Università degli Studi di Salerno

Dipartimento di Informatica

A.A. 2021/22

cASpER

Supporto per nuove versioni di IntelliJ IDEA e aggiunta strategia di Extract Class Refactoring basata sulla Teoria dei Giochi

Analisi, Progettazione e Realizzazione delle modifiche

Marco Calenda
0522501165

prof. **Andrea De Lucia**
prof. **Dario Di Nucci**

tutor **Manuel De Stefano**

Indice

Introduzione

cASpER [1] è un tool in grado di ispezionare codice sorgente Java allo scopo di rilevare Code Smells e fornire una strategia di refactoring appropriata.

Il seguente documento è un report delle attività di manutenzione proposte e, per ciascuna, è esposta una Change Request (CR) ed eseguita una Impact Analysis al fine di verificarne la fattibilità. In seguito, sono documentate la progettazione e la realizzazione delle modifiche. Infine, sarà prodotto un documento incentrato sul testing del sistema prima della modifica e sull'analisi e progettazione dei nuovi casi di test relativi alle modifiche introdotte.

La prima attività proposta è una manutenzione adattiva: cASpER è implementato come un plugin dell'IDE IntelliJ IDEA ed utilizza Gradle 4.4 e Java 1.8 come Build Automation, garantendo al tool una compatibilità al più alla versione 2020.3 dell'IDE. È stato scelto quindi di aggiornare Gradle alla versione 7.4.2 e di utilizzare Java 11. Inoltre, anche Gradle IntelliJ Plugin, responsabile del building, testing, verifica e pubblicazione dei plugin per IDEs basati su IntelliJ, sarà aggiornato dalla versione 0.4.9 alla versione 1.6.0. In questo modo, cASpER sarà compatibile con le versioni dell'IDE successive alla 2020.3 ed, in particolare, sarà prodotta la build per la versione 2022.1.

La seconda attività di manutenzione prevede l'aggiunta di una nuova strategia di Extract Class Refactoring (ECR), proposto da Bavota *et. al.* [2], che riguarda il refactoring del Code Smell "Blob". L'algoritmo attuale è basato sulla Teoria dei Grafi [3] e cerca di massimizzare la coesione delle classi estratte; lo scopo del nuovo approccio è invece di trovare un compromesso tra coesione ed accoppiamento (effetto collaterale dell'ECR), basandosi sulla Teoria dei Giochi e modellando il problema ECR come un gioco nel quale a turno i giocatori (*i.e.* le classi da estrarre) sceglieranno l'equilibrio di Nash [4] calcolato sulla matrice contenente tuple di payoffs (il guadagno per ciascun giocatore). Il numero dei giocatori che terranno parte al gioco (e per ciascuno un metodo seme da cui partire) è

scelto utilizzando un'euristica basata su Latent Dirichlet Allocation (LDA), un Topic Model.

Maggiori informazioni sull'implementazione ed i pro e contro del nuovo approccio sono descritti nel documento prodotto per l'esame di Software Dependability nella repository Github [5].

Change Requests

In questa sezione sono esposte le Change Request (CR) ed ulteriori informazioni relative alle motivazioni dei due interventi di manutenzione proposti.

CR_01: supporto alle nuove versioni di IntelliJ IDEA

A partire dalla versione 2020.3 è necessario Java 11 per poter avviare IntelliJ IDEA [6]. Inoltre, considerata la matrice di compatibilità Java & Gradle [7], è opportuno aggiornare Gradle per poter supportare la nuova versione di Java. Infine sarà aggiornato Gradle IntelliJ Plugin che, raggiunta la versione 1.x, ha introdotto significativi miglioramenti e funzionalità.

<table>
<tr><th colspan="2">ID</th><th colspan="2">Richiedente</th></tr>
<tr><td colspan="2">CR_01</td><td colspan="2">Andrea De Lucia</td></tr>
<tr><th>Data</th><th colspan="2">Stato</th><th>Progetto</th></tr>
<tr><td>25 Aprile 2022</td><td colspan="2">Approvata</td><td>cASpER</td></tr>
<tr><th colspan="4">Descrizione</th></tr>
<tr><td colspan="4">Upgrade delle seguenti componenti:
Gradle v4.4 —> v7.4.2
Java 1.8 —> 11
Gradle IntelliJ Plugin v0.4.9 —> v1.6.0</td></tr>
<tr><th colspan="4">Stima tempi</th></tr>
<tr><td>Fase</td><td>Tempo Stimato</td><td>Tempo Reale</td><td>Data Completamento</td></tr>
<tr><td>Analisi e Progettazione</td><td>30 ore</td><td>40 ore</td><td>10 Maggio</td></tr>
<tr><td>Implementazione</td><td>10 ore</td><td>10 ore</td><td>12 Maggio</td></tr>
<tr><td>Testing</td><td>10 ore</td><td>5 ore</td><td>15 Maggio</td></tr>
<tr><td>TOTALE</td><td>50 ore</td><td>60 ore</td><td></td></tr>
<tr><th colspan="2">Motivazione</th><th colspan="2">Conseguenze, se non accettata</th></tr>
<tr><td colspan="2">Rendere compatibile cASpER alle versioni di IntelliJ IDEA successive alla 2020.3</td><td colspan="2">cASpER non potrà essere installato su versioni di IntelliJ IDEA successive alla 2020.3</td></tr>
</table>

CR_02: strategia ECR basata sulla Teoria dei Giochi

ID		Richiedente	
CR_02		Andrea De Lucia	
Data	**Stato**		**Progetto**
25 Aprile 2022	Chiusa - Completata		cASpER
Descrizione			
Aggiunta di un nuovo algoritmo di Extract Class Refactoring (ECR), basato sulla Teoria dei Giochi, proposto da Bavota *et. al.* [2].			
Stima tempi			
Fase	Tempo Stimato	Tempo Reale	Data Completamento
Analisi e Progettazione	30 ore	30 ore	20 Maggio
Implementazione	50 ore	70 ore	8 Giugno
Testing	40 ore	50 ore	2 Luglio
TOTALE	120 ore	150 ore	
Motivazione		**Conseguenze, se non accettata**	
Implementare un approccio che punta a trovare un compromesso tra l'aumento di coesione (effetto desiderato) e l'aumento di accoppiamento (effetto collaterale) quando si applica un algoritmo di ECR.		Nessuna.	

Reverse Engineering

Non essendo fornito di una documentazione, è stato necessario condurre una fase di Reverse Engineering al fine di recuperare il design del tool identificando i vari componenti e le relazioni fra questi. In particolare, è utilizzata una rappresentazione UML basata su Class Diagram e Sequence Diagram che garantiscono il giusto livello di dettagli per comprendere le relazioni tra le componenti ed il workflow di specifiche azioni interessate dalle CR. In Figura 1 è mostrata la gerarchia dei packages di cASpER, ottenuta attraverso il tool "StarUML".

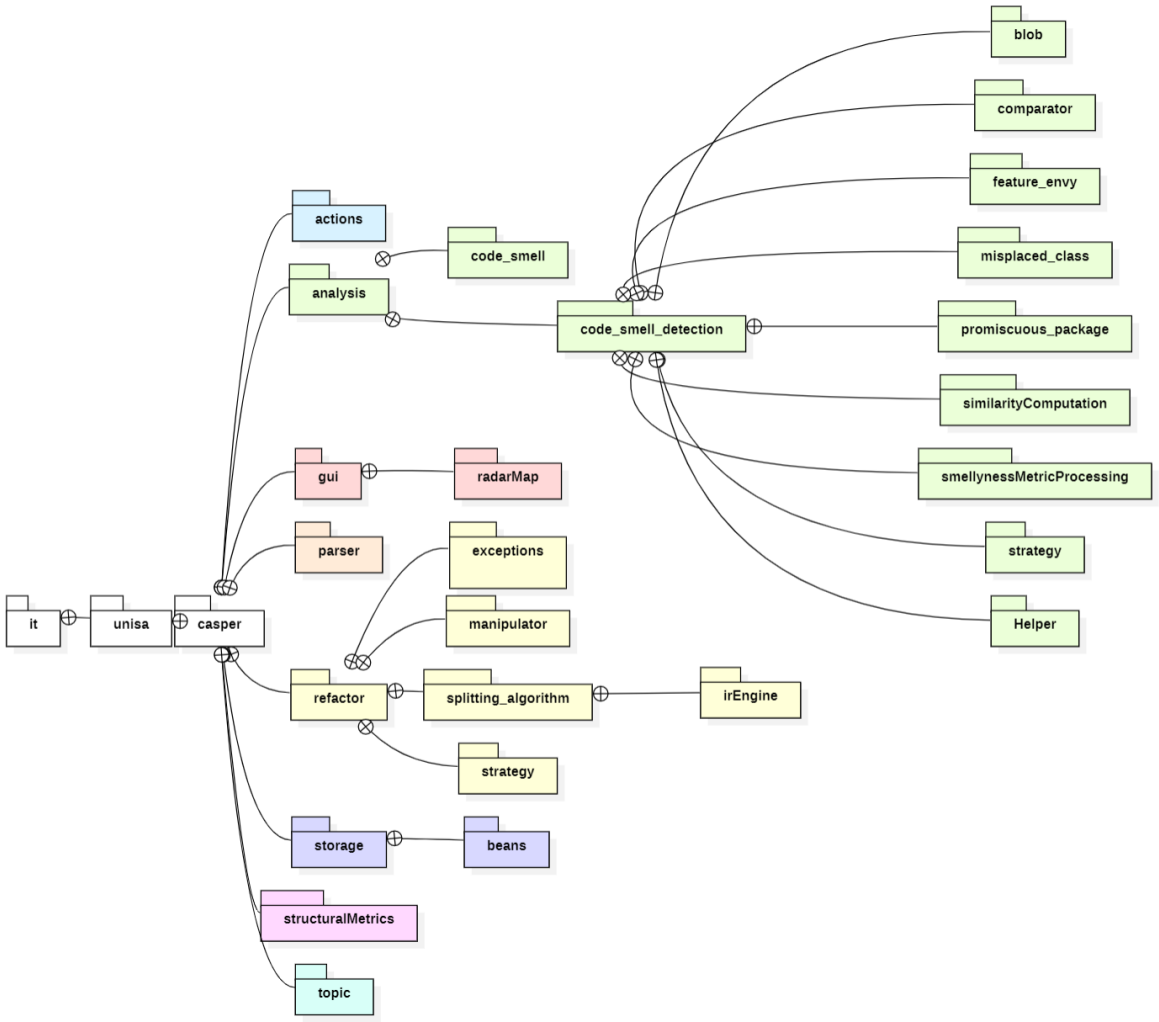

Figura 1: la gerarchia dei packages di cASpER

In seguito, sono stati ispezionati i singoli packages allo scopo di ottenere informazioni sul loro contenuto:

- actions: si occupa del funzionamento principale del tool (avvio, configurazione dei threshold, esecuzione).
- analysis: si occupa dell'analisi statica del progetto target per rilevare Code Smells. Al suo interno ci sono le diverse tipologie di Code Smells le strategie per la detection (testuale e strutturale).
- gui: si occupa di creare e visualizzare la GUI basata su Java Swing. Ogni tipologia di Code Smell ha una class page ed un wizard specifici responsabili della visualizzazione delle informazioni relative al Code Smell rilevato e alla possibile soluzione.
- parser: si occupa del parsing del codice del progetto target per la creazione dei diversi tipi di bean. L'implementazione è affidata a PsiParser di IntelliJ.
- refactor: si occupa del refactoring del Code Smell, è incaricato di compiere due azioni che per disambiguazione sono definite:
  - splitting: nel caso dei Code Smells "Blob" e "Promiscuous package", viene applicata una strategia di Extract Class Refactoring per dividere la classe (o package) iniziale in classi (o packages) più piccoli. L'attuale implementazione prevede un algoritmo basato sulla Teoria dei Grafi.
  - refactoring: è il processo vero e proprio di modifica del codice sorgente e riceve in input il risultato della computazione dell'algoritmo di ECR. È effettuato utilizzando le API di IntelliJ.
- storage: contiene i beans che rappresentarono classi, metodi, pacchetti del progetto target.
- structuralMetrics: si occupa di gestire le metriche di Chidamber-Kemerer (CK metrics).
- topic: estrae i topic ovvero i termini più frequenti dal codice del progetto target.

Utilizzando "Structure101", che permette un'analisi delle collaborazioni tra i packages, è stato modellato il Class Diagram (Figura 2). Infine, per osservare meglio le classi coinvolte nell'attuale algoritmo di Extract Class Refactoring, è stato modellato il Sequence Diagram (Figura 3) relativo all'attuale processo di splitting dei Blob.

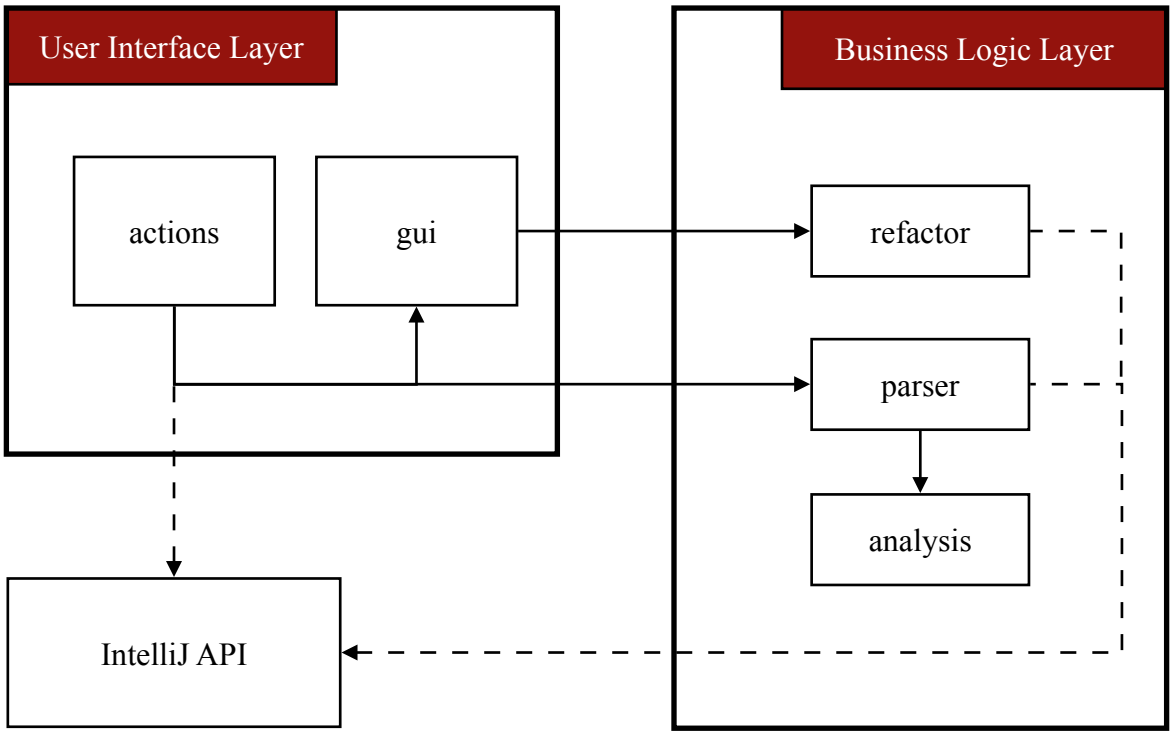

Figura 2: Class Diagram

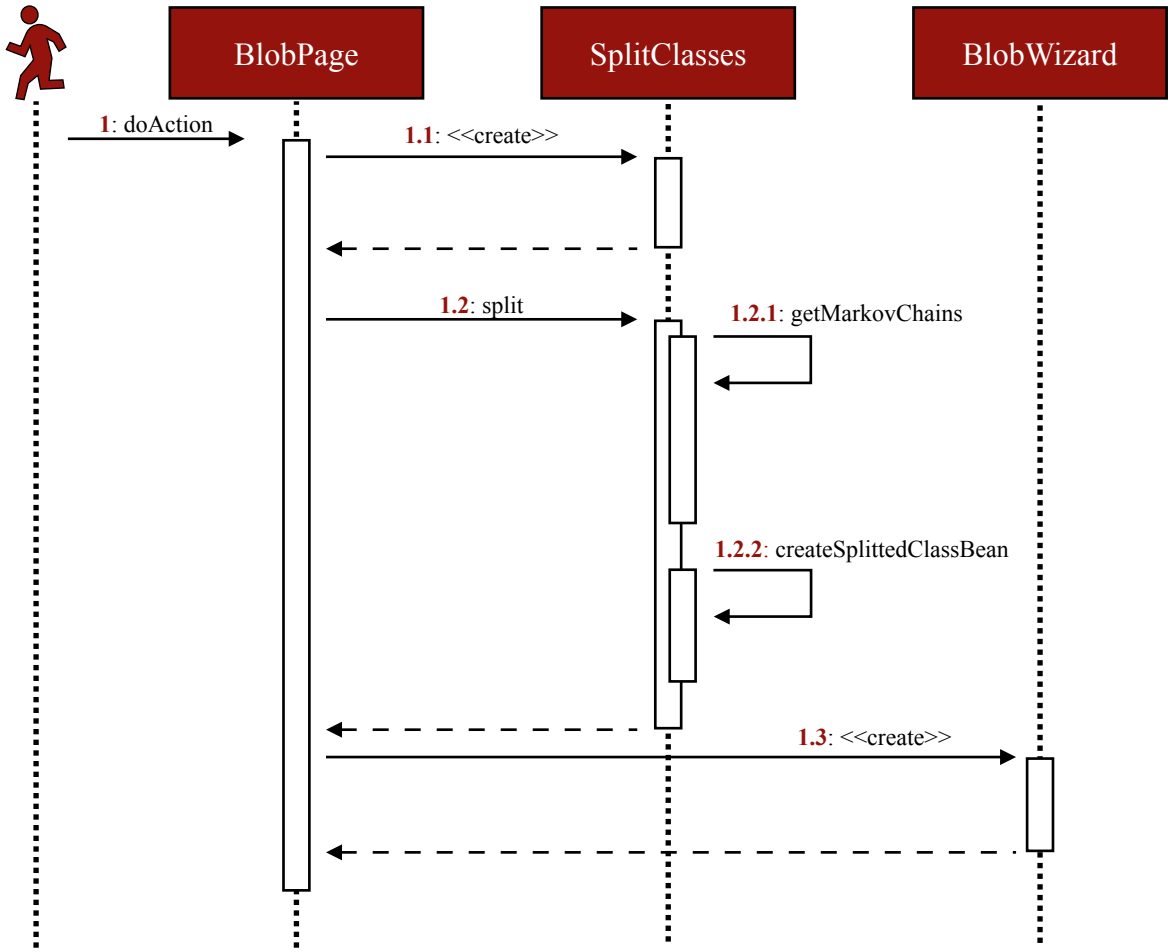

Figura 3: Sequence Diagram

Durante questa fase è stato identificato un design pattern noto come “Strategy Pattern” nel package analysis>code_smell_detection, come mostrato in Figura 4. Scopo del pattern è estrarre i differenti algoritmi di rilevazione dei Code Smell (testuale e strutturale) in classi esterne dette strategie che implementano i metodi della stessa generica interfaccia e permettono al contesto (*i.e.* la classe che applica una strategia scelta dal Client) di essere completamente indipendente dall’effettiva implementazione della strategia.

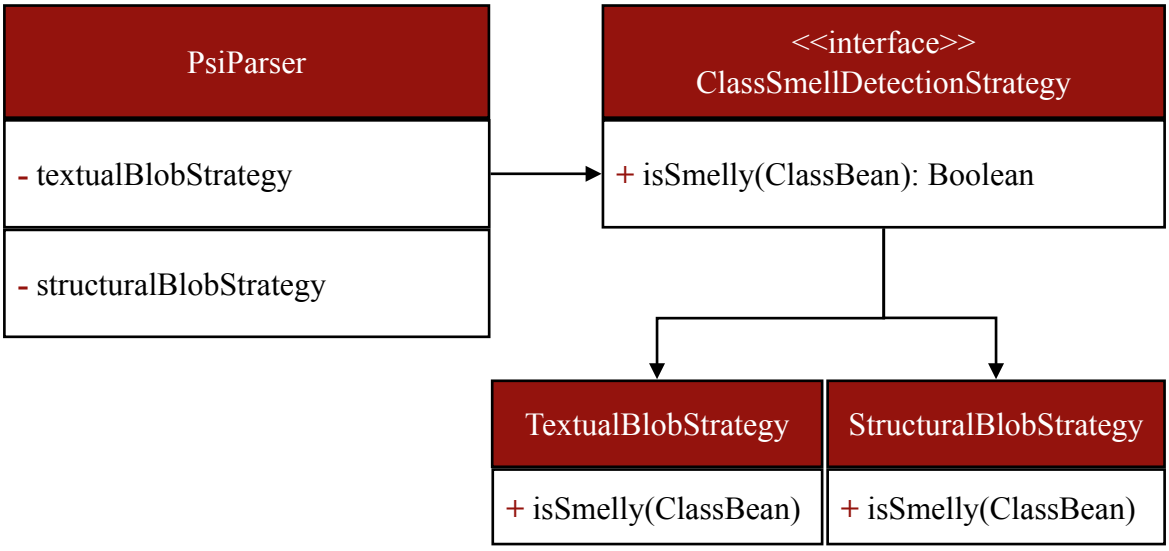

Figura 4: Strategy Pattern dell’operazione di detection di Code Smell (Blob)

Tale design pattern è presente anche nel package refactor e si occupa di differenziare le diverse strategie di refactoring a seconda della tipologia di Code Smell.

Impact Analysis

In questa sezione sono esposte le Impact Analysis relative alle CR. Utilizzando i risultati della fase di Reverse Engineering sono stati identificati gli Starting Impact Set (SIS) relativi alle due CR. In seguito, con l'utilizzo di Structure101, sono state analizzate le matrici di dipendenza per appurare i Candidate Impact Set (CIS).

CR_01

La CR_01 ha richiesto una Impact Analysis diversa; considerata la natura adattiva, sono state inclusi nel SIS i soli file di configurazione di Gradle.

	SIS	CIS	AIS	DIS	FPIS
Configuration files					
build.gradle	X	X	X		
gradle-wrapper.properties	X	X	X		

Per individuare il CIS, sono state seguite le linee guida, fornite dagli sviluppatori per facilitare e velocizzare il processo di migrazione [8][9][10], facendo molta attenzione ai cambiamenti che potenzialmente generano fallimenti nel processo di compilazione o nel prodotto finale (*i.e.* breaking changes). Dopo un processo di ispezione manuale dei cambiamenti si è passati ad un'analisi statica con "jdeps" al fine di ispezionare il codice sorgente per rilevare componenti, tools ed API rimossi.

Inoltre, sono state considerate eventuali perdite di compatibilità con le librerie esterne di cASpER:

- lucene-core v7.6.0
- sqlite-jdbc v3.25.2
- jsoup v1.11.3
- jfreechart v1.5.0
- junit v4.12

CR_02

Utilizzando i risultati della fase di Reverse Engineering relativi alle dipendenze tra il Business Logic Layer e lo User Interface Layer, al processo di splitting e allo Strategy Pattern è stato identificato lo SIS. In particolare, sono stati inclusi anche i componenti dell'attuale approccio di splitting (da convertire a strategia di splitting). In seguito, è stata analizzata la matrice di dipendenza per assicurarsi dell'effettivo impatto della modifica. Come mostrato nella tabella seguente, varie componenti presenti nel SIS non sono state incluse nel CIS e la loro assenza nel Actual Impact Set (AIS) conferma la bontà dell'analisi. Tuttavia alcuni componenti non preventivati sono stati modificati (sebbene in maniera lieve) nel corso dell'implementazione e per questo l'inclusività risulta diversa da 1.

	SIS	CIS	AIS	DIS	FPIS
Business Logic files					
SplitClasses.java	X	X	X		
Utility.java			X	X	
GUI files					
BlobPage.java	X	X	X		
BlobWizard.java	X	X	X		
ConfigureThreshold.java	X				
Test files					
JUnitTestSuite.java	X	X	X		
SplitClassTest.java	X				
Configuration files					
build.gradle	X	X	X		

Le metriche individuate relative al processo di Impact Analysis sono:

$$\text{Recall} = \frac{|CIS \cap AIS|}{|AIS|} \qquad \text{Precision} = \frac{|CIS \cap AIS|}{|CIS|}$$

$$\text{Inclusiveness} = \begin{cases} 1 & se\ AIS \subseteq CIS \\ 0 & altrimenti \end{cases}$$

	Recall	Precision	Inclusiveness
CR_01	1	1	1
CR_02	0,833	1	0

Progettazione e Realizzazione

CR_01

CR_01 non ha richiesto un'esplicita progettazione poiché, a valle della Impact Analysis, non sono stati rilevati API rimosse o componenti deprecati. I breaking changes riguardavano sostanzialmente modifiche sintattiche nei soli file di configurazione.

CR_02

L'attività relativa alla CR_02 è stata divisa nei seguenti task:

- T_01: progettare e realizzare il back-end
- T_02: progettare e realizzare il front-end
- T_03: testing delle modifiche, la progettazione e la realizzazione è descritta nel documento "Testing post-modifica".

CR_02: T_01

In questo task è aggiunta la principale funzionalità esterna:

- ER_01: "Il sistema deve applicare un'operazione di splitting basata sulla Teoria dei Giochi"

Per ottenere questa funzionalità sono state introdotte al sistema le seguenti funzionalità interne (non visibili all'utente):

- IR_01: "Il sistema deve calcolare la Jaccard Similarity tra due liste di stringhe"
- IR_02: "Il sistema deve normalizzare il contenuto testuale di una classe"
- IR_03: "Il sistema deve permettere il merge tra liste di stringhe che hanno una Jaccard Similarity maggiore di una soglia"
- IR_04: "Il sistema deve calcolare i payoffs data una lista di possibili scelte ed una matrice MethodByMethod"
- IR_05: "Il sistema deve trovare l'Equilibrio di Nash (o più di uno) data una matrice di payoffs"

Per coerenza ed efficienza è stato scelto di utilizzare lo stesso design pattern utilizzato per l'operazione di refactoring (i.e. Strategy Pattern), in modo tale da poter utilizzare all'occorrenza l'attuale strategia di splitting piuttosto che quella introdotta dalla CR, come mostrato in Figura 5.

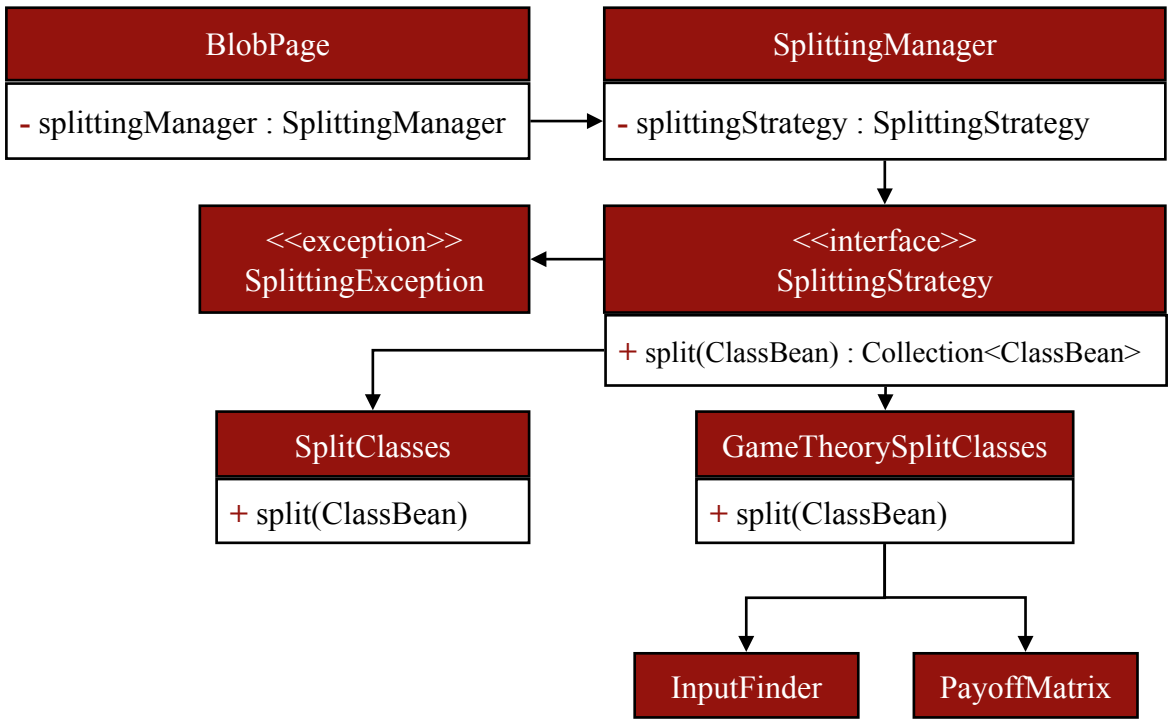

Figura 5: Strategy Pattern dell'operazione di Splitting

Le classi introdotte nel task T_01 sono:

- InputFinder.java: si occupa di produrre gli input per l'algoritmo di ECR applicando l'euristica basata su Latent Dirichlet Allocation; in particolare è stato utilizzato Mallet [11], un toolkit in Java per Natural Language Processing, Topic Modeling, *etc*., che fornisce un'implementazione dell'algoritmo Gibbs sampling. Mallet è stato quindi inserito tra le dipendenze di cASpER in build.gradle.
- GameTheorySplitClasses.java: è la strategia di splitting basata sulla Teoria dei Giochi. Si occupa di gestire l'intero processo iterativo (*i.e.* gioco), creando ad ogni round una PayoffMatrix; al termine del gioco si occupa di creare le nuovi classi estratte dal Blob.
- PayoffMatrix.java: rappresenta la matrice dei payoff e si occupa di eseguire operazioni su quest'ultima tra cui la computazione dei payoff e la ricerca dell'Equilibrio di Nash.

CR_02: T_02

Il task T_02 è incentrato sulle seguenti funzionalità esterne:

- ER_02: “La GUI deve permettere di abilitare lo splitting basato sulla Teoria dei Giochi”
- ER_03: “La GUI deve permettere di Modificare il valore di soglia della Jaccard Similarity utilizzata per il merge dei topic simili”

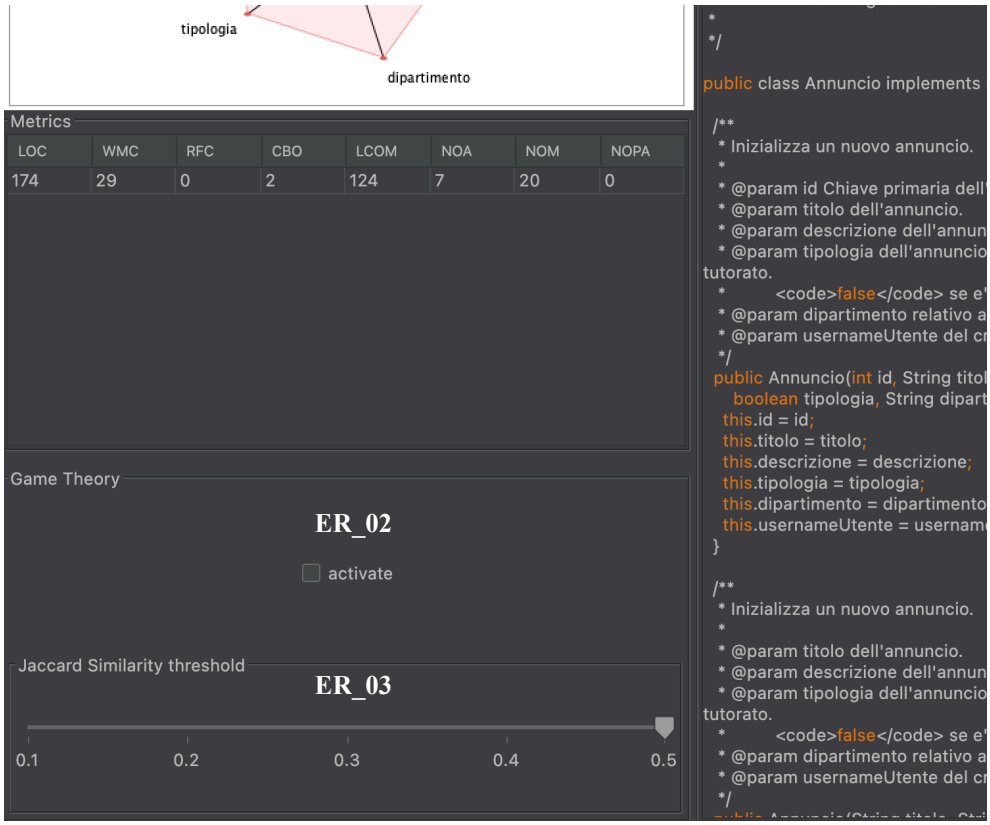

Figura 6: aggiunte alla BlobPage

Come mostrato in Figura 6, tali modifiche hanno riguardato la classe BlobPage.java ed inoltre è stato modificato il titolo della GUI prodotta da BlobWizard.java per notificare l’utente su quale strategia è stata usata per ottenere i risultati dell’operazione di splitting mostrata a schermo.

Riferimenti

[1] Stefano, Manuel & Gambardella, Michele & Pecorelli, Fabiano & Palomba, Fabio & Lucia, Andrea. (2020). cASpER: A Plug-in for Automated Code Smell Detection and Refactoring. 1-3. 10.1145/3399715.3399955.

[2] G. Bavota, R. Oliveto, A. De Lucia, A. Marcus, Y. Guehéneuc and G. Antoniol, "In medio stat virtus: Extract class refactoring through nash equilibria," 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 2014, pp. 214-223

[3] Bavota, G., De Lucia, A., Marcus, A. et al. Automating extract class refactoring: an improved method and its evaluation. Empir Software Eng 19, 1617–1664 (2014). https://doi.org/10.1007/s10664-013-9256-x

[4] Nash, J. (1951). Non-Cooperative Games. Annals of Mathematics, 54(2), 286–295. https://doi.org/10.2307/1969529

[5] https://github.com/MarcoCalenda14/cASpER

[6] https://blog.jetbrains.com/platform/2020/09/intellij-project-migrates-to-java-11/

[7] https://docs.gradle.org/current/userguide/compatibility.html/

[8] https://docs.oracle.com/en/java/javase/11/migrate/index.html/

[9] https://docs.gradle.org/current/userguide/upgrading_version_4.html/

[10] https://lp.jetbrains.com/gradle-intellij-plugin/

[11] http://www.cs.umass.edu/~mccallum/mallet/