dai client.

Il supporto applicativo lato server è fornito da un servlet *HttpServlet* java facente parte del package Java opzionale *javax*. Il servlet fornisce le procedure necessarie per la gestione delle chiamate REST e conterrà l'implementazione dell'applicativo di conversione in MDD di *FeatureModel*.

Chapter 3

Iterazione 3 - Sviluppo classe MddInstance

3.1 Nota iterazione 2

L'iterazione 2 è stata omessa dal documento in quanto principalmente riguardante laboriosi tentativi di accoppiare le chiamate REST con servlet, far funzionare il passaggio di documenti xml tramite chiamata POST e progettazione html e javascript dell'applicazione web.

Di nota è l'implementazione della funzionalità FU1 per il caricamento di Feature Model al server.

3.2 Planning

La terza iterazione sarà concentrata sullo sviluppo della classe `MddInstance` che fungerà da interfaccia tra l'applicazione web e le classi per la conversione degli MDD da feature model.

L'obiettivo della classe è l'implementazione delle funzionalità FU2, FU4 ed FU5, mentre si appoggerà alle classi per la conversione di MDD per l'implementazione della funzionalità FU3.

In fase di analisi, lo sviluppo della funzionalità FU3 prevista si è rivelato non fattibile per via di limitazioni dell'implementazione della conversione e limitazioni della libreria per la creazione e gestione di MDD stessa.

Inoltre, si è presentata la possibilità di analizzare il Feature Model e restituire i dati ottenuti per fornire all'utilizzatore un ulteriore metodo per avere informazioni riguardanti la complessità e dimensioni del feature model, rappresentata da una nuova funzionalità FU6

I casi d'uso rilevanti allo sviluppo di questa iterazione sono quindi UC4 (e UC collegati), UC7 (e UC collegati) ed UC3 (illustrati nella sezione successiva a seguito di aggiornamento degli UC).

3.3 Design

3.3.1 Funzionalità

In base ad analisi più approfondite sulle possibili funzionalità da implementare, le funzionalità ed il diagramma dei casi d'uso sono stati aggiornati per riflettere le funzionalità aggiuntive da implementare e quelle non implementabili. I requisiti funzionali sono ora 6:

FU1	Caricamento Feature Model su server
Funz. Utente	Upload di un feature model al server

FU2	Creazione MDD
Funz. Utente	Calcolo del diagramma MD del feature model caricato precedentemente.

FU3	Aggiunta feature
Funz. Utente	Aggiornamento del MDD tramite l'aggiunta di una feature fornita dall'utente senza vincoli

FU4	Aggiunta constraint
Funz. Utente	Aggiornamento del MDD tramite il ricalcolo basato su un vincolo aggiuntivo fornito dall'utente

FU5	Calcolo configurazioni valide
Funz. Utente	Richiesta, calcolo e risposta contenente il numero di configurazioni valide sul MDD attualmente in considerazione.

FU6	Informazioni su Feature Model
Funz. Utente	Richiesta, calcolo e risposta di informazioni relative al feature model, quali numero di feature del modello e numero di constraint.

3.3.2 Casi d'uso

I casi d'uso sono stati aggiornati per riflettere i cambiamenti delle funzionalità del progetto. In particolare, viene rimosso il caso d'uso UC09, mentre il caso d'uso temporaneo UC09 viene espanso in 3 diversi casi d'uso legati ad UC07 così definiti:

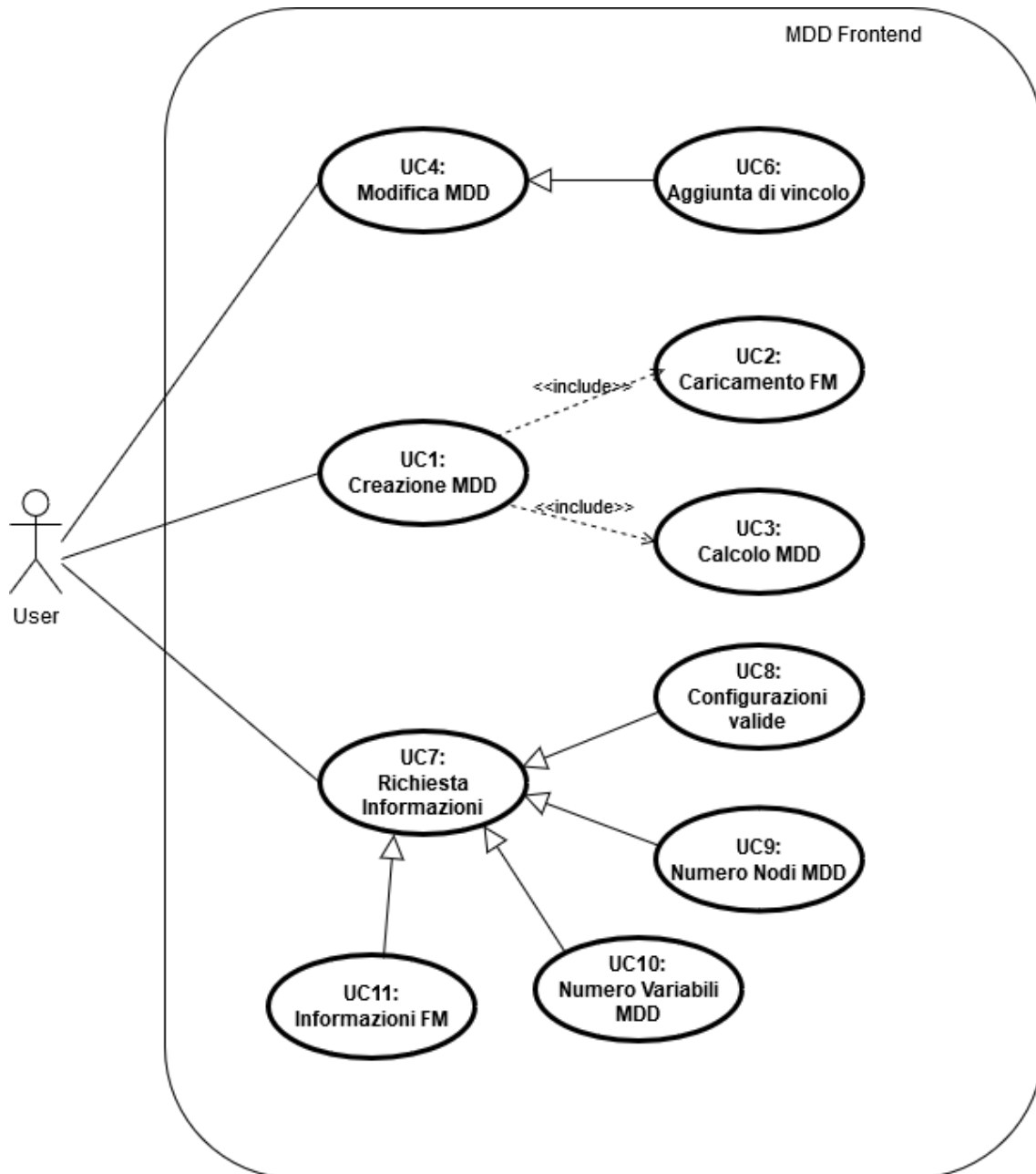
UC09	Numero Nodi MDD
Actors	User
Description	Calcola e ritorna il numero di nodi che compongono il diagramma MD.
Pre-conditions	Il server ha creato l'MDD del FM
Steps	<ol style="list-style-type: none">1. Tasto "mostra numero nodi" invia la richiesta al server2. L'informazione di ritorno dal server viene mostrata
Related U.C.	Richiesta Informazioni
Exceptions	Diagramma troppo complesso per calcolo combinazioni
Post-conditions	L'applicazione web mostra a schermo le informazioni richieste

UC10	Numero Variabili MDD
Actors	User
Description	Calcola e ritorna il numero di variabili che compongono il diagramma MD.
Pre-conditions	Il server ha creato l'MDD del FM
Steps	<ol style="list-style-type: none">1. Tasto "mostra numero variabili" invia la richiesta al server2. L'informazione di ritorno dal server viene mostrata
Related U.C.	Richiesta Informazioni
Exceptions	Diagramma troppo complesso per calcolo combinazioni
Post-conditions	L'applicazione web mostra a schermo le informazioni richieste

UC11	Informazioni FM
Actors	User
Description	Calcola e ritorna il numero di feature e/o constraint del FM caricato.
Pre-conditions	Feature Model ricevuto dal server
Steps	<ol style="list-style-type: none"> 1. Tasti "mostra numero feature" e "mostra numero constraint" invia le richieste al server 2. L'informazione di ritorno dal server viene mostrata
Related U.C.	Richiesta Informazioni
Exceptions	Feature Model malformato
Post-conditions	L'applicazione web mostra a schermo le informazioni richieste

Il diagramma dei casi d'uso riflette i cambiamenti come segue:

Use Case Diagram - Iterazione 3



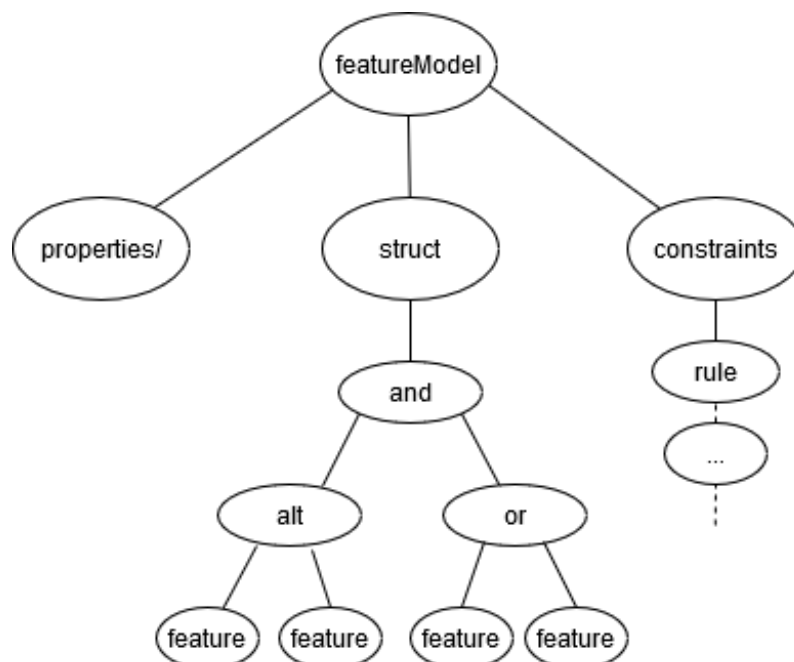
3.3.3 Algoritmo di ricerca informazioni Feature Model

Un feature model è un documento xml e come tale, è una struttura gerarchica rappresentabile come un albero di elementi. E' composto da un nodo radice <featureModel> il quale ha 3 figli:

1. <properties/> non contenente informazioni rilevanti
2. <struct> contenente la struttura di feature legate da constraint inline, composta da elementi
 - (a) <and> constraint inline di tipo AND (Nodo interno)
 - (b) <alt> constraint inline di tipo ALT (Nodo interno)
 - (c) <or> constraint inline di tipo OR (Nodo interno)
 - (d) <feature> feature (Nodo foglia)
3. <constraints> avente come figli tutti i constraint cross-tree (elementi <rule>)

A scopo di esempio, il FeatureModel d'esempio *gplTiny* può essere rappresentato come:

Albero XML gplTiny



Pseudocodice

L'algoritmo per l'ottenimento delle informazioni dovrà dunque eseguire una esplorazione depth-first pre-ordine completa sui sottoalberi *struct* e *constraints*. Pseudocodice algoritmo di ricerca ricorsivo:

```
GetInformation(lista nodo iniziale NL) -> numero figli d'interesse N
  N <- 0
  foreach (child in NL)
    if(child in Casi d'interesse)
      increase N
    if(child has ChildNodes)
      N <- N + GetInformation(ChildNodes di child)

  return N
```

Complessità algoritmo

Definiamo **F** come il numero di nodi completo del feature model, **S** il numero di nodi del sotto albero di *struct* e **C** il numero di nodi del sotto albero di *constraints*. Essendo l'algoritmo una esplorazione in profondità completa, ciascun nodo dell'albero XML sarà visitato una ed una sola volta.

Supponendo il costo di accesso ad un elemento dell'albero sia $O(1)$, la complessità dell'algoritmo applicato all'intero albero XML sarà dunque pari ad $O(F)$.

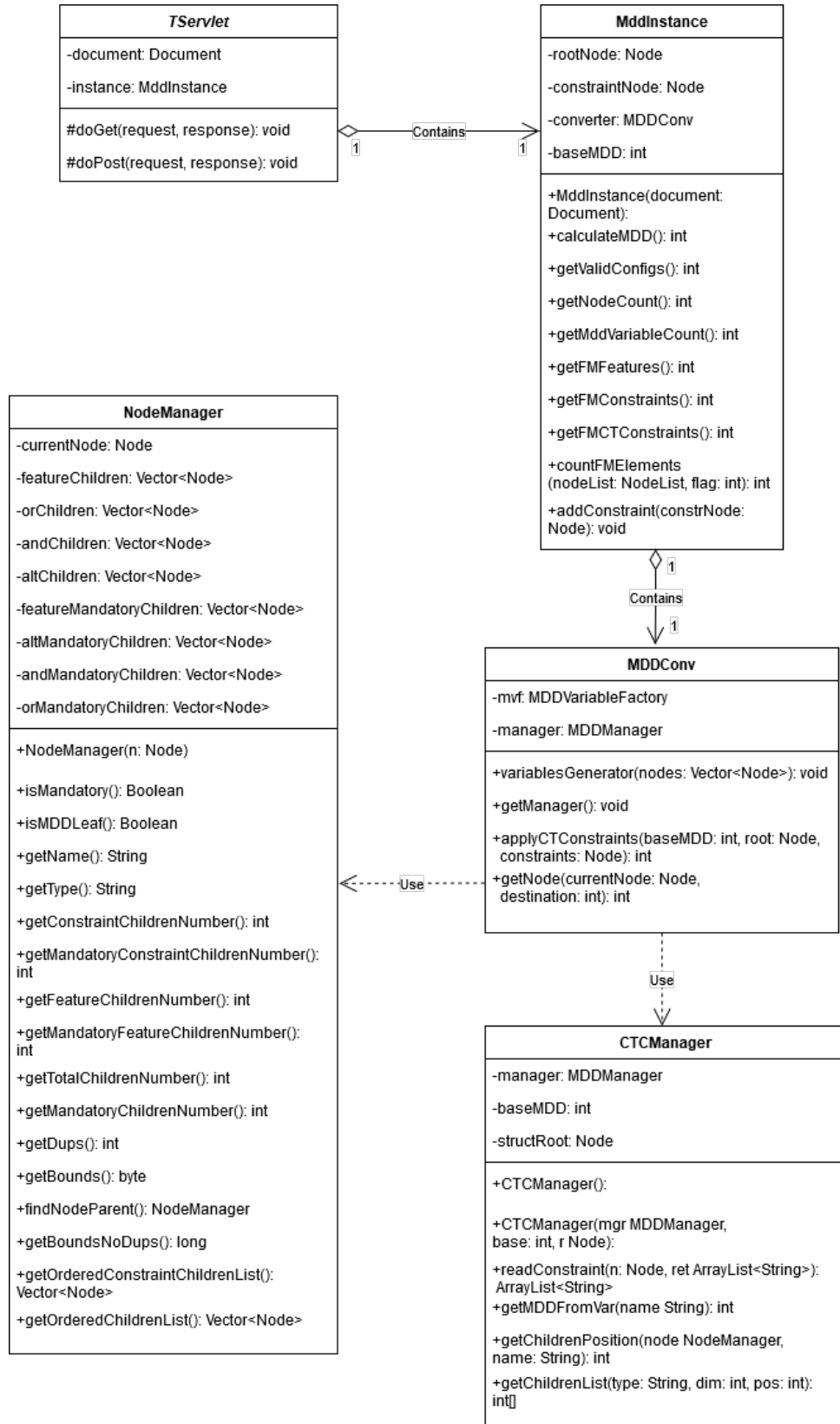
La complessità può però essere ridotta, separando il calcolo per la ricerca di informazioni legate ai constraint inline da quello dei constraint cross-tree, siccome questi abitano in due sottoalberi completamente distinti.

Separando le due ricerche, la complessità scende a $O(S)$ per il calcolo di constraint e feature ed a $O(C)$ per il calcolo di constraint cross-tree.

3.4 Diagramma delle classi

Introduciamo un diagramma delle classi per modellizzare la struttura di classi che comporrà la struttura backend del progetto:

3.4.1 Class Diagram



3.5 Implementazione

L'algoritmo viene implementato dalla funzione *countFMInfo()*, la quale implementa una soluzione per la ricerca di tutti gli elementi richiesti, differenziandoli tramite l'utilizzo di un parametro flag.

I flag differenziano tra le tre richieste, associando al valore 1 la conta dei constraint inline, al valore 2 la delle feature ed al valore 3 la conta dei constraint cross-tree

La differenziazione delle diverse ricerche viene effettuata da 3 distinte funzioni le quali ritornano il valore della funzione sviluppata in giunzione con flag e nodo di riferimento corretti.

Assieme all'algoritmo, sono state implementate numerose funzioni e procedure che sfruttano la libreria di conversione di MDD per il soddisfacimento delle funzionalità richieste, quali calcolo del diagramma MD, conta delle configurazioni valide, conta dei nodi e delle variabili che compongono il diagramma.

3.5.1 Analisi Dinamica

Per assicurarsi che l'algoritmo e le funzioni che utilizzano il convertitore di MDD funzionino, è stata implementata una classe di test tramite la libreria JUnit per unit testing per il linguaggio Java.

Questa classe fa uso di tre diversi FeatureModel di dimensioni contenute, permettendo di confrontare i valori ritornati dall'algoritmo con i valori attesi. Similarmente, i risultati del calcolo del diagramma MD su questi feature model vengono confrontati coi risultati ottenuti individualmente in calcoli separati.

I casi di test sono dunque raggruppati a gruppi di 3 con ciascuno di questi gruppi incentrato sul testing di una singola funzione/procedura implementata dalla classe.

Inizializzazione casi test

La classe di test viene inizializzata nel costruttore con la creazione di tre istanze della classe *MddInstance*.

mddInst1, 2 e 3 conterranno istanze rispettivamente degli FM *gplTiny*, *gplSmallTest* e *gameOfLife* (*gol*). La classe, in seguito all'inizializzazione, rappresenterà lo stato dell'applicativo in seguito all'invio del feature model al server.

Costruttore MddInstanceTest

```
public MddInstanceTest() throws SAXException, IOException,
    ↳ ParserConfigurationException {
    final DocumentBuilder documentBuilder = DocumentBuilderFactory.
        ↳ newInstance().newDocumentBuilder();
    Document document;
    // Modello gplTiny
    document = documentBuilder.parse("src/main/webapp/WEB-INF/
        ↳ featureModels/gplTinyModel.xml");
    mddInst1 = new MddInstance(document);
    // Modello gplSmallTest
    document = documentBuilder.parse("src/main/webapp/WEB-INF/
        ↳ featureModels/gplSmallTestModel.xml");
    mddInst2 = new MddInstance(document);
    // Modello Game of Life
    document = documentBuilder.parse("src/main/webapp/WEB-INF/
        ↳ featureModels/golModel.xml");
    mddInst3 = new MddInstance(document);
}
```

Casi di test

Qui inclusi alcuni casi di test di rivelanza.

1. Caso di test per la conta di feature nel progetto

```
@Test
public void featureCountTest2 () {
    // gplSmallTest feat# = 19
    assertEquals(mddInst2.getFMFeatures(), 19);
}
```

2. Caso di test per il calcolo del MDD

```
@Test
public void MDDTest3 () {
    // golModel
    assertTrue(mddInst3.calculateMDD() > 0);
}
```

3. Caso di test per il calcolo delle configurazioni valide

```
@Test
public void validConfigsTest1 () {
    // gplTiny
    mddInst1.calculateMDD();
    assertTrue(mddInst1.getValidConfigs() == 6);
}
```

4. Caso di test per l'aggiunta di constraint (possiamo vedere come l'aggiunta di un constraint abbia ridotto il numero di configurazioni valide)

```
@Test
    @SuppressWarnings("PMD.LocalVariableCouldBeFinal")
    public void addConstraint1 () throws
        ↳ ParserConfigurationException, SAXException, IOException {
        // gplSmallTest
        mddInst1.calculateMDD();
        int oldVC = mddInst1.getValidConfigs();
        String constraintNode = "<constraints><rule><var>Cycle</var></
            ↳ rule></constraints>";
        DocumentBuilderFactory factory = DocumentBuilderFactory.
            ↳ newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        InputSource is = new InputSource(new StringReader(
            ↳ constraintNode));
        Document tempDoc = builder.parse(is);
        Node constrNode = tempDoc.getElementsByTagName("constraints")
            ↳ .item(0);

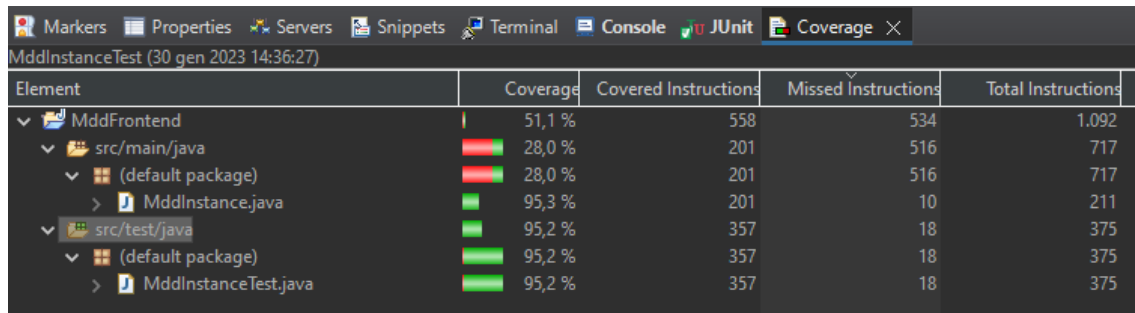
        mddInst1.addConstraint(constrNode);

        assertTrue(mddInst1.getValidConfigs() == 2);
    }
```

3.5.2 Copertura del codice

La copertura risultante dall'esecuzione dei casi di test è calcolata tramite la libreria di calcolo della copertura JaCoCo. L'esecuzione dei casi di test risulta in una copertura del 95% sulle istruzioni della classe *MddInstance*.

Coverage



The screenshot shows the Coverage view in an IDE. The title bar indicates 'MddInstanceTest (30 gen 2023 14:36:27)'. The table below displays the coverage data for various elements.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
MddFrontend	51,1 %	558	534	1.092
src/main/java	28,0 %	201	516	717
(default package)	28,0 %	201	516	717
MddInstance.java	95,3 %	201	10	211
src/test/java	95,2 %	357	18	375
(default package)	95,2 %	357	18	375
MddInstanceTest.java	95,2 %	357	18	375

3.5.3 Analisi Statica

Per l'analisi statica viene fatto uso della libreria PMD per eclipse, la quale mostra warnings riguardanti il codice direttamente dall'IDE, secondo la base delle regole selezionate.

E' stato fatto uso di un set di regole standard chiamato *All-Java* contenente tutte le regole di bontà del codice del linguaggio.

Inoltre è stato fatto uso dell'applicativo standalone STAN, di seguito riportate informazioni contenute nel report generato:

Analisi Stan4 - Report

Metrics Summary

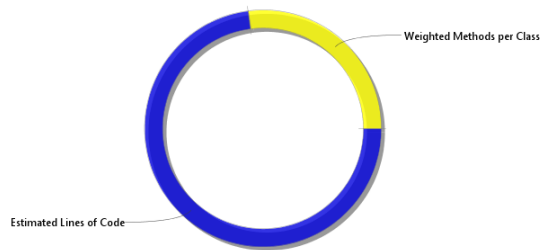
Metric	Value
Number of Libraries	1
Number of Packages	1
Number of Top Level Classes	5
Average Number of Top Level Classes per Package	5
Average Number of Member Classes per Class	0
Average Number of Methods per Class	8.20
Average Number of Fields per Class	4.40
Estimated Lines of Code	1235
Estimated Lines of Code per Top Level Class	247
Average Cyclomatic Complexity	6.95
Fat for Library Dependencies	0
Fat for Flat Package Dependencies	0
Fat for Top Level Class Dependencies	5
Tangled for Library Dependencies	0%
Average Component Dependency between Libraries	0%
Average Component Dependency between Packages	0%
Average Component Dependency between Units	25%
Average Distance	0
Average Absolute Distance	0
Average Weighted Methods per Class	57
Average Depth of Inheritance Tree	1
Average Number of Children	0
Average Coupling between Objects	n/a
Average Response for a Class	n/a
Average Lack of Cohesion in Methods	n/a

Top Violations (2 of 2)

Artifact	Metric	Value
MDDConv	ELOC	476
MDDConv	WMC	102

Pollution Chart

Pollution 0.52



Metric Ratings

Count Metrics

Metric	Rating	Linear
Number of Top Level Classes	... 20 40 60 80 ...	true
Number of Methods	... 25 50 100 200 ...	true
Number of Fields	... 10 20 40 80 ...	true
Estimated Lines of Code	... 200 300 400 500 ...	true
Estimated Lines of Code	... 30 60 120 240 ...	true

Chidamber & Kemerer Metrics

Metric	Rating	Linear
Weighted Methods per Class	... 0 100 200 300 ...	true
Depth of Inheritance Tree	... 4 6 8 10 ...	true
Average Depth of Inheritance Tree	... 2 1 0 ...	true
Coupling between Objects	... 0 25 250 ...	true
Response for a Class	... 0 100 1000 ...	true

stan4.com

Complexity Metrics

Metric	Rating	Linear
Cyclomatic Complexity	... 10 15 20 30 ...	true
Fat	... 30 60 120 240 ...	true
Fat	... 30 60 120 240 ...	true
Fat	... 30 60 120 240 ...	true
Tangled	... 0 1 ...	true
Tangled for Library Dependencies	... 0 1 ...	true
Average Component Dependency between Libraries	... 0 .5 1 ...	true
Average Component Dependency between Packages	... 0 .5 1 ...	true

Robert C. Martin Metrics

Metric	Rating	Linear
Distance	... -1 -.5 0 .5 1 ...	true
Average Absolute Distance	... 0 .4 .5 1 ...	true

Chapter 4

Iterazione 4 - Implementazione risposte REST per eccezioni

4.1 Planning

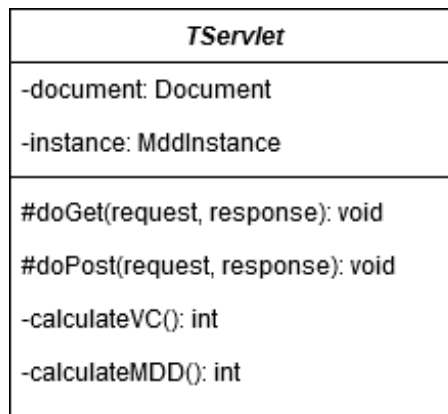
Obiettivo dell'ultima iterazione è l'implementazione delle eccezioni riguardanti i casi d'uso UC03 ed UC08.

Allo stato attuale, il client non riceve informazioni sul completamento o meno della conversione del MDD, informazione fondamentale vista la suscettibilità del sistema a non completare il calcolo in tempi sufficientemente brevi per feature model complessi. Si prospetta dunque di implementare un computatore asincrono interrompibile dopo un certo periodo di tempo voluto (scelto arbitrariamente a 30 secondi)

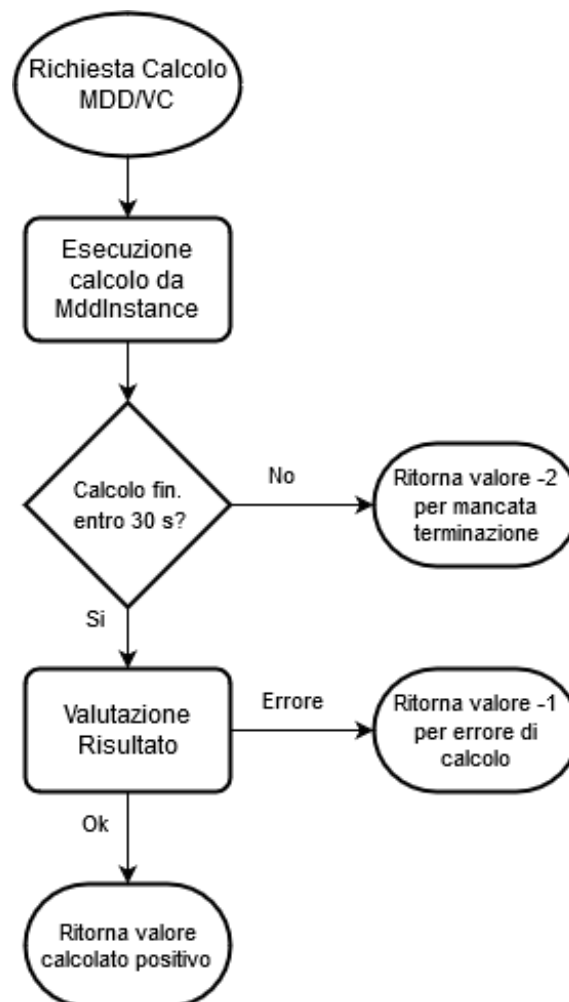
4.2 Design

La funzionalità verrà introdotta tramite 2 funzioni aggiuntive nella classe servlet. Il funzionamento delle due classi è funzionalmente identico, con l'esecuzione del proprio compito delegata ad un esecutore asincrono governato da un timer. Alla scadenza del tempo prefissato, l'esecuzione viene terminata ed il client viene informato del risultato.

Classe Servlet Aggiornata



FlowChart delle chiamate interessate



Infine sul client viene implementato un elemento HTML fornito da Materialize per mostrare visivamente come il server sia in elaborazione, a scopo informativo del client

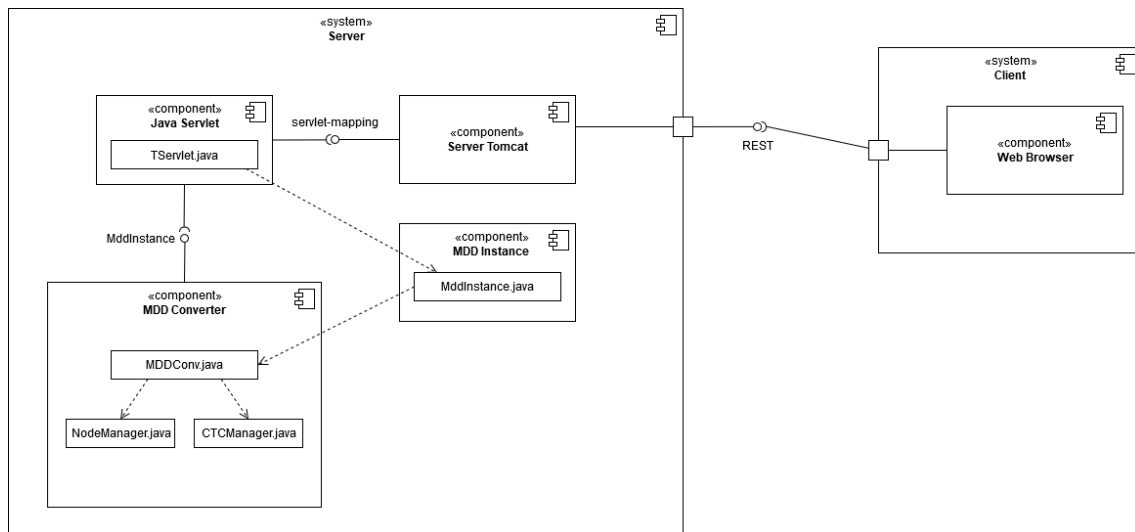
Loading Bar



4.2.1 Diagramma dei componenti

Il diagramma dei componenti finito include i due sistemi principali, client e server, contenenti i componenti che formano il sistema ed i collegamenti esistenti tra essi

Component Diagram



Chapter 5

Bibliografia

5.1 Manuale Utente

5.1.1 Server Setup

Per il funzionamento dell'applicativo bisogna impostare il server tomcat con la applicazione web e la libreria WAR dell'applicativo server sviluppati.

Il server correttamente configurato conterrà la pagina web index.html, il documento di configurazione web.xml e la libreria WAR collegata, come segue:

Server Config

esto PC > Volume (D:) > Leo > Progetto Info 3 > Tomcat 9.0 > webapps > ROOT				
Nome	Ultima modifica	Tipo	Dimensione	
css	28/01/2023 17:16	Cartella di file		
js	28/01/2023 17:16	Cartella di file		
META-INF	28/01/2023 17:16	Cartella di file		
WEB-INF	28/01/2023 17:16	Cartella di file		
index.html	30/01/2023 15:46	Firefox HTML Doc...	153 KB	
LICENSE	09/09/2018 22:21	File	2 KB	
MddFrontend.war	30/01/2023 16:21	File WAR	591 KB	
README.md	09/09/2018 22:21	File MD	5 KB	

Nome

- featureModels
- lib
- web.xml

5.1.2 Utilizzo client

I passi da seguire per l'utilizzatore del servizio tramite client web sono i seguenti:

1. Caricamento del Feature Model al server. Questo passo è necessario per l'utilizzo di tutti gli altri componenti del client.
 - (a) Selezione del Feature Model tramite esplora risorse
 - (b) Invio del feature model tramite tasto

Caricato il Feature Model, verrà reso disponibile al client l'utilizzo delle funzionalità di raccolta informazioni dal feature model stesso e il calcolo del MDD.

2. (Opzionale) Raccolta informazioni del feature model
3. Calcolo del diagramma MD relativo al Feature Model caricato sul server. Terminato il calcolo del diagramma, verranno rese disponibili al client le funzionalità di raccolta informazioni del MDD e della modifica dello stesso tramite aggiunta di constraint.
4. (Opzionale) Ottenere informazioni da MDD, ovvero il numero di configurazioni valide, il numero di nodi che costituiscono il diagramma o il numero di variabili.
5. (Opzionale) Aggiungere constraint al modello, inserimento di un constraint sotto forma di `<rule>.....</rule>`. Una volta inviato al server, il constraint viene elaborato ed applicato direttamente al diagramma attualmente istanziato.

Calcoli seguenti di configurazioni valide o numero di nodi rifletteranno i cambiamenti.

Step 1 - Selezione Feature Model

MDD Frontend

Esecuzione calcolo MDD server-side

Carica Feature Model XML al server

Seleziona il feature model da inviare al server per calcolo MDD e informazioni connesse.

Select a file: Nessun file selezionato.

Calcola MDD del modello

Effettua il calcolo del MDD del modello. Potrebbe potenzialmente bloccarsi se calcolo troppo complesso.

Informazioni dal Feature model

Ottiene informazioni dal feature model, senza bisogno di calcolare l'MDD.

Ottieni informazioni dal MDD

Informazioni ottenute dal MDD, principalmente il numero di configurazioni valide, e il numero di nodi e variabili del diagramma.

Aggiungi constraint al modello

Inserisci il constraint nel formato cross-tree, rinchiodendolo tra tag "rule"

Constraint

Step 2/3 - Informazioni FM e calcolo MDD

MDD Frontend

Esecuzione calcolo MDD server-side

Carica Feature Model XML al server

Seleziona il feature model da inviare al server per calcolo MDD e informazioni connesse.

Select a file: gplModel.xml

Calcola MDD del modello

Effettua il calcolo del MDD del modello. Potrebbe potenzialmente bloccarsi se calcolo troppo complesso.

Informazioni dal Feature model

Ottiene informazioni dal feature model, senza bisogno di calcolare l'MDD.

Ottieni informazioni dal MDD

Informazioni ottenute dal MDD, principalmente il numero di configurazioni valide, e il numero di nodi e variabili del diagramma.

Aggiungi constraint al modello

Inserisci il constraint nel formato cross-tree, rinchiodendolo tra tag "rule"

Constraint

Step 3 - Status chiamata in attesa di risposta

MDD Frontend

Esecuzione calcolo MDD server-side

Carica Feature Model XML al server
Seleziona il feature model da inviare al server per calcolo MDD e informazioni connesse.

Select a file: eshopModel.xml

Calcola MDD del modello
Effettua il calcolo del MDD del modello. Potrebbe potenzialmente bloccarsi se calcolo troppo complesso.

Informazioni dal Feature model
Ottiene informazioni dal feature model, senza bisogno di calcolare l'MDD.

Ottieni informazioni dal MDD
Informazioni ottenute dal MDD, principalmente il numero di configurazioni valide, e il numero di nodi e variabili del diagramma.

Aggiungi constraint al modello
Inserisci il constraint nel formato cross-tree, rinchiudendolo tra tag "rule"

Constraint

Step 4/5 - Informazioni MDD e aggiunta constraint

MDD Frontend

Esecuzione calcolo MDD server-side

Carica Feature Model XML al server
Seleziona il feature model da inviare al server per calcolo MDD e informazioni connesse.

Select a file: gplModel.xml

Calcola MDD del modello
Effettua il calcolo del MDD del modello. Potrebbe potenzialmente bloccarsi se calcolo troppo complesso.

Informazioni dal Feature model
Ottiene informazioni dal feature model, senza bisogno di calcolare l'MDD.

Ottieni informazioni dal MDD
Informazioni ottenute dal MDD, principalmente il numero di configurazioni valide, e il numero di nodi e variabili del diagramma.

Aggiungi constraint al modello
Inserisci il constraint nel formato cross-tree, rinchiudendolo tra tag "rule"

5.2 Toolchain

Elenco degli strumenti utilizzati:

- **GitHub:** Repository Versioning
- **Eclipse:** IDE per lo sviluppo java
- **Apache Tomcat:** Server virtuale
- **Catalina CorsFilter:** Meccanismo gestione richieste cross-origin
- **Java Servlet:** Esecuzione applicativo su server
- **JST Server Adapters:** Server Tools framework e adattatori per Apache Tomcat
- **Java:** Linguaggio di programmazione principale
- **Javascript:** Linguaggio di programmazione per client
- **Ajax:** Meccanismo di gestione scambio messaggi RESTful lato client
- **MDDLib:** Libreria di Colomoto per creazione MDD su java
- **w3c.dom:** Libreria per la gestione di documenti XML
- **Overleaf:** Documentazione Latex per il progetto
- **Draw.io:** Design diagrammi di progetto
- **Notepad++:** Editor testo slim
- **Mozilla Firefox:** Web Browser/Client
- **Maven:** Framework automazione build
- **HTML:** Linguaggio ipertestuale per sviluppo pagine web
- **Materialize CSS:** Libreria fogli di stile per HTML
- **JUnit:** Tool per analisi dinamica del codice
- **Eclipse PMD:** Tool per analisi del codice statica eclipse
- **Stan4J:** Tool per analisi statica delle classi