Università degli Studi di Milano Bicocca

Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea Magistrale in Informatica

DPB: Una soluzione di benchmark per strumenti di design pattern detection

Relatore: Prof.ssa Francesca ARCELLI FONTANA

Correlatore: Dott. Marco ZANONI

Tesi di Laurea di:

Andrea Enrico Francis Caracciolo

Matr. 073494

Anno Accademico 2010/2011

Indice

Elenco delle figure

Elenco delle tabelle

Acronimi

DPB Design Pattern detection tools Benchmark platform

DP Design Pattern

DPD Design Pattern Detection

RDBMS Relational Database Management System

Introduzione

Uno dei campi dell'ingegneria del software che sta assumendo sempre maggiore importanza per il mantenimento e l'evoluzione dei sistemi è la reverse engineering [20, 73, 27]. Tale disciplina si occupa dell'analisi di sistemi software a partire dall'identificazione delle loro componenti fondamentali e dalle relazioni presenti tra di esse. Lo scopo principale di questa disciplina è facilitare la comprensione di un sistema attraverso la creazione di un modello di rappresentazione di livello di astrazione più alto, opportunamente strutturato e di facile lettura.

Tra le componenti che è possibile individuare all'interno di un sistema, vi sono i design pattern [43] (soluzioni di progettazione generiche applicabili a problemi ricorrenti). Per trovare i Design Pattern (DP) all'interno del codice sorgente di un'applicazione sono stati sviluppati diversi strumenti, basati su approcci molto diversi tra loro (vedi capitolo 1). Ognuno di essi è in grado di riconoscere un determinato sottoinsieme di design pattern e presenta i propri risultati seguendo uno schema di rappresentazione specifico e non standardizzato.

Questa mancanza di uniformità e l'impossibilità di effettuare una verifica automatica dei risultati generati dai singoli strumenti hanno finora impedito di effettuare un confronto oggettivo e condiviso tra gli strumenti sviluppati in questo ambito. La formulazione di una metrica di riferimento, che garantisca l'attribuzione di una misura di qualità consistente ad ognuno di questi strumenti, consentirebbe non solo di identificare facilmente il prodotto più adatto ai propri requisiti di analisi, ma anche di affrontare il complesso tema del riconoscimento dei DP in maniera costruttiva, offrendo ai ricercatori la possibilità di migliorare le tecniche che nel tempo si sono verificate essere le più valide. Ad oggi, le soluzioni proposte in tal senso sono state poche e non particolarmente significative (vedi capitolo 2).

Alla luce di questi fatti, è stato deciso di creare una nuova applicazione web per il confronto dei risultati prodotti da strumenti di riconoscimento di design pattern, chiamata Design Pattern detection tools Benchmark platform (DPB). L'applicazione in questione consente ai propri utenti di condividere e valutare istanze di DP individuate da uno specifico insieme di strumenti precedentemente registrati all'interno dell'applicazione. I dati ottenuti, derivanti dalle valutazioni espresse dagli utenti in merito al grado di correttezza delle singole istanze pubblicate, permetto-

no di calcolare una serie di metriche che consentono un confronto oggettivo tra gli strumenti coinvolti nella sperimentazione.

I punti distintivi che caratterizzano la nuova soluzione sono: la definizione di un nuovo meta-modello per la rappresentazione delle istanze di DP (descritto nel capitolo 3), l'implementazione di un algoritmo per la comparazione per istanze di DP, e la progettazione di una serie di funzionalità aventi lo scopo di semplificare il processo di valutazione e supportare al meglio l'utente nella propria analisi.

L'applicazione offre ad ogni utente la possibilità di iscriversi, condividere i risultati prodotti da uno strumento di riconoscimento di DP e valutare la correttezza delle istanze di DP condivise dagli altri membri dalla comunità.

In particolare, se un utente registrato desiderasse condividere i risultati ottenuti dall'esecuzione di un qualsiasi strumento di riconoscimento di DP, avrebbe la possibilità di farlo accedendo al proprio pannello d'amministrazione e caricando una versione opportunamente codificata della propria analisi di sistema[1]. Tale operazione prevede l'inserimento di alcuni meta-dati, utilizzati dalla piattaforma per connotare le istanze contenute nell'analisi presentata. Il formato di codifica adottato in questa fase, descritto nell'appendice A, è basato su un nuovo meta-modello appositamente creato nel contesto del progetto descritto in questa tesi.

Il capitolo 3, oltre a fornire un'esauriente descrizione del meta-modello sopra citato, offre una valutazione critica e comparativa di un insieme scelto di meta-modelli di analoga funzione presentati nell'ambito della letteratura scientifica di riferimento. Per ognuno di tali meta-modelli è stata definita una possibile soluzione di integrazione con il meta-modello adottato nell'applicazione presentata in questa tesi.

Una volta caricate, le istanze contenute nell'analisi di sistema sottomessa dall'utente saranno visibili agli altri visitatori della piattaforma, i quali potranno visionarle e valutarne la correttezza. Come descritto nel capitolo 4, ogni istanza di design pattern può essere analizzata attraverso la consultazione di varie viste grafiche (diagramma delle classi UML, vista dinamica con struttura ad albero, vista statica a riquadri annidati) e testuali (codice sorgente e documentazione Javadoc). Un'istanza può essere valutata esprimendo un giudizio numerico su una scala da 1 a 5, ed ogni valutazione può essere a sua volta votata (in modo positivo o negativo) dagli altri utenti della comunità. Ciò consente di valorizzare le valutazioni con molti voti favorevoli e nascondere quelle aventi un numero di voti inferiore ad una certa soglia prestabilita.

[1]Per "analisi di sistema" si intende il risultato prodotto da uno strumento di riconoscimento di design pattern applicato a uno specifico sistema software. Il risultato è composto da un numero variabile di istanze di design pattern.

Questo concetto di votazione democratica, basata sul giudizio di una comunità di utenti esperti, consente di creare una buona stima della qualità complessiva di uno strumento, fornendo una valida base per la formulazione di un benchmark.

Oltre ad offrire la possibilità di esprimere il proprio giudizio in relazione a singole istanze di design pattern, la piattaforma presenta altre funzionalità di notevole interesse. Tra queste citiamo il confronto per analisi di sistema, che permette di comparare le istanze contenute in due analisi di sistema fornendo una stima percentuale del loro grado di somiglianza, e quella del confronto per coppie di istanze, che, adottando un approccio grafico, illustra le differenze nella composizione delle strutture di due istanze di design pattern. Entrambe queste funzionalità consentono di cogliere le differenze presenti tra le varie istanze pubblicate sulla piattaforma, fornendo dati di confronto e spunti di valutazione agli utenti della comunità.

Il capitolo 5 descrive in dettaglio l'algoritmo di comparazione adottato per il calcolo della percentuale di similarità tra coppie di istanze di design pattern, fornendo una descrizione formale del suo comportamento ed un confronto analitico rispetto ad alcune varianti definite nel corso del progetto. Il capitolo si conclude con la descrizione di una sperimentazione empirica che permette al lettore di verificare, attraverso opportuni esempi, l'applicabilità e la consistenza dell'algoritmo.

In ultima analisi è opportuno soffermarsi anche sulla funzionalità di ricerca. Tale funzionalità non solo consente un'identificazione pratica e veloce delle istanze di design pattern a cui l'utente è interessato, ma permette anche di conoscere il numero delle istanze valutate come attendibili o non attendibili per ogni analisi di sistema selezionata. Questo dato arricchisce le informazioni comunemente restituite da uno strumento di ricerca, offrendo sin da subito una panoramica generale dei dati restituiti, che consente, in maniera facile e intuitiva, di formulare un giudizio qualitativo riguardo agli elementi presenti nel risultato.

L'applicazione presentata in questa tesi è stata ampiamente sperimentata secondo le modalità descritte nel capitolo 6. Attualmente ospita un numero consistente di istanze di DP, riconosciute dagli strumenti WOP (vedi sezione 1.2.1) e DPD-tool (vedi sezione 1.2.2) o riportate nel dataset P-Mart (vedi sezione 2.2).

Il progetto è stato (o sarà) presentato alla comunità scientifica nell'ambito delle seguenti conferenze:

- CSMR 2012, 16th European Conference on Software Maintenance and Reengineering, 27-30 Marzo 2012 (Szeged, Ungheria) [5].
- PATTERNS 2010, The Second International Conferences on Pervasive Patterns and Applications, 21-26 Novembre 2010 (Lisbona, Portogallo) [10].

Una descrizione sommaria del progetto è stata inoltre pubblicata in:

- ERCIM News No. 88, European Research Consortium for Informatics and Mathematics, Gennaio 2012 [4].

Capitolo 1

Design pattern detection

All'interno di questo capitolo verranno presentati alcuni tra i più noti approcci di Design Pattern Detection (DPD) presentati in letteratura nel corso di questi ultimi dieci anni. Ognuno di essi presenta una propria strategia di ricerca e in alcuni casi è associato allo sviluppo di uno strumento.

1.1 Classificazione degli approcci per la design pattern detection

Negli anni sono stati proposti molti approcci diversi alla DPD. Essi differiscono tra loro principalmente nei seguenti punti:

- *Tipo di analisi*: La maggior parte degli approcci sinora proposti in letteratura sono basati su analisi di tipo statico e sono tipicamente incentrati sulla valutazione di aspetti strutturali del codice (generalizzazione, associazione, aggregazione, attributi, e metodi). Altri approcci considerano utile verificare i risultati prodotti dall'analisi statica con l'analisi dinamica, osservando il comportamento del sistema (invocazione di metodi) in fase di esecuzione. Un numero ridotto di soluzioni ritiene utile considerare anche informazioni di tipo semantico (convenzioni nella denominazione delle entità di codice).

- *Tipo di riconoscimento*: Il riconoscimento di un'istanza di DP può essere esatto o approssimato. Nel primo caso, lo strumento terrà in considerazione i soli risultati in grado di soddisfare tutte le regole di ricerca definite. Alcune tecniche che ricadono in questa categoria sono: ricerca su grafi, pattern matching, espressioni regolari, query SQL, RDF e query SPARQL, SAT, logica del primo ordine. Nel secondo caso, lo strumento definisce una soglia di tolleranza minima rispetto al numero di condizioni che un'istanza deve essere in gra-

do di soddisfare per poter essere compresa nei risultati. Alcune tecniche che ricadono in questa categoria sono: data mining, fuzzy logic, algoritmi genetici.

- *Tipo di input*: Ogni strumento effettua la propria analisi a partire dalle informazioni estratte da un sistema specificato in ingresso. Tale sistema è codificato secondo un certo linguaggio di programmazione e può essere fornito in formato testuale o binario.
- *Rappresentazione intermedia*: Per agevolare l'analisi del sistema, risulta spesso vantaggioso tradurre le informazioni estratte dal sistema analizzato in un formato più astratto e facilmente interrogabile. Tra le tipologie di rappresentazione più comunemente adottate dagli strumenti di riconoscimento di DP si possono citare: Abstract Syntax Tree (AST), Abstract Semantic Graph (ASG), Unified Modeling Language (UML), meta-modelli ad-hoc.

1.2 Strumenti di design pattern detection

In questa sezione verranno presentati alcuni tra i più noti strumenti di DPD sinora presentati in letteratura. La presentazione degli strumenti è stata organizzata secondo una classificazione per tipo di analisi e tipo di riconoscimento (vedi sezione 1.1). In particolare sono state definite le seguenti categorie: strumenti basati su analisi statica e strumenti basati su analisi dinamica. Ognuna delle due categorie è stata ulteriormente suddivisa in: strumenti con riconoscimento esatto e strumenti con riconoscimento approssimato.

1.2.1 Analisi statica con riconoscimento esatto

Pat

Basato sull'approccio di Kramer et al.[69]. Lo strumento propone una metodologia basata sull'analisi strutturale del codice finalizzata alla creazione di una base di conoscenza Prolog. Tale base di conoscenza, integrata con una serie di regole derivate dalle definizioni dei design pattern, può essere facilmente interrogata e restituire l'elenco dei pattern individuati. Questo approccio è limitato ai pattern strutturali.

Homepage: `n.d.`

DP++

DP++, sviluppato da Bansiya et al. [14], propone un approccio basato sul riconoscimento delle relazioni strutturali presenti tra i componenti del sistema analizzato.

Il processo di riconoscimento dei DP è descritto in maniera sommaria e pertanto non consente di addurre ulteriori commenti.

Homepage: `n.d.`

SPOOL

Costruito secondo i principi dell'approccio definito da Keller et al. [64]. Lo strumento propone una procedura divisa in due fasi: una automatizzata e l'altra interattiva. Nella prima fase vengono identificate alcune strutture generiche di pattern. Tali strutture verranno successivamente mostrate in forma grafica (utilizzando UML/CDIF) ad un analista al quale spetterà il compito di raffinare la selezione eliminando i risultati scorretti e completando quelli corretti .

Homepage: `n.d.`

Hedgehog

Definito secondo le specifiche dell'approccio di Blewitt et al. [18]. Tale approccio definisce un nuovo linguaggio in logica del primo ordine chiamato SPINE, che consente di descrivere in forma sintetica alcuni aspetti strutturali e semantici di un design pattern utilizzando una sintassi simile a Prolog. Questa descrizione può essere interpretata da un sistema ad-hoc in grado di provare se una certa classe soddisfi o meno i requisiti imposti dal pattern. Questo approccio supporta pattern strutturali, comportamentali e semantici.

Homepage: `n.d.`

JBOORET

JBOORET, sviluppato da Mei et al. [61], analizza il codice sorgente di un programma C++ inserendo le informazioni estratte in una base dati. Tali informazioni vengono successivamente utilizzate per ricostruire delle viste di alto livello del codice analizzato. L'utente dello strumento può esplorare i modelli così ottenuti per individuare l'eventuale presenza di DP.

Homepage: `n.d.`

Ptidej

Ispirato dall'approccio di Albin-Amiot et al.[1], i quali propongono una soluzione basata sulla modellazione dei design pattern come entità facilmente manipolabili e analizzabili. I modelli così descritti possono essere interpretati come una serie di

vincoli che una certa parte del codice deve rispettare per essere considerata conforme al pattern.

Homepage: `www.ptidej.net`

SPQR

Concepito sulla base delle teorie espresse da Smith et al. [85], i quali propongono un approccio basato sul rho-calcolo e sull'utilizzo di un tool di dimostrazione di teoremi chiamato OTTER. I dati necessari all'elaborazione automatica da parte di OTTER provengono dal codice sorgente, opportunamente analizzato e tradotto dal programma di compilazione gcc, da un catalogo EDP (Elemental Design Patterns: un insieme di concetti base utilizzati per definire i design pattern) e da un insieme di formalizzazioni in rho-calcolo. Questo procedimento è in grado di individuare solamente pattern strutturali.

Homepage: `n.d.`

CroCoPat

CroCoPat, realizzato nell'ambito delle ricerche condotte da Beyer et al. [16], è uno strumento di interrogazione che elabora dati rappresentati in forma relazionale. Lo strumento accetta in input il formato RSF (Rigi Standard Format), popolare tra gli strumenti di reverse engineering, e ne deriva dei diagrammi di decisione binari (BDD). Tali diagrammi, abbinati a tecniche di calcolo basate su predicati logici, consentono un'analisi formale del sistema, consentendo l'individuazione di pattern attraverso semplici query logiche.

Homepage: `www.sosy-lab.org/~dbeyer/CrocoPat/`

DPRE

Basato sull'approccio, composto in due fasi, di Costagliola et al. [24]. Nella prima avviene il processo di estrazione del diagramma delle classi UML a partire dal codice sorgente. Questo diagramma, codificato in SVG, viene successivamente sottoposto ad un secondo tool avente il compito di ispezionare il diagramma e, applicando concetti derivati dalla teoria dei grafi, rilevare l'eventuale presenza di design pattern all'interno del sistema.

Homepage: `n.d.`

PINOT

Basato sull'approccio definito da Shi et al. [82], i quali criticano le tecniche presenti nella letteratura precedente proponendo una ri-classificazione dei design pattern abbinata ad un processo di ricerca diversificato per categoria. Tale approccio sfrutta le caratteristiche strutturali e comportamentali di ogni gruppo di pattern e promette di fornire risultati più accurati e rapidi da generare. Lo strumento proposto è stato sviluppato come modifica del compilatore open source jikes [23].

Homepage: `www.cs.ucdavis.edu/~shini/research/pinot/`

DP-Miner

Ideato e implementato da Dong et al. [31], i quali presentano una soluzione articolata in tre fasi. La prima prevede l'analisi strutturale del codice, in cui vengono confrontati i dati ricavati dal diagramma delle classi con un certo insieme di metriche relative alla definizione dei design pattern. I risultati prodotti a questo passo verranno successivamente raffinati da una seconda analisi di tipo comportamentale e da un'ultima analisi di tipo sintattico, in cui è prevista la verifica di alcune convenzioni sulla denominazione delle classi.

Homepage: `www.utdallas.edu/~jdong/DesignPattern/DP_Miner/index.htm`

DPJF

Realizzato da Binun et al., i quali introducono una metodologia innovativa basata sull'utilizzo ottimizzato di strategie di analisi pre-esistenti. La soluzione presentata, deriva da uno studio accurato dei punti di debolezza che maggiormente incidono sui risultati ottenuti da altri approcci. L'applicazione della metodologia vanta una "precision" pari al 100%, una notevole "recall" ed un ottima media nei tempi di esecuzione. I dettagli dell'approccio sono descritti in un articolo che verrà pubblicato nei proceedings di CSMR 2012 (16th European Conference on Software Maintenance and Reengineering).

Homepage: `sewiki.iai.uni-bonn.de/research/dpd/dpjf/start`

WOP

Sviluppato da Dietrich et al. [29, 30]. WOP si distingue da altri strumenti per la notevole attenzione dedicata alla specifica delle informazioni associate alle istanze riconosciute. Tali informazioni sono conformi ad un ontologia OWL pubblicata sulle pagine del sito di riferimento del progetto [28]. La tecnica adottata dallo strumento si basa sulla verifica di formule descritte in logica del primo ordine.

Homepage: `www-ist.massey.ac.nz/wop/`

RSA

Una suite commerciale realizzata da IBM [62]. Le tecniche utilizzate per il reperimento dei DP non sono state rese pubbliche, ma è facile presumere che i risultati generati dallo strumento siano il frutto di un analisi di tipo statico (non è richiesta alcuna traccia di esecuzione) e che lo strumento non riconosca istanze parziali (i risultati non associano alcun grado di precisione al riconoscimento dell'istanza).

Homepage: `www.ibm.com/developerworks/rational/products/rsa/`

D-CUBED

Uno strumento sviluppato da Stencel et al. [84]. Le regole di riconoscimento adottate per la ricerca dei DP sono specificate da formule in logica del primo ordine. Tali formule sono successivamente tradotte in query SQL, per consentire l'interrogazione di una base di dati in cui sono stati precedentemente salvati gli elementi di codice estratti dall'applicazione analizzata.

Homepage: `www.yonlabs.com/dcubed.html`

1.2.2 Analisi statica con riconoscimento approssimato

Columbus

Basato sull'approccio di Ferenc et al. [35]. Lo strumento si serve di tecniche di apprendimento automatico abbinate ad approcci "classici" basati sulla ricerca dei grafi. La procedura adottata prevede una prima fase di ispezione del codice sorgente, opportunamente trasformato in grafo semantico astratto (ASG) e confrontato con le definizioni strutturali dei design pattern supportati, seguita da una fase di raccolta delle informazioni, in cui si estraggono le caratteristiche che non appartengono alla descrizione strutturale del design pattern e le si sottopone ad un sistema in grado di apprendere.

Homepage: `n.d.`

DPD-tool

Ideato e sviluppato da Tsantalis et al. [90], i quali scelgono di rappresentare il sistema in analisi utilizzando un semplice albero di ereditarietà e una serie di matrici $n \times n$ (dove n è il numero delle classi) contenenti informazioni di carattere strutturale (ad

es. associazione, generalizzazione, metodi di invocazione, ecc...). La ricerca dei design pattern inizia dall'esplorazione dell'albero e può essere allargata alla lettura delle matrici. Ad ogni classe analizzata viene assegnato un punteggio che definisce il grado di similarità con il ruolo di un determinato pattern.

Homepage: `java.uom.gr/~nikos/pattern-detection.html`

MARPLE

Sviluppato dal gruppo di Arcelli et al. [6]. L'approccio consiste nell'affrontare l'analisi di un sistema a partire dal suo abstract syntax tree (AST). Tale struttura dati viene utilizzata come fonte per la ricerca di alcuni elementi basilari del sistema (Elemental Design Patterns, design pattern clues e micro pattern) e la raccolta di un determinato insieme di metriche. I dati così ricavati andranno confrontati con alcune regole di riconoscimento e potranno essere utilizzati per l'identificazione di design pattern all'interno del sistema analizzato. I pattern rilevati nella prima fase verranno poi valutati e ordinati per rilevanza da un algoritmo di machine learning precedentemente addestrato e configurato. I nuovi pattern (corretti e non) rilevati dall'utente possono essere aggiunti alla base di conoscenza dal tool per migliorarne le performance.

Homepage: `essere.disco.unimib.it/reverse/Marple.html`

1.2.3 Analisi dinamica con riconoscimento esatto

DPVK

Strumento sviluppato da Wang et al. [93]. L'approccio considerato si compone di una prima fase di ispezione statica del codice, in cui viene identificato un primo insieme di corrispondenze tra alcune parti del codice e le definizioni strutturali dei pattern, seguita da una fase di analisi dinamica, in cui è possibile raffinare i risultati ottenuti basandosi sulle informazioni di comportamento dei vari pattern supportati.

Homepage: `n.d.`

mb-pde

mb-pde, sviluppato da Marcel Birkner [17], consente di riconoscere un vasto numero di DP offrendo al tempo stesso una buona flessibilità nella definizione delle regole di riconoscimento. Il processo di riconoscimento prevede una prima fase di analisi statica, in cui le informazioni estratte dal codice sorgente del programma analizzato vengono confrontate con un diagramma delle classi UML del DP che si desidera

individuare. Nella fase successiva, lo strumento confronta il comportamento in fase di esecuzione di una versione instrumentata del programma con un diagramma di sequenza UML dello stesso DP. Le informazioni acquisite vengono infine comparate per fornire una valutazione del grado di corrispondenza tra il programma e i diagrammi forniti.

Homepage: `code.google.com/p/mb-pde/`

ePAD

EPad è un plug-in di Eclipse sviluppato da Andrea De Lucia et al [25] che consente di individuare DP strutturali e comportamentali. Il processo di riconoscimento si compone di una prima fase di analisi statica, in cui un diagramma delle classi UML, derivato dal programma in analisi, viene esaminato al fine di individuare particolari sequenze di testo che andranno a comporre delle prime ipotesi in merito alle possibili istanze presenti nel codice. Tali ipotesi verranno successivamente verificate attraverso una seconda analisi di tipo dinamico.

Homepage: `www.sesa.dmi.unisa.it/ePAD/`

1.2.4 Analisi dinamica con riconoscimento approssimato

Fujaba/Reclipse

Definito secondo le specifiche dell'approccio di Niere et al. [74] e successivamente raffinato da Wendehals [94]. Lo strumento prevede la traduzione del codice sorgente del sistema in esame in una struttura dati navigabile rappresentabile in forma di grafo sintattico astratto (abstract syntax graph, ASG). Tale grafo, confrontato iterativamente con una serie di regole caratterizzanti un certo pattern, portano alla scoperta di un insieme di istanze. Per evitare cicli computazionali superflui e ridurre al minimo il numero di falsi risultati, il processo di scoperta viene coadiuvato dall'intervento di un agente umano.

Homepage: `www.fujaba.de/no_cache/projects/reengineering/reclipse.html`

1.3 Approcci teorici di design pattern detection

Oltre alle tecniche presentate nella sezione 1.2, esistono altri approcci per la DPD che, dal momento della loro presentazione in letteratura, non sono mai stati concretamente implementati e sperimentati. La presentazione degli strumenti è stata organizzata secondo una classificazione per tipo di analisi e tipo di riconoscimento (vedi sezione 1.1). In particolare sono state definite le seguenti categorie: strumenti

basati su analisi statica e strumenti basati su analisi dinamica. Ognuna delle due categorie è stata ulteriormente suddivisa in: strumenti con riconoscimento esatto e strumenti con riconoscimento approssimato.

1.3.1 Analisi statica con riconoscimento esatto

Seemann et al. (1998)

Seemann et al. [81] presentano un approccio basato sull'analisi e il raffinamento di un grafo rappresentante la struttura della base di codice del sistema analizzato utilizzando la logica del primo ordine. I risultati ottenuti da questa tecnica sono limitati a pattern strutturali.

Antoniol et al. (1998)

Antoniol et al. [2] definiscono un approccio che prevede la trasformazione del codice sorgente in un modello intermedio basato sulla formalizzazione di alcuni concetti propri dei linguaggi object-oriented. Analizzando le metriche estratte da questo modello e confrontandole con quelle proprie di un certo design pattern è possibile calcolare la possibilità che una certa classe possa appartenere o meno al design pattern.

Asencio et al. (2002)

Asencio et al. [13] rappresentano i pattern come classi correlate con altre classi rappresentanti a loro volta dei pattern, delle qualità del software, dei concetti di progettazione o delle condizioni di applicabilità. Ad ogni pattern corrisponde uno strumento di riconoscimento (recognizer) scritto in un linguaggio di specifica basato sugli insiemi. Le specifiche utilizzate servono a definire un insieme di vincoli che determinano il grado di conformità del codice al pattern. Ogni recognizer è da intendere come una soluzione imperfetta, facilmente modificabile ed estendibile dagli utenti interessati.

Espinoza et al. (2002)

Espinoza et al. [34] partono del presupposto che i design pattern possano essere descritti in forma soddisfacente utilizzando un modello che definisca le relazioni strutturali presenti tra i suoi componenti. Tali relazioni possono essere: ereditarietà, aggregazione, conoscenza e creazione (come definito da [43]). Una volta descritto il codice sorgente utilizzando lo stesso modello, sarà sufficiente confrontarlo

con i modelli dei vari pattern. Questo approccio consente la rilevazione di pattern strutturali.

Zhang et al. (2004)

Zhang et al. [95] propongono di rappresentare il sistema in analisi e le definizioni dei pattern utilizzando grafi estesi basati sulle seguenti relazioni: associazione, composizione ed ereditarietà. La ricerca dei pattern viene affrontata con un semplice confronto tra grafi alla ricerca di eventuali corrispondenze (isomorfismi totali o isomorfismi tra sotto-grafi).

Streitferdt et al. (2005)

Streitferdt et al. [86] discutono una soluzione basata su un nuovo paradigma per la descrizione dei design pattern, secondo il quale la struttura di ogni pattern dev'essere caratterizzata da una serie di elementi necessari, vietati, e da ignorare. Tale soluzione arricchirebbe il numero di discriminanti da poter utilizzare in fase di ricerca e restituirebbe un insieme di risultati più preciso rispetto ad altre tecniche illustrate in precedenza.

Kaczor et al. (2006)

Kaczor et al. [63] concepiscono la DPD come un problema di calcolo combinatorio, proponendo una soluzione basata su vettori di bit. Un algoritmo basato su questa tecnica non solo permette di affrontare l'analisi di un sistema in modo efficiente ma consente anche di terminare le operazioni di riconoscimento in un tempo indipendente dalla lunghezza del programma analizzato.

Bayley et al. (2007)

Bayley et al. [15] propongono un approccio basato sulla descrizione delle regole di riconoscimento attraverso formule in logica del primo ordine. I dati estratti dal sistema analizzato, in forma di diagramma delle classi UML, sono specificati da un'istanza del meta-modello GEBNF, un'estensione di EBNF che permette di rappresentare le informazioni contenute in un qualsiasi modello grafico.

Pande et al. (2010)

Pande et al. [56] propongono un nuovo approccio basato sulla ricerca di corrispondenze tra grafi a partire da diagrammi delle classi UML. In particolare, l'approccio

esaminato prevede una prima fase di analisi, in cui nodi e archi vengono etichettati rispettivamente in funzione della propria profondità e natura, e una seconda fase di ricerca, in cui viene applicato un algoritmo di ricerca denominato DNIT.

Rasool et al. (2010)

Rasool et al. [80] propongono un approccio innovativo basato sull'introduzione di annotazioni all'interno del codice sorgente del sistema analizzato. L'utilizzo di annotazioni consentirebbe un trasferimento di conoscenza semplice e diretto tra i vari attori responsabili della progettazione e del mantenimento del sistema. La scelta del linguaggio di specifica da adottare nelle annotazioni è lasciata allo sviluppatore.

1.3.2 Analisi statica con riconoscimento approssimato

Guéhéneuc et al. (2004)

Guéhéneuc et al. [55] individuano una soluzione per la DPD in un tecnica basata sull'apprendimento automatico. La sua proposta si compone in una prima fase di raccolta delle informazioni, in cui un agente umano si occupa di reperire e catalogare un buon campione di micro-architetture corrispondenti ad uno o più design pattern, seguita da una fase di automazione, in cui un tool analizza e associa ad ogni design pattern una serie di metriche derivate dal campione elaborato.

Guéhéneuc et al. (2008)

Guéhéneuc et al. [54] introducono un approccio, basato su tre livelli, per l'identificazione e la tracciabilità di micro-architetture all'interno del codice sorgente. L'approccio prevede l'adozione di un certo insieme di meta-modelli, applicabili sia al codice sorgente del programma analizzato sia ai DP che si desidera ricercare all'interno di tale programma. Lo strumento costruito a supporto dell'approccio descritto si occuperà di trasformare la descrizione di ogni DP in un insieme di vincoli che possano essere utilizzati per la ricerca delle rispettive micro-architetture all'interno del modello che rappresenta il programma analizzato.

Guéhéneuc et al. (2009)

Guéhéneuc et al. [51] presenta un approccio in due fasi. La prima fase prevede l'analisi di un certo numero di sistemi e la successiva identificazione manuale di classi facenti parte di istanze di DP. Le classi identificate sono messe in correlazione con il DP a cui sono state associate. Tali relazioni sono in seguito sottoposte ad un

algoritmo di apprendimento automatico che applicherà la conoscenza acquisita per il riconoscimento di ulteriori istanze.

1.3.3 Analisi dinamica con riconoscimento esatto

Heuzeroth et al. (2003)

Heuzeroth et al. [58] propongono un approccio in cui si effettua una prima analisi del codice sorgente per mezzo del tool di compilazione Recoder, capace di analizzare il codice e generare una sua rappresentazione in forma di albero sintattico astratto (AST). Segue una prima analisi statica dell'albero in grado di rilevare una lista di candidati successivamente raffinata attraverso un processo di analisi dinamica. I pattern riconosciuti sono di tipo strutturale e comportamentale.

Hayashi et al. (2008)

Hayashi et al. [57] descrivono un approccio integrato, basato su analisi statica e dinamica. I DP supportati dal processo, sono descritti come meta-pattern di Pree [79]. Ciò consente di ottenere una riduzione dei tempi di elaborazione attraverso una semplificazione della procedura di identificazione. L'individuazione delle istanze di DP è ottenuta attraverso la formulazione di condizioni di ricerca in forma di predicati Prolog.

1.4 Precisazioni in merito agli strumenti presentati

In questa sezione sono presenti tre tabelle, contenenti informazioni rilevanti associate agli strumenti descritti nella sezione 1.2.
La sintesi presentata attraverso queste tabelle offre un'utile visione d'insieme degli strumenti presentati. Attraverso di esse è possibile maturare un primo confronto superficiale basato su un insieme scelto di informazioni. I dati raccolti forniscono inoltre informazioni utili ad identificare eventuali conflittualità tra determinati strumenti e l'applicazione sviluppata nel contesto del progetto discusso in questa tesi.
Parte delle informazioni presenti in queste tabelle, è tratta da un articolo di Dong et al. [32].

La tabella 1.1 consente di confrontare gli strumenti sulla base dei DP che ognuno di essi è in grado di riconoscere.

La tabella 1.2 propone un confronto basato su alcune informazioni di carattere generale. I dati raccolti consentono di classificare gli strumenti in base ai linguaggi di programmazione supportati, al formato di esportazione previsto, alla presenza di documentazione e alla data dell'ultimo aggiornamento.

Strumento	Abstract Factory	Adapter/Command	Builder	Bridge	Chain of Respons.	Composite	Decorator	Facade	Factory method	Flyweight	Mediator	Observer	Prototype	Proxy	Singleton	Strategy/State	Template Method	Visitor
Columbus		×														×		
CroCoPat	- informazione non disponibile -																	
D-CUBED	×		×						×						×			×
DP++	- informazione non disponibile -																	
DPJF					×	×	×					×		×				
DPD-tool		×				×	×		×			×	×		×	×	×	
DP-Miner		×		×		×										×		
DPRE		×		×		×	×							×				
DPVK	- informazione non disponibile -																	
ePAD		×		×		×	×	×				×		×		×	×	×
Reclipse				×		×										×		
Hedgehog		×		×					×					×	×	×		
JBOORET	- riconoscimento effettuato dall'utente -																	
MARPLE	×					×												×
mb-pde	×	×	×	×	×	×	×		×	×	×	×	×	×	×	×	×	×
Pat		×		×		×	×							×				
PINOT	×	×		×	×	×	×	×	×	×	×	×		×	×	×	×	×
Ptidej	- algoritmi di riconoscimento definiti dall'utente -																	
RSA							×		×			×			×			×
SPOOL		×		×					×					×	×			
SPQR	- algoritmi di riconoscimento definiti dall'utente -																	
WOP	×	×		×		×								×	×		×	

Tabella 1.1: Design pattern supportati dagli strumenti citati

L'informazione relativa al formato di esportazione permette di identificare gli strumenti potenzialmente compatibili con l'applicazione presentata nel corso di questa tesi. Se uno strumento è in grado di esportare le istanze riconosciute in un formato testuale strutturato o semi-strutturato riconducibile ad un file XML, allora tale strumento soddisfa i requisiti minimi necessari a poter ipotizzare un percorso di integrazione con la piattaforma descritta nei capitoli seguenti (per maggiori informazioni si invita a leggere i capitoli 3 e 4).
La data dell'ultimo aggiornamento, qualora disponibile, consente di valutare il grado di mantenimento di un determinato progetto.

La tabella 1.3 consente di confrontare gli strumenti sulla base dei programmi analizzati dagli autori durante il periodo di sperimentazione dello strumento. I dati riportati possono essere utilizzati per consentire una stima indiretta del grado di scalabilità dei singoli strumenti.

Strumento	Linguaggi supportati	Esportazione	Documentaz.	Ultimo aggiornamento
Columbus	C++	n.d.	n.d.	n.d.
CroCoPat	Java	RSF	Si	02/2008
D-CUBED	Java	n.d.	n.d.	n.d.
DP++	C++	n.d.	n.d.	n.d.
DPJF	Java	Prolog	Si	11/2011
DPD-tool	Java	XML	Si	05/2010
DP-Miner	Java	XML	Si	01/2010
DPRE	C++ e Java	n.d.	n.d.	n.d.
DPVK	Eiffel	n.d.	n.d.	n.d.
ePAD	Java	non supportata	Si	10/2010
Reclipse	Java	non supportata	Si	10/2011
Hedgehog	Java	n.d.	n.d.	n.d.
JBOORET	C++	n.d.	n.d.	n.d.
MARPLE	Java	XML	n.d.	12/2011
mb-pde	Java	XML	Si	03/2008
Pat	C++	n.d.	n.d.	n.d.
PINOT	Java	testo non strutturato	Si	10/2004
Ptidej	Java	testo semi-strutturato	Si	04/2006
RSA	Java	XML	Si	11/2011
SPOOL	C++	n.d.	n.d.	n.d.
SPQR	C++	XML	n.d.	n.d.
WOP	Java	XML	Si	1.4.3

Tabella 1.2: Caratteristiche principali degli strumenti citati

Strumento	Java AWT	Java Swing	Java IO	Java JDK	JHotDraw	JRefactory	JUnit	JEdit	ArgoUML	Eclipse	Batik	JasperReports	StarWriter	TrackerLib	Galib	Libg++	Mec	Altri
Columbus													×					
CroCoPat	×			×	×													×
D-CUBED	- informazione non disponibile -																	
DP++	- informazione non disponibile -																	
DPJF	×		×		×			×	×									
DPD-tool					×	×	×											
DP-Miner	×				×		×											×
DPRE															×	×	×	×
DPVK	- informazione non disponibile -																	
ePAD					×													
Reclipse	×																	
Hedgehog	×																	
JBOORET	- informazione non disponibile -																	
MARPLE											×	×						
mb-pde	- informazione non disponibile -																	
Pat	- informazione non disponibile -																	
PINOT	×	×			×													×
Ptidej	×				×		×	×										
RSA	- informazione non disponibile -																	
SPOOL																		×
SPQR														×				×
WOP	- informazione non disponibile -																	

Tabella 1.3: Sperimentazioni effettuate dagli autori degli strumenti citati

Capitolo 2

Soluzioni per il confronto di risultati di design pattern detection

Come mostrato nel capitolo precedente, esistono molti strumenti per il riconoscimento dei DP. Ciascuno di essi adotta un approccio ben distinto dagli altri e ciò porta ad ottenere risultati tra di loro molto diversi. A causa della mancanza di un meta-modello di rappresentazione comune, tali risultati non sono comparabili tra di loro.
In questo capitolo saranno analizzate le principali proposte per il confronto dei risultati prodotti da strumenti di DPD presenti in letteratura. Per ogni proposta verranno riportate le caratteristiche principali e alcune critiche personali.

2.1 DEEBEE

Fulop et al. sono stati i primi, e per ora gli unici, ad aver realizzato e reso pubblica un'applicazione per il benchmark di strumenti di DPD[42]. L'applicazione, chiamata DEEBEE, è stata sviluppata come estensione della piattaforma Trac [83] ed è liberamente fruibile online [37]. Di seguito verranno esaminate le principali funzionalità di questo progetto.

Caricamento dei risultati

DEEBEE offre la possibilità di caricare nuovi risultati, prodotti da strumenti di DPD, in formato testuale CSV (Comma-separated values).

Per caricare nuovi risultati, è necessario compilare una maschera, che prevede l'inserimento di un certo numero di meta-dati (nome dello strumento adottato, nome del software analizzato, linguaggio di programmazione del software analizzato), dell'indirizzo SVN, CVS o HTTP tramite il quale reperire il codice sorgente del software analizzato e il caricamento di un file CSV che contenga tutti i dati necessari per ricostruire le singole istanze riconosciute (vedi figura 2.1).
Il formato adottato per la specifica delle istanze di DP che compongono il risultato dello strumento di DPD (descritto nelle pagine del sito web dell'applicazione [38]), risulta elementare e poco strutturato. Ogni istanza di DP è rappresentata come una semplice lista di ruoli a cui è associato il nome di un'arbitraria entità di codice, il nome del file che la contiene e i numeri di linea che definiscono l'inizio e la fine di tale entità. La validazione delle informazioni specificate è solamente di tipo sintattico. La corrispondenza tra valori specificati e informazioni reali non è verificata. Il formato utilizzato per la specifica delle istanze non prevede la possibilità di includere istanze aventi più ruoli dello stesso tipo.

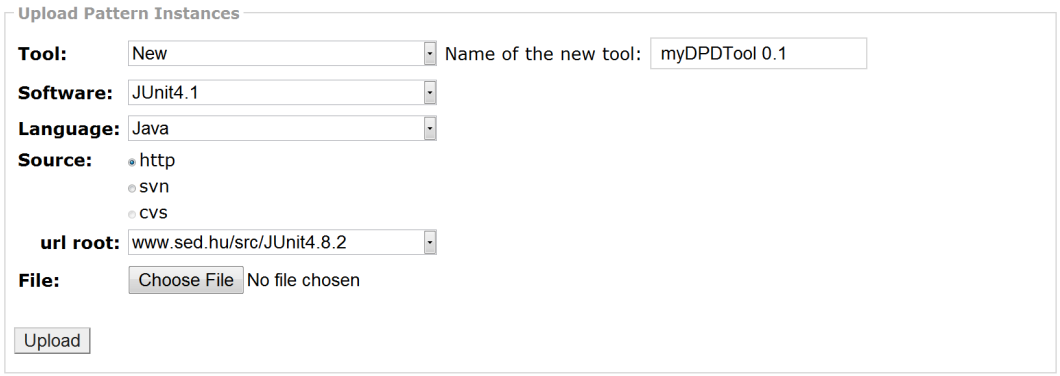

Figura 2.1: DEEBEE: upload di nuovi contenuti.

Ricerca delle istanze

DEEBEE consente di interrogare la base dati dei risultati attraverso la compilazione di una maschera di ricerca (vedi figura 2.2).
I campi della maschera comprendono: Linguaggio, Software, Tool e Pattern. Tutti i campi sono facoltativi, e possono assumere un valore specifico (selezionabile da un menu a tendina) o il valore speciale "All" (che corrisponde a tutte le voci presenti nel menu). I risultati della ricerca sono riportati in una tabella ordinata alfabeticamente (vedi figura 2.3).
Le istanze individuate sono riportate in forma di semplici stringhe numeriche. Non sono presenti alcune informazioni aggiuntive che permettono di distinguere un'istanza da un'altra. Inoltre non sono presenti metriche sintetiche in grado di esprimere

i valori di precision e recall attribuibili a singoli strumenti. L'unica informazione relativa alla "qualità" dei risultati rappresentati, può essere ricostruita accedendo alla pagina delle informazioni statistiche (descritta nel paragrafo seguente).

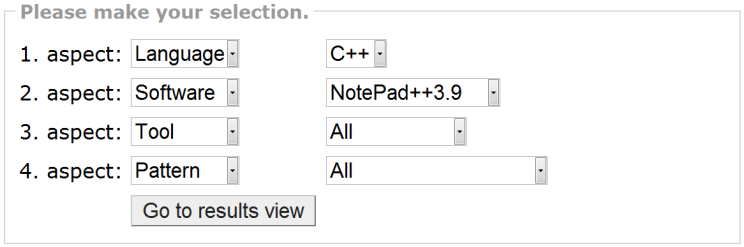

Figura 2.2: DEEBEE: campi di ricerca.

Language	Software	Tool	Pattern	Pattern instance ids
C++	NotePad++3.9	Human	Prototype	#1134
			Composite	#1131
			Proxy	#1129 #1130
			TemplateMethod	#1135
			Singleton	#1126 #1127
			Observer	#1128
			Iterator	#1125
			Facade	#1133
			Interpreter	#1132
		Columbus3.5	AdapterObject	#29 #30 #31 #32 #33 #34 #35 #36 #37
			State	#38
			TemplateMethod	#39 #40 #41
		Maisa0.5	AdapterObject	#6 #7 #8 #9 #10
			Prototype	#12 #13 #14 #15 #16 #17
			Proxy	#18 #19 #20 #21 #22 #23 #24 #25 #26 #27 #28
			Builder	#11
			AdapterClass	#2 #3 #4 #5

Figura 2.3: DEEBEE: risultati della ricerca.

Analisi dei dati statistici associati alle valutazioni delle istanze

DEEBEE offre la possibilità di accedere a dati statistici relativi alle istanze di DP associate ad una qualsiasi voce (strumento di DPD, sistema analizzato, ecc...) selezionabile nella maschera di ricerca (vedi figura 2.4).
Basandosi sui dati raccolti dalle valutazioni degli utenti, è possibile ricostruire una serie di dati statistici che consentono di trarre delle conclusioni sul grado di accuratezza dei singoli riconoscimenti. In particolare, è possibile consultare i valori relativi alla media, alla mediana, alla deviazione e ai valori minimi e massimi relativi alle

percentuali di correttezza e completezza attribuite all'istanza da parte dagli utenti dell'applicazione.
Le informazioni sono rilevanti e pertinenti ma purtroppo anche di difficile lettura e interpretazione. Una funzionalità secondaria, accedibile dalla pagina appena descritta, consente di confrontare coppie di risultati ottenute dall'analisi di uno stesso sistema (vedi figura 2.5). Il confronto tra singole istanze è basato sulla ricerca dell'identità assoluta dei valori che caratterizzano un'istanza. L'algoritmo di confronto adottato non contempla similarità parziali, ed è perciò da considerare di limitata utilità.

Correctness

#1125	83.0%	24.04%	66.0%	100.0%	83.0%
#1126	100.0%	0.0%	100.0%	100.0%	100.0%
#1127	100.0%	0.0%	100.0%	100.0%	100.0%
#1128	83.0%	24.04%	66.0%	100.0%	83.0%
#1129	66.5%	47.38%	33.0%	100.0%	66.5%
#1130	49.5%	23.33%	33.0%	66.0%	49.5%
Mean	**77.05%**	**15.11%**	**66.36%**	**87.73%**	**77.05%**
Deviation	**26.21%**	**15.99%**	**33.4%**	**22.68%**	**26.21%**
Min	**16.5%**	**0.0%**	**0.0%**	**33.0%**	**16.5%**
Max	**100.0%**	**47.38%**	**100.0%**	**100.0%**	**100.0%**
Median	**83.0%**	**23.33%**	**66.0%**	**100.0%**	**83.0%**

Summary

Number of pattern instances:	11
Number of evaluated pattern instances:	11
Number of pattern instances above the threshold:	9
Precision:	81.82%
Total number of pattern instances:	29
Total number of evaluated pattern instances:	29
Total number of pattern instances above the threshold:	16
Recall:	56.25%

Figura 2.4: DEEBEE: dati statistici.

Visualizzazione e valutazione di un'istanza

DEEBEE consente di visualizzare singole istanze di design pattern (vedi figura 2.6). In seguito ad una ricerca, è possibile consultare le istanze di design pattern riportate nella tabella dei risultati. In alcuni casi, l'istanza è associata ad una serie di file sorgente, consultabili all'interno della stessa pagina. Sono inoltre riportati alcuni dati descrittivi che caratterizzano l'istanza (Tool, Software, Pattern). Se l'utente è registrato può esprimere due tipi di valutazione.

A = ReferenceC++/Human
B = ReferenceC++/Columbus3.5

A	#1152	#1153	#1154	#1155	#1156	#1157	#1158	#1159	#1160	#1173
	#1138	#1139	#1140	#1141	#1142	#1143	#1144	#1145	#1146	#1147
	#1148	#1149	#1150	#1151						
B	#1154	#1155	#1157	#1158	#1159	#1160	#1138	#1139	#1140	#1141
	#1145	#1147	#1150	#1173						
A - B	#1152	#1153	#1156	#1142	#1143	#1144	#1146	#1148	#1149	#1151
B - A										
A & B	#1154	#1155	#1157	#1158	#1159	#1160	#1138	#1139	#1140	#1141
	#1145	#1147	#1150	#1173						

Figura 2.5: DEEBEE: confronto tra due tool.

La prima è una valutazione di tipo strutturale, dove è necessario considerare il numero di componenti individuati rispetto al numero di componenti comunemente presenti nel design pattern dichiarato.
La seconda invece è una valutazione semantica, in cui è richiesto di stabilire la correttezza nell'attribuzione del ruolo alle componenti individuate nell'istanza. Le valutazione circa l'istanza, possono essere espresse rispettivamente su una scala a 3 e 4 valori e opzionalmente applicate a tutte le istanze "gemelle" identificate in altre analisi.
La valutazione di un'istanza, data l'assenza di informazioni più fruibili rispetto alle semplici righe di codice delle classi associate, risulta purtroppo faticosa e spesso non banale. L'analisi dei singoli componenti dell'istanza potrebbe anche richiedere una minima conoscenza del programma all'interno del quale è stata individuata, nonché una buona conoscenza del linguaggio di programmazione adottato. L'impossibilità di visionare più viste contemporaneamente, limita inoltre la capacità di trarre delle conclusioni corrette e ponderate. Tutte le valutazioni sono anonime e prive di giustificazione. L'unica opzione che consente uno scambio di opinioni e una critica più analitica relativa ai contenuti dell'istanza, è data dall'utilizzo di una funzionalità per l'aggiunta di commenti riportata in fondo alla pagina.

Conclusioni

L'applicazione presentata da Fulop et al. può essere considerata una buona soluzione al problema della definizione di un sistema di benchmark per tool di DPD, ma a mio modesto avviso, presenta ancora delle caratteristiche piuttosto immature o di scarsa utilità. Di seguito verranno elencate alcune, tra le più evidenti, di queste caratteristiche:

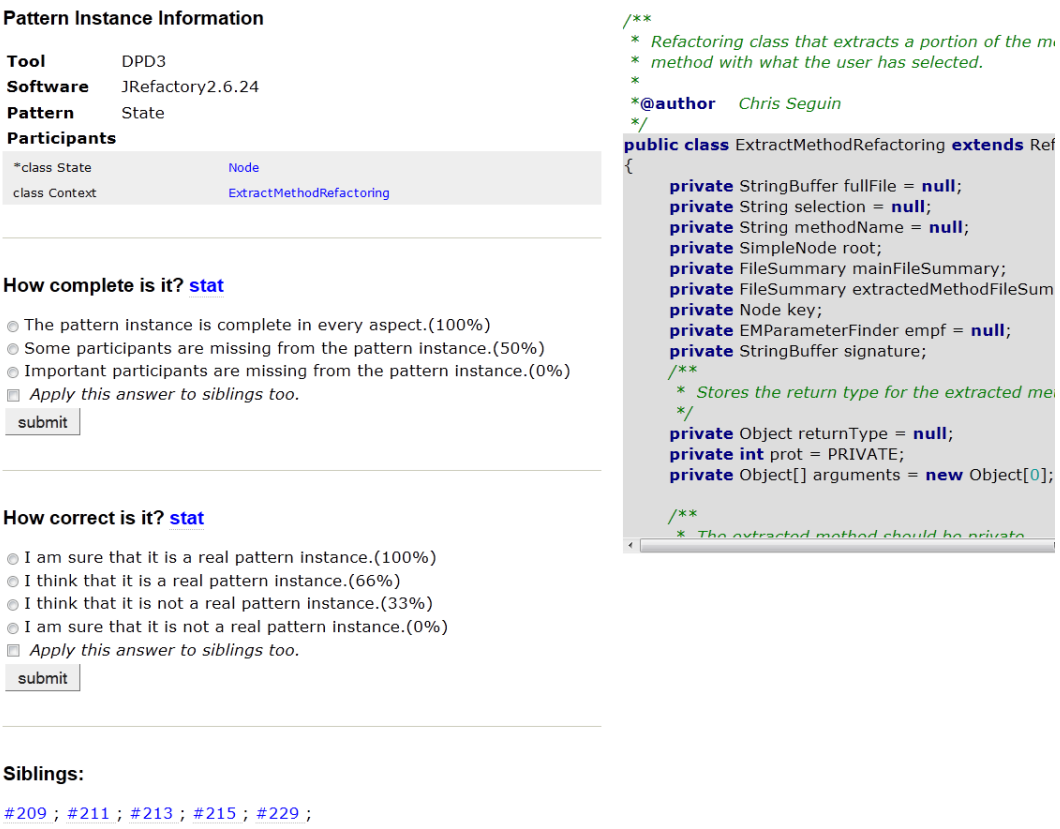

Figura 2.6: DEEBEE: visualizzazione di un'istanza.

- Ricerca: i risultati vengono presentati in modo elementare, confuso e poco informativo. Le informazioni di carattere qualitativo non sono integrate nei risultati di ricerca, ma sono riportate in una pagina secondaria difficilmente accedibile e di complessa interpretazione.

- Visualizzazione dell'istanza: i dati che caratterizzano l'istanza sono complessi, organizzati in modo non ottimale e, in casi non banali, non sufficienti a garantire una corretta valutazione della stessa. Le valutazioni sono anonime e non possono essere né discusse né contestate.

- Confronto: presenta dei risultati che non si discostano molto da quelli della funzione di ricerca. Appare limitata e di poca rilevanza per l'utente.

2.2 P-MARt

Un ulteriore contributo al confronto dei risultati prodotti da strumenti di DPD, è fornito dal progetto P-MARt [53]. Scopo del progetto è la costituzione di un ricco campione di istanze, verificate da utenti esperti, che possa fungere da riferimento per la misurazione del grado di qualità del risultato di uno strumento di DPD in termini di precision e recall. Tale campione di istanze è composto da un numero complessivo di 149 istanze di DP, individuate su un insieme di 9 sistemi distinti. Tutte le istanze presenti nel campione, sono state accuratamente catalogate e rese disponibili in formato XML e XLS.

Non sono comunque assenti eterogeneità nella strutturazione delle istanze e nel numero di ruoli adottati per la loro descrizione. Appare piuttosto evidente che l'insieme complessivo finale di istanze sia stato ottenuto dalla composizione di risultati differenti, elaborati da persone diverse per formazione e qualità di giudizio.
Valutare uno strumento sulla base di un confronto rispetto ad un modello poco omogeneo, non verificato da terzi e composto da un insieme di singole valutazioni personali appare, a mio modesto avviso, scientificamente poco corretto.
Nonostante questo giudizio, ritengo tuttavia che il contributo offerto da tale prodotto sia di notevole valore e possa essere ulteriormente valorizzato se inserito all'interno di un contesto critico aperto alle discussioni e alle valutazioni di una comunità di esperti.

Non sono attualmente note applicazioni scientificamente rilevanti di tale campione di dati.

Di seguito è riportato l'elenco dei sistemi analizzati per la creazione del catalogo P-MARt:

- QuickUML 2001.
- Lexi 0.1.1 alpha.
- JRefactory 2.6.24.
- Netbeans 1.0.x.
- JUnit 3.7.
- JHotDraw 5.1.
- MapperXML 1.9.7.
- Nutch 0.4.
- PMD 1.8.

Capitolo 3

Un modello per la rappresentazione di design pattern

In questo capitolo verrà presentato nel dettaglio un nuovo meta-modello per la rappresentazione dei DP. Tale meta-modello è stato ideato e sviluppato nell'ambito del progetto discusso in questa tesi ed è attualmente utilizzato come schema di riferimento per la gestione e l'interscambio delle definizioni e delle istanze di DP registrate nella piattaforma DPB.
A seguire, verranno esaminati e valutati criticamente altri meta-modelli presentati in letteratura aventi scopi analoghi al meta-modello realizzato. Per ognuno di essi, verrà descritta una possibile soluzione di integrazione rispetto al meta-modello adottato nel progetto DPB.

3.1 Rappresentazione dei design pattern in DPB

Al fine di garantire coerenza tra i dati presenti sulla piattaforma e supportare attività come il confronto e la valutazione di istanze (vedi i capitoli 4 e 5 per maggiori dettagli), è stato necessario implementare un meta-modello che fosse in grado di soddisfare i seguenti requisiti:

- Minimo sforzo per capire come definire una nuova istanza/definizione di DP.
- Rappresentazione compatta (per rendere efficiente archiviazione ed elaborazione dei dati).
- Supporto per istanze di DP aventi ruoli multi-valore.
- Sufficiente flessibilità per supportare qualsiasi tipo di definizione di DP.

Poiché nessuno dei meta-modelli attualmente documentati in letteratura (vedi sezione 3.2) è stato in grado di soddisfare tutti questi obiettivi, è stato necessario creare un nuovo meta-modello, partendo da alcune idee iniziali descritte in un articolo di Arcelli et al. [8].

Le sezioni seguenti forniranno una descrizione analitica del modello.

3.1.1 Panoramica del modello

La descrizione del modello adottato nel progetto DPB sarà suddivisa nelle tre parti identificate nel diagramma rappresentato in figura 3.1.

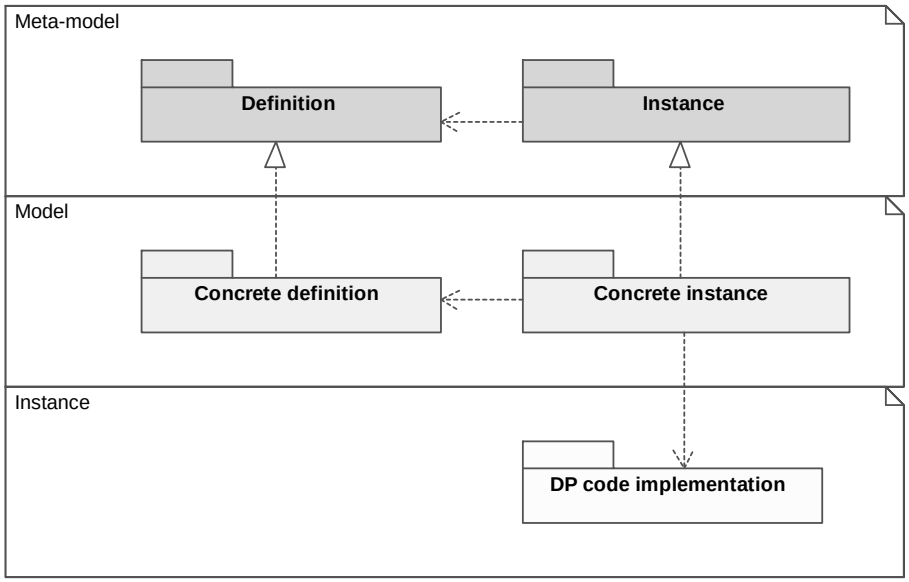

Figura 3.1: Composizione del modello utilizzato dall'applicazione DPB

Il livello più alto (*Meta-model*) contiene la rappresentazione astratta di una generica definizione di DP (*Definition*) e di una sua generica istanza (*Instance*).

Il secondo livello (*Model*) contiene la rappresentazione di una specifica definizione di DP (*Concrete definition*) e di una sua specifica istanza (*Concrete instance*). La struttura di entrambi gli elementi contenuti in questo livello è conforme all'insieme di regole e vincoli definiti nel livello soprastante. Ogni DP supportato dalla piattaforma, possiede una propria definizione, e qualsiasi istanza di DP deve attenersi alle regole strutturali definite dalla definizione ad essa associata.

L'ultimo livello (*Instance*) contiene l'implementazione concreta (*DP code implementation*) dell'istanza di DP contenuta nel livello soprastante (*Concrete instance*). L'elemento in esso contenuto non è parte del modello, ma descrive la realtà che il modello intende rappresentare.

Se coerentemente definita e correttamente codificata (vedi appendice A per ulteriori informazioni sul formato di codifica), un'istanza di DP (*Concrete instance* in figura 3.1) può essere inserita all'interno di un rapporto d'analisi sottoponibile alla piattaforma da un qualsiasi utente registrato.

Una volta caricata sulla piattaforma, un'istanza di DP potrà essere visualizzata, valutata e confrontata con altre istanze associate alla stessa definizione. La percentuale di istanze corrette, associate a rapporti riconducibili allo stesso strumento di DPD, fornirà un elemento quantitativo per determinare la qualità dei risultati prodotti dallo strumento stesso.

Agli utenti che desiderino definire un'istanza pienamente compatibile con la piattaforma, è richiesta la sola analisi della *Definition* associata al DP prescelto e la conoscenza delle semplici regole strutturali specificate nel modulo *Instance*. Per la creazione di una nuova definizione di DP, è necessario invece fare riferimento alle specifiche contenute nel modulo *Definition* e ai principi guida descritti in 3.1.6.

Nelle pagine seguenti, ciascuno degli elementi sopra accennati sarà descritto in maggiore dettaglio.

3.1.2 Meta-modello

Il livello *Meta-model*, rappresentato nella figura 3.2, può essere suddiviso in due componenti: *Definition* e *Instance*.

Definition è la parte del meta-modello utilizzata per specificare le caratteristiche strutturali di una qualsiasi definizione di DP, a sua volta struttura di riferimento per la costruzione di un'istanza di DP. Gli elementi che la costituiscono sono `LevelDef` e `RoleDef`. Ogni `LevelDef` può possedere una o più associazioni ad elementi di tipo `RoleDef`. `RoleDef` descrive il ruolo di una classe nel contesto del DP, mentre i livelli sono necessari a definire la struttura gerarchica ad albero secondo la quale tali ruoli sono organizzati.

Instance è la parte del meta-modello utilizzata per creare una rappresentazione di una generica realizzazione di una data definizione nel contesto di un determinato programma software. La strutturazione del modello risultante dal meta-modello in esame è ottenuta attraverso un'opportuna correlazione tra elementi di tipo `Level` e `LevelInstance`. Ogni elemento di tipo `Level`/`Role` è associato in maniera esplicita all'elemento `LevelDef`/`RoleDef` corrispondente presente nella definizione a cui è conforme.

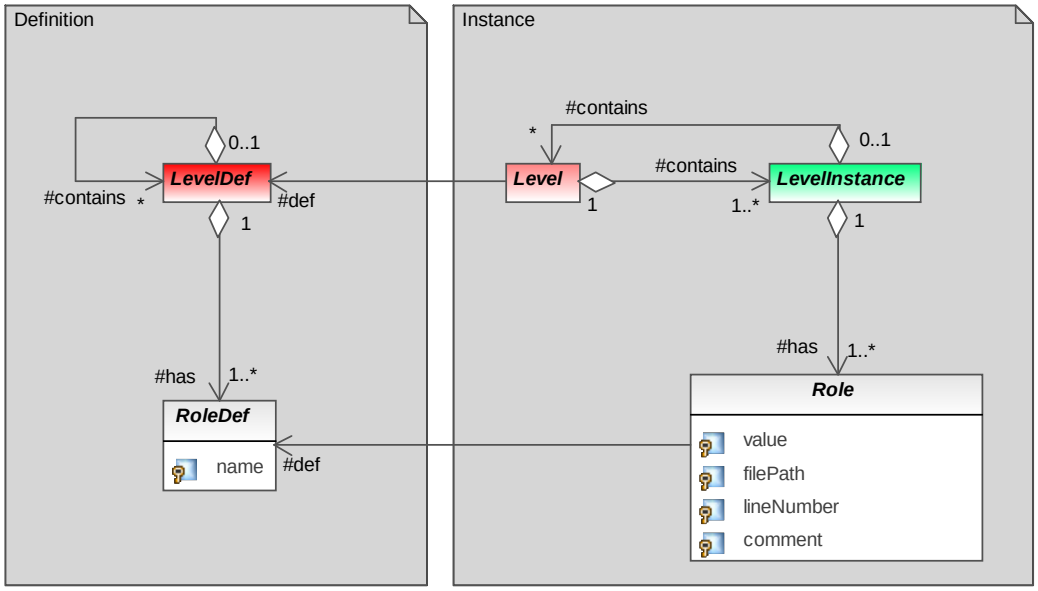

Figura 3.2: Modello DPB: meta-modello

3.1.3 Modello

Il livello model può essere suddiviso in due componenti: *Concrete definition* e *Concrete instance*.

La *Concrete definition* rappresenta una possibile realizzazione di *Definition*. Come accennato in precedenza, esiste un modello per ognuno dei DP supportati dalla piattaforma. Ogni `RoleDef` è caratterizzato da un nome che ne chiarisce lo scopo e dalla tipologia di costrutto alla quale fa riferimento. Tale nome, nel caso dei DP definiti da Gamma et al.[43], corrisponde alla denominazione utilizzata per identificare i *participants*. Le tipologie di ruoli supportate dalla piattaforma DPB sono: classe, metodo e attributo.

La *Concrete instance* rappresenta la realizzazione di *Instance* rispetto ad una specifica *Concrete definition*. La struttura ad albero del modello, composta da una serie di elementi di tipo `Level`, è perfettamente speculare alla quella specificata nella corrispondente *Concrete definition*. Ogni `Level` può essere declinato in uno o più `LevelInstance` a cui viene associata almeno un'istanza di `Role` per ogni `RoleDef` associato al `LevelDef` corrispondente. Ogni `Role` è caratterizzato dalle informazioni associate al `RoleDef` corrispondente e dai seguenti attributi:

- value: riferimento univoco dell'elemento di codice a cui è stato attribuito quel ruolo.

- filePath e lineNumber: percorso del file e numero di linea necessari alla localizzazione dell'elemento di codice all'interno della base di codice analizzata (ospitata sul server SVN di DPB).
- comment: un commento utile a giustificare la scelta di attribuzione del ruolo o per specificare il grado di confidenza dell'attribuzione.

La figura 3.3 mostra la correlazione tra un esempio di definizione concreta e una sua relativa istanza. Al fine di agevolare la lettura del diagramma, sono stati omessi alcuni attributi di `Role` e sono state aggiunte delle etichette agli elementi `Level`, `LevelDef` e `LevelInstance`.

Una lista completa di definizioni attualmente supportati, è consultabile nella sezione Documentation della piattaforma [19].

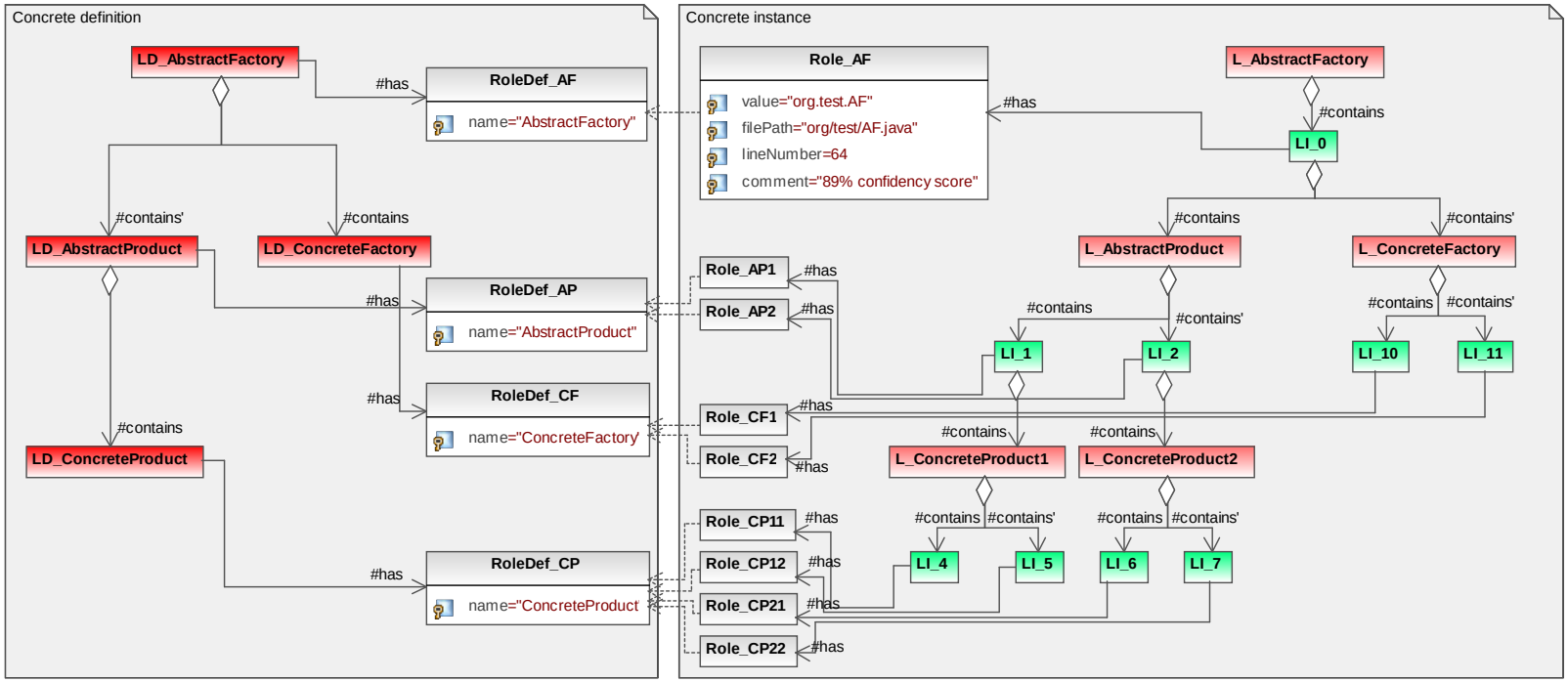

Figura 3.3: Modello DPB: modello

3.1.4 Istanza

Il livello *Instance* contiene l'implementazione concreta, in termini di codice sorgente, del DP modellato dall'elemento *Concrete instance*.

La figura 3.4 presenta la rappresentazione UML di una possibile implementazione associata all'istanza raffigurata nell'immagine 3.3.

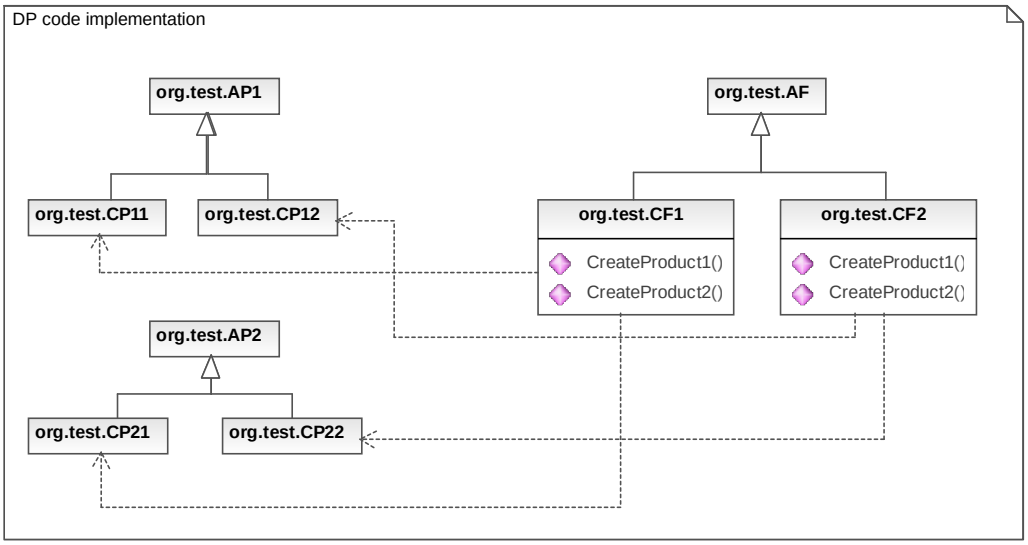

Figura 3.4: Modello DPB: esempio reale di istanza di DP.

3.1.5 Considerazioni sul modello

Rispetto ai requisiti stabiliti nella sezione 3.1, il modello descritto nei paragrafi precedenti garantisce tutte e quattro le proprietà elencate all'inizio del capitolo:

- *Minimo sforzo per capire come definire una nuova istanza/definizione di DP.*

  La definizione di una nuova istanza richiede la sola analisi e comprensione della *Concrete definition* associato alla definizione prescelta. La comprensione del modello può essere facilitata da una rappresentazione in forma di diagramma UML e da un esempio XML.
  La formulazione di una nuova definizione di DP risulta essere di media complessità e richiede sia l'analisi delle specifiche della *Definition* sia la consultazione di una serie di linee guida che consentano di comprendere la logica di definizione della gerarchia dei livelli (vedi sezione 3.1.6).

- *Rappresentazione compatta.*

  La quantità di informazioni necessarie per specificare un'istanza è stata ridotta al minimo, in modo da garantire una buona efficienza in fase di analisi ma al tempo stesso conservare un numero di dati sufficiente per supportare la fase di valutazione dell'istanza. In particolare si è scelto di fare a meno di una specifica accurata e completa degli elementi del codice, garantita dai vari modelli descritti nella sezione 3.3 e parzialmente implementata dal modello presentato nella sezione 3.2.1, optando per una strategia di localizzazione basata su semplici coppie nome-valore (vedi attributi di `Role` nella sezione 3.1.3).

- *Supporto per istanze di DP aventi ruoli multi-valore.*

  Ogni `Level` può essere istanziato in uno o più `LevelInstance`. Questa scomposizione garantisce la possibilità di definire sotto-alberi distinti per ogni livello presente nella struttura ad albero dell'istanza specificata, permettendo di creare collezioni multiple di ruoli dello stesso tipo con valori diversi. Inoltre ogni `LevelInstance` può essere associato ad un numero indefinito di assegnamenti dello stesso `Role`.

- *Sufficiente flessibilità per supportare qualsiasi tipo di definizione di DP.*

  L'alto grado di astrazione della *Definition* garantisce la possibilità di specificare qualsiasi definizione di DP concepibile. L'unica limitazione è data dal vincolo di organizzare la struttura della definizione in modo gerarchico, associando ciascun ruolo ad un elemento di tipo `LevelDef`.

3.1.6 Linee guida per la strutturazione delle definizioni

Nella presente sotto-sezione verranno esaminati i principi di riferimento adottati nella definizione delle definizioni di DP.

1. **Principio di molteplicità**: Dati due livelli A e B, a cui sono associati rispettivamente gli insiemi di ruoli $A_1, A_2, .., A_n$ e $B_1, B_2, .., B_n$, si può affermare che B è sotto-livello di A se (e solo se) per ogni istanza di un qualsiasi ruolo associato al livello A, esiste sempre almeno una istanza di ognuno dei ruoli associati al livello B. In altre parole, il rapporto di molteplicità tra il numero di istanze di un qualsiasi ruolo A_i (appartenente ad A) e un qualsiasi ruolo B_j (appartenente a B) è sempre pari a 1:1 o 1:molti. Se ad esempio esaminiamo il DP Abstract Factory rappresentato in figura 3.3, possiamo osservare che il livello `LD_ConcreteFactory` è sotto-livello di `LD_AbstractFactory`, perché ad ogni istanza del ruolo `RoleDef_AF` possono corrispondere una o più istanze del ruolo `RoleDef_CF`.

2. **Principio di accoppiamento**: Due ruoli A_1 e A_2 sono associati allo stesso livello, se ogni volta che si presenta un elemento corrispondente al ruolo A_1 è

possibile osservare con certezza anche uno e un solo elemento che corrisponde al ruolo A_2.

L'applicazione dei precedenti principi consente di produrre facilmente e in piena autonomia una nuova definizione di DP.

3.2 Modelli correlati per la rappresentazione dei design pattern

Nella sezione presente verrà analizzato l'unico meta-modello proposto in letteratura per la rappresentazione delle istanze di DP. Alla descrizione del meta-modello segue la specifica semi-formale di una possibile ipotesi di integrazione con il meta-modello descritto nella prima parte di questo capitolo (sezione 3.1).

3.2.1 DPDX

Il meta-modello DPDX, proposto nel 2010 da Kniesel et al.[66, 67], è un modello orientato alla rappresentazione dei design pattern che si compone delle seguenti parti:

- *Schema meta-model*: specifica le entità da adottare nella descrizione degli schemi dei DP.
- *Program element meta-model*: descrive gli elementi utilizzati per la rappresentazione del codice sorgente a cui fanno riferimento le istanze di DP riportate nei risultati.
- *Result meta-model*: raccoglie le entità necessarie a codificare le singole istanze di DP trovate all'interno del codice sorgente analizzato.

Tutte le istanze di *Result meta-model* facenti capo allo stesso DP devono essere conformi allo stesso schema definito da un'istanza dello *Schema meta-model*. Le istanze di *Result meta-model* mantengono i riferimenti al codice sorgente analizzato richiamando gli elementi di un'istanza di *Result meta-model*.

L'intento di questo meta-modello è quello di fornire una rappresentazione comune che possa fungere da standard di codifica per tutti gli strumenti che operano nel campo della DPD.

L'introduzione e l'adozione di un formato comune ai tool attivi in questo settore porterebbe notevoli vantaggi e consentirebbe di semplificare l'interazione tra vari strumenti attualmente mantenuti che si occupano di rilevamento, validazione, visualizzazione, comparazione e fusione di istanze di DP.

L'approccio adottato dagli sviluppatori di DPDX consiste nel definire una rappresentazione sufficientemente generica da soddisfare le esigenze di ogni tipologia di strumento presente nel settore.

Analisi della struttura

- *Schema meta-model* (figura 3.5): grazie a questo meta-modello è possibile specificare la struttura di una definizione di DP. Ogni definizione è vista come un'aggregazione piatta di ruoli aventi delle proprietà e correlate tra di loro attraverso tipologie di relazioni arbitrarie. Le istanze derivanti dalla realizzazione di tale meta-modello sono dei grafi composti da ruoli interconnessi.

- *Result meta-model* (figura 3.6): è stato concepito per rappresentare le istanze di DP riconosciute all'interno di un dato codice sorgente. Ogni istanza consiste in un'aggregazione di `RoleAssignement`, ai quali vengono assegnati dei valori specifici propri dell'istanza. Ogni elemento di tipo `RoleAssignement` fa riferimento al codice attraverso l'associazione ad un altro elemento di tipo `ProgramElement`.

- *Program element meta-model* (figura 3.7): definisce le entità fondamentali necessarie a rappresentare determinate parti di interesse del codice sorgente di un dato sistema software esistente. La scelta di elementi attualmente messi a disposizione dal meta-modello è decisamente limitata e non è certamente volta a rappresentare in maniera esaustiva le varie componenti di un sistema. La categorizzazione degli elementi è basata sulla distinzione in quattro differenti tipologie di elementi ed appare sufficientemente generica da supportare un vasto numero di linguaggi di programmazione.

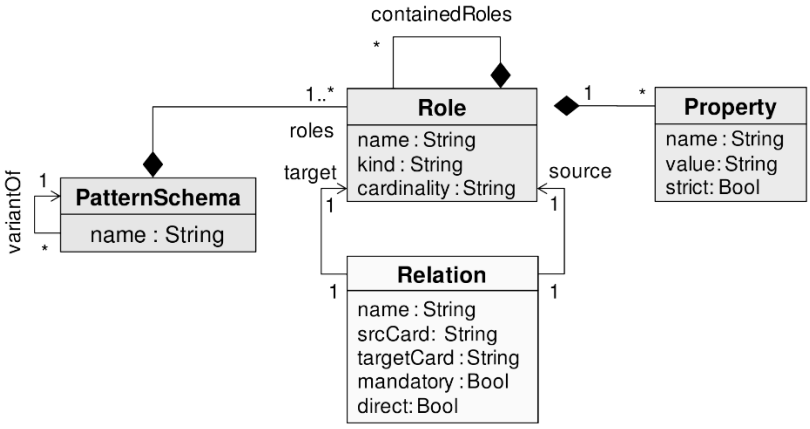

Figura 3.5: DPDX: Schema meta-model

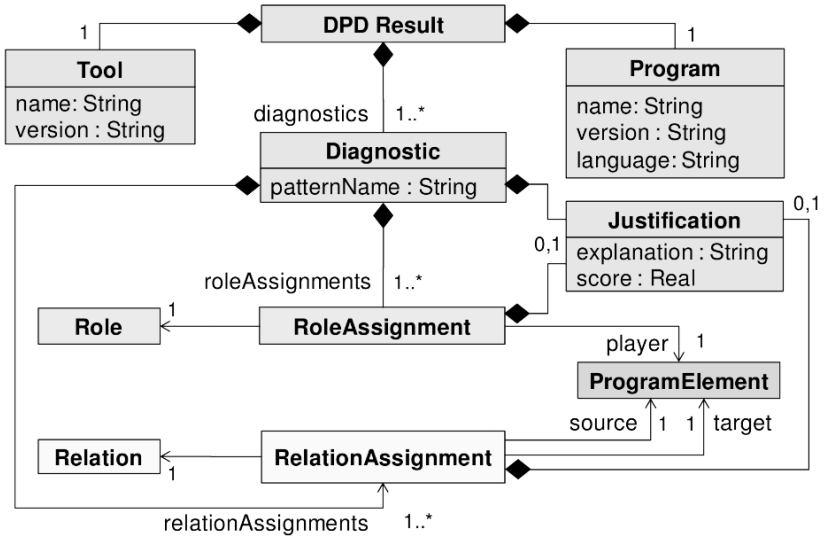

Figura 3.6: DPDX: Result meta-model

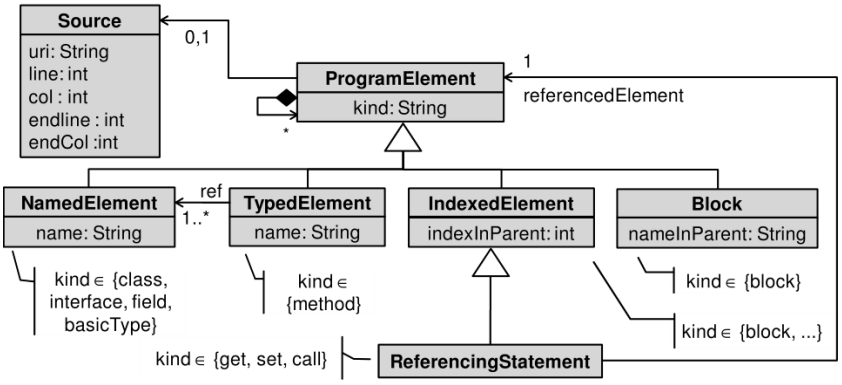

Figura 3.7: DPDX: Program element meta-model

Diffusione

Attualmente non sono note implementazioni del meta-modello DPDX.

Documentazione

Il meta-modello DPDX è stato documentato in un technical report di Günter Kniesel et al. [68].

Considerazioni finali

DPDX promette di diventare il prossimo standard de-facto per la rappresentazione dei dati prodotti e consumati dai tool concepiti nell'ambito della DPD. Al momento però, a causa dell'assenza di una documentazione completa e formale del modello, risulta piuttosto difficile trarre ulteriori conclusioni circa l'effettiva usabilità dello stesso.

Da quanto è emerso da una prima analisi, è comunque possibile osservare che il meta-modello in esame risulta essere piuttosto esteso e in diversi casi troppo generico per poter accomodare le esigenze applicative di alcuni strumenti. La codifica dei modelli derivanti da DPDX è poco leggibile e l'assenza della specifica di un insieme condiviso di *Schema meta-model*, non consente di rendere realmente interoperabili i modelli generati a partire dalla sua definizione. Inoltre, la scelta di specificare le definizioni dei DP utilizzando il formato XSD, impedisce agli autori di nuove definizioni di verificare la correttezza formale delle proprie soluzioni in modo automatico.

3.2.2 Possibilità di integrazione con DPB

Per integrare DPDX con il modello di DPB è possibile seguire due strategie: eliminare il concetto di livello dal modello DPB o creare un'estensione a DPDX introducendo nuovi elementi che permettano di ottenere una strutturazione gerarchica equivalente a quella prevista in DPB. Dal momento che il concetto di livello consente di ottenere una maggiore espressività e una strutturazione più ricca delle istanze, verrà di seguito illustrato uno scenario di integrazione che risponde alla seconda ipotesi presentata.

La soluzione proposta in questa sede, illustrata nel diagramma 3.8, prevede l'introduzione di due nuovi elementi (`DPDX_Level` e `DPDX_LevelInstance`) e di una nuova classe di elementi (`PatternElement`) che permette di astrarre gli elementi `Role`, `DPDX_Level` e `ProgramElement`. Mentre le prime due entità rappresentano delle semplici estensioni, `PatternElement` introduce una modifica sostanziale, ma

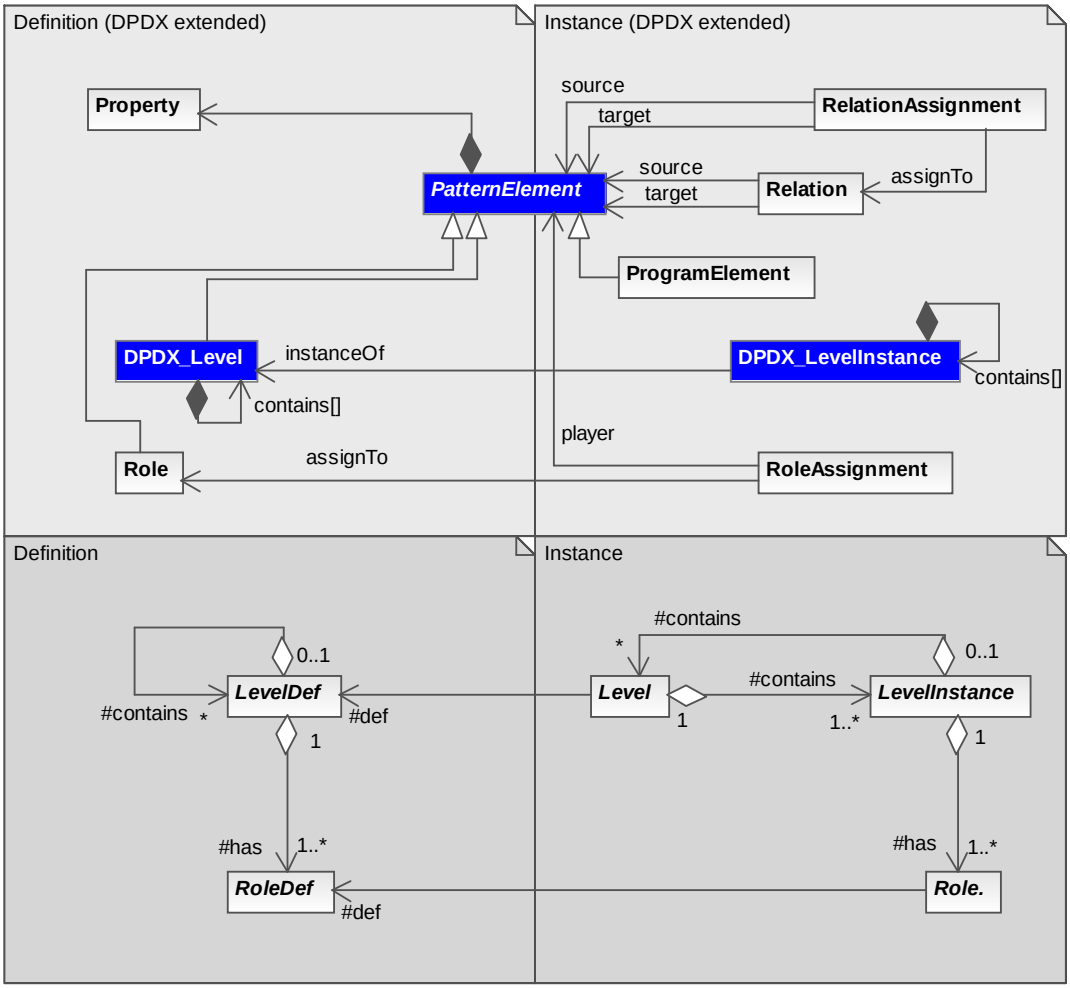

Figura 3.8: Integrazione DPDX-DPB: confronto tra meta-modello esteso DPDX e meta-modello DPB. Le entità evidenziate in blu sono state introdotte ai fini dell'integrazione.

necessaria, al meta-modello. L'obiettivo finale di questo arricchimento è da attribuire all'esigenza di rendere più astratto il significato delle entità `Relation` e `RelationAssignment`, al fine di consentire la creazione di relazioni tra coppie di entità `DPDX_Level/Role` e `DPDX_LevelInstance/RoleAssignment`. La somma delle modifiche descritte consente di ottenere istanze strutturate su più livelli, ottenendo risultati del tutto equivalenti alle istanze definibili a partire dal meta-modello DPB.

La figura 3.9 mostra un confronto tra due rappresentazioni di una stessa istanza, modellate secondo il meta-modello DPB e la nuova estensione di DPDX (di seguito chiamata *DPDX-esteso*) appena discussa. Le entità arancioni rappresentano livelli o istanze di livello, mentre quelle verdi rappresentano ruoli o istanze di ruolo.

Questo tipo di integrazione tra i meta-modelli DPB e DPDX, consentirebbe di creare un repository comune di definizioni in formato DPDX-esteso per i 23 pattern definiti da Gamma et al.[43]. Tale repository, unito ad un insieme di regole di conversione da un modello DPDX-esteso ad un modello DPB, garantirebbe la compatibilità della piattaforma con tutte le istanze specificate nel formato DPDX-esteso aderenti ad uno degli schemi comuni indicati nel repository.

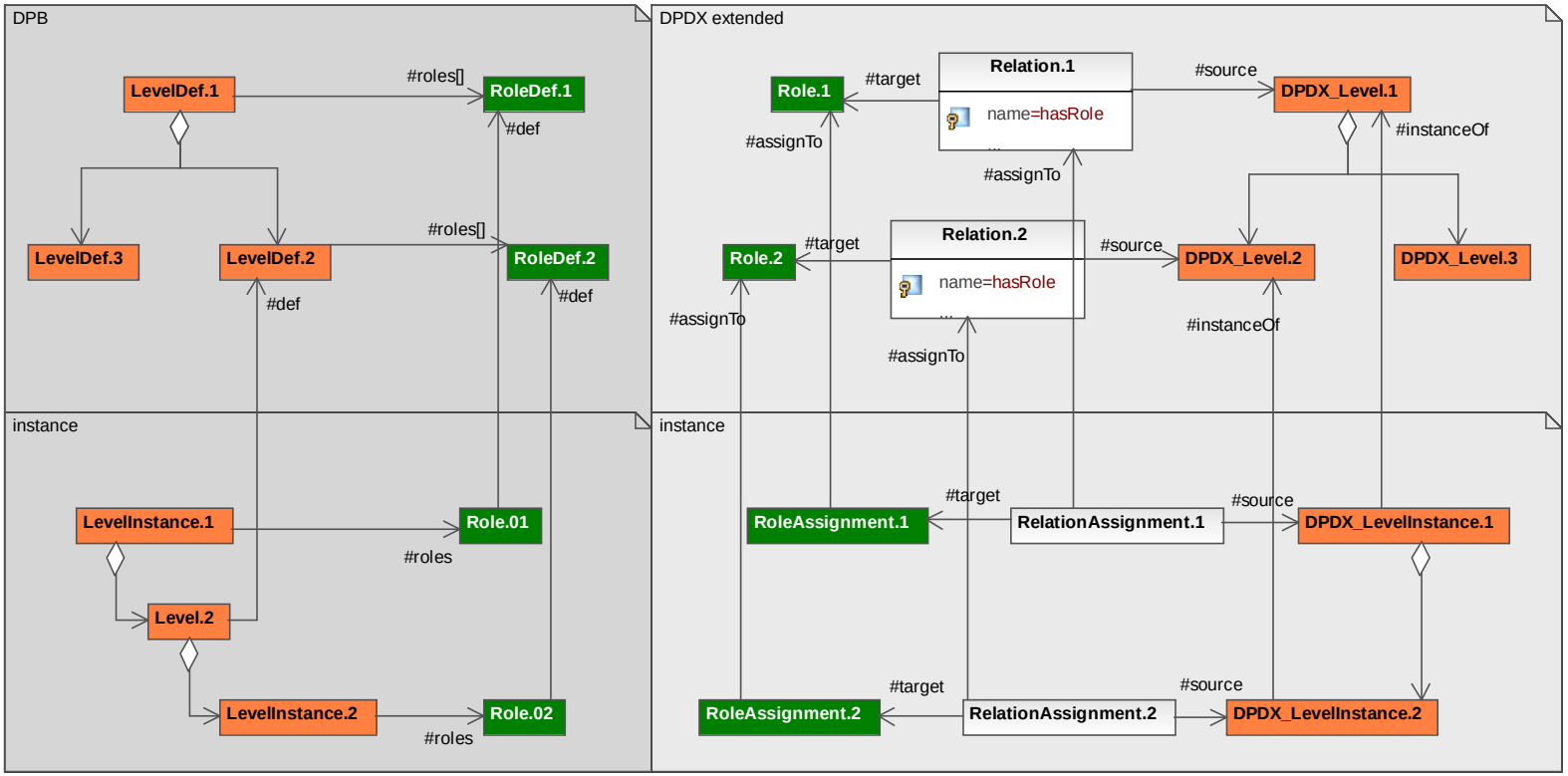

Figura 3.9: Integrazione DPDX-DPB: confronto tra istanza basata sul meta-modello esteso DPDX e istanza basata sul meta-modello DPB

3.3 Modelli correlati per la reverse engineering

Nelle sezioni seguenti, verranno analizzati alcuni modelli adottati nel campo della reverse engineering. In coda alle singole descrizioni, verrà presentata una possibile ipotesi di integrazione con il meta-modello definito nel contesto di questo progetto.

3.3.1 KDM

KDM (Knowledge Discovery Metamodel)[47] è la specifica di un meta-modello definito dall'Object Management Group (OMG)[50] definita tramite MOF[49]. La specifica si compone di vari meta-modelli che, opportunamente combinati, sono in grado di fornire una rappresentazione completa e accurata di un sistema software esistente, indipendentemente dal suo linguaggio di programmazione e dalle tecnologie adottate in fase di realizzazione.

La collezione di meta-modelli che compongono KDM può essere vista come uno schema di rappresentazione omni-comprensivo ed estensibile volto a descrivere gli aspetti chiave della conoscenza associata alle varie sfaccettature di un sistema software enterprise.

L'obiettivo di KDM consiste nel definire una rappresentazione condivisa e completa, atta a garantire l'interoperabilità fra strumenti differenti e in grado di supportare in maniera efficace ed efficiente attività di mantenimento, evoluzione, valutazione e modernizzazione del software.

L'adozione su larga scala di KDM consentirebbe di abbandonare il codice sorgente come unico elemento di riferimento oggettivamente condiviso e adottare come base di conoscenza comune una rappresentazione di più alto livello indipendente dalle tecnologie adottate e dal contesto operativo in uso. Una tale rappresentazione consentirebbe di praticare una separazione logica netta tra componenti analizzati e componenti interpretati, offrendo agli autori di strumenti di analisi la possibilità di demandare completamente l'attività di interpretazione a strumenti di terze parti e focalizzare i propri sforzi su attività a più alto contenuto creativo.

KDM facilita inoltre i processi di analisi basati su attività incrementali, dove una rappresentazione iniziale derivata dall'ispezione automatica del codice viene gradualmente arricchita di dettagli modellistici derivati da altri strumenti o da utenti esperti che interagiscono con il modello.

Analisi della struttura

KDM si presenta come un modello composto da 9 meta-modelli (o package) suddivisi in 4 layer (vedi figura 3.10).

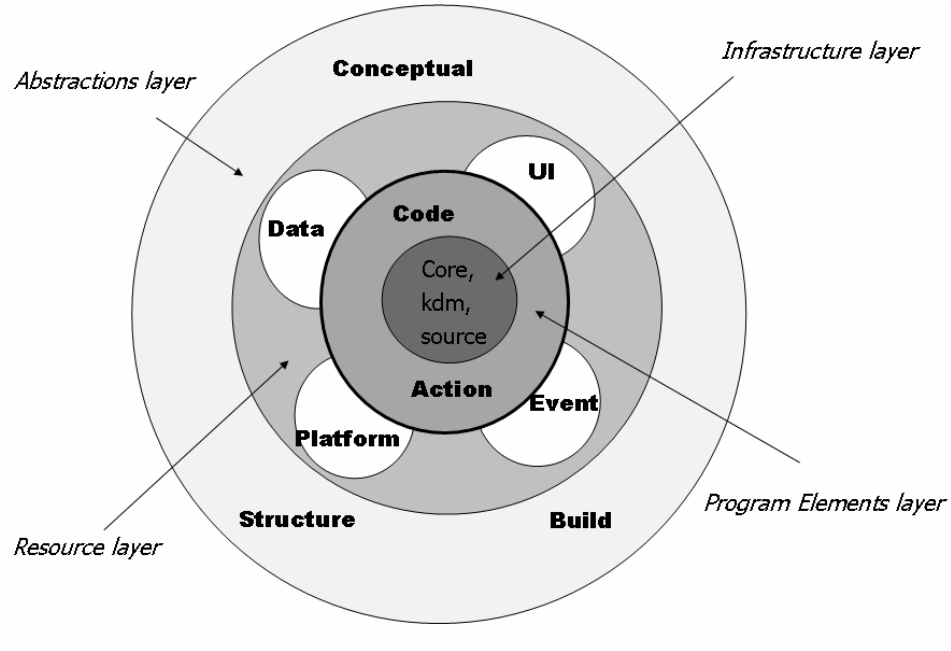

Figura 3.10: KDM: rappresentazione grafica del meta-modello

Ogni layer si pone ad un livello di astrazione via via crescente.

L'implementazione di un layer richiede l'adozione di tutti i layer sottostanti. Tutti i layer, ad esclusione del primo, sono opzionali e devono essere utilizzati a secondo del contesto e degli obiettivi dell'analisi.

Di seguito verranno descritti in maniera sommaria i 4 layer citati in precedenza e i rispettivi package:

- *Infrastructure layer*: Raccoglie i concetti fondamentali del modello KDM.
  - *Core*: definisce un insieme di tipologie di elementi, i vincoli ad essi associati e le relazioni che è possibile definire tra di essi. Ogni relazione è di tipo binario e rappresenta un'associazione semantica tra le entità che connette.
  - *Kdm*: definisce un certo insieme di elementi che nel loro insieme costituiscono il framework di ogni rappresentazione KDM.
  - *Source*: definisce le entità utilizzate per esplicitare i riferimenti di tracciabilità tra elementi KDM e gli artefatti corrispondenti presenti nel codice sorgente del sistema software analizzato.
- *Program elements layer*: Prevede la definizione di entità necessarie a descrivere i costrutti fondamentali comuni alla maggior parte dei linguaggi di

programmazione comunemente adottati nella codifica di un sistema software.

  - *Code*: definisce delle entità necessarie ad esprimere gli elementi e le relazioni strutturali presenti nel codice sorgente del sistema analizzato.

  - *Action*: definisce dei concetti utili alla descrizione delle caratteristiche comportamentali del sistema analizzato. In particolare, grazie a questo package, è possibile descrivere flussi di controllo e di dati tra coppie di elementi che estendono il package *Code*.

- *Runtime resource layer*: Rappresenta l'ambiente operativo del sistema software analizzato.

  - *Platform*: definisce il vocabolario di base da utilizzare al fine di descrivere gli elementi che costituiscono l'ambiente operativo del sistema considerato (es: O.S., middleware, ecc...) e i flussi logici che si instaurano tra di essi in fase di runtime della piattaforma.

  - *UI*: definisce le entità che possono essere adottate per descrivere concetti relativi all'interfaccia utente di un software esistente.

  - *Event*: rappresenta la conoscenza associata agli eventi e alle transizioni di stato osservabili durante la fase di runtime del software analizzato.

  - *Data*: consente di descrivere artefatti utilizzati per la persistenza dei dati (quali Relational Database Management System (RDBMS), files strutturati, ecc...).

- *Abstractions layer*: Rappresenta vari tipi di astrazione di dominio e dell'applicazione.

  - *Conceptual*: definisce gli strumenti pratici necessari per descrivere regole di business e vincoli di dominio.

  - *Structure*: consente di descrivere la struttura logica di un sistema in termini di sotto-componenti logici/moduli, layer, e sottosistemi.

  - *Build*: rappresenta le caratteristiche delle risorse generate dal processo di build.

Diffusione

Al momento della stesura di questa tesi, l'applicazione di KDM è limitata ad un progetto della Eclipse Foundation [39] chiamato MoDisco[33]. Lo scarso interesse dimostrato dalla comunità nei confronti di KDM potrebbe essere dovuto alla complessità delle specifiche di dettaglio, alla sua relativamente giovane età o alla poca visibilità dedicata al progetto nell'ambito delle pubblicazioni scientifiche.

Documentazione

Il meta-modello KDM è ampiamente documentato nei documenti di specifica ufficiali [46] e in alcune pagine esplicative pubblicate sul sito dell'organizzazione [48].

Considerazioni finali

KDM è senza alcun dubbio il meta-modello più completo attualmente documentato in letteratura.

La possibilità di rappresentare un sistema software a diversi livelli di astrazione cogliendo solo alcuni aspetti e trascurandone altri, consente allo sviluppatore di muoversi in direzioni diverse a seconda delle proprie esigenze di analisi e di appoggiarsi ad altri tool complementari qualora fosse possibile evitare di re-implementare funzioni già sviluppate da terzi.

Il meta-modello fornisce un'ampia espressività ed è perfettamente estensibile.

3.3.2 FAMIX

FAMIX si presenta come un meta-modello, compatibile con MOF [49], avente lo scopo di rappresentare le caratteristiche statiche di un sistema software, indipendentemente dal linguaggio di programmazione adottato per la sua codifica. Il modello in questione supporta sia linguaggi procedurali sia object-oriented.

L'obiettivo principale di questo meta-modello è la definizione di uno schema sufficientemente completo da soddisfare le esigenze operative di strumenti di reverse engineering e attività di refactoring.

Le entità considerate sono in numero contenuto e lo schema risultante, pur essendo meno completo rispetto a quello previsto dalle specifiche di KDM, possiede molte caratteristiche interessanti che lo rendono adattabile, facilmente estensibile e ben inseribile in un contesto pratico.

Tra le caratteristiche che contraddistinguono questo schema citiamo:

- Supporto per l'ereditarietà multipla.
- Supporto di linguaggi tipizzati in modo statico e dinamico.
- Possibilità di rappresentare Eccezioni e Annotazioni.

Analisi della struttura

Al momento della stesura di questa tesi, sono state rese pubbliche le seguenti specifiche di FAMIX:

- FAMIX 2.2 [87].
- FAMIX 3.0 (beta) [88].

La prima è molto legata al paradigma object-oriented e si propone di individuare delle caratteristiche comuni a vari linguaggi e di aggregarli adottando un approccio bottom-up.

La seconda specifica invece, consente di astrarre maggiormente dai costrutti specifici adottati nei singoli linguaggi supportati, e mira a creare un meta-modello sufficientemente generico da comprendere un insieme più vasto ed eterogeneo di sistemi.

Per maggiori dettagli in merito alla descrizione di FAMIX 3.0, è possibile fare riferimento al diagramma UML presentato nelle pagine di documentazione del sito web ufficiale [88].

Diffusione

FAMIX è stato adottato nell'ambito del progetto open-source Moose [76, 75]. Tale progetto, nato nel 1996, vede coinvolti tra i suoi sviluppatori anche gli autori del meta-modello.

La piattaforma Moose permette di tradurre il codice sorgente del sistema analizzato, in un modello conforme a FAMIX. Tale modello potrà in seguito essere impiegato per effettuare analisi comparative su dati derivanti dal modello elaborato. Tra le funzionalità offerte dallo strumento citiamo: modellazione, computazione di metriche, visualizzazione, rilevazione di codice duplicato, ecc...

Per rendere più facile la consultazione dei dati generati dal framework, è stato scelto di implementare lo schema nel linguaggio Smalltalk. Tra i benefici derivati da questa scelta vi è la possibilità di navigare i contenuti dei modelli facendo uso di un linguaggio di query nativo ereditato da Smalltalk.

Documentazione

Il meta-modello FAMIX è parzialmente documentato nella tesi di dottorato dell'autore [89] e nelle pagine del sito del progetto Moose [44]. I dati del primo documento non sono aggiornati e fanno riferimento alla prima versione del meta-modello, mentre le informazioni presenti nel sito sono frammentarie e poco formalizzate.

Considerazioni finali

Il meta-modello in oggetto ha molte caratteristiche comuni al *Program elements layer* di KDM. Entrambi vengono utilizzati per descrivere le caratteristiche statiche di un sistema a livello di costrutti presenti nel codice sorgente.

KDM offre un livello di dettaglio maggiore e una chiara documentazione a cui far riferimento, mentre FAMIX risulta essere più contenuto nel numero delle entità e decisamente carente sotto il profilo documentale.

Nonostante ciò, FAMIX presenta molte caratteristiche interessanti e una struttura gerarchica lineare e di facile lettura. Se presentasse una documentazione più organica, potrebbe essere benissimo utilizzato come versione ridotta dello strato *Program elements layer* di KDM.

3.3.3 Dagstuhl

Dagstuhl, o Dagstuhl Middle Model (DMM), è un meta-modello, compatibile con GXL [59], sviluppato nel contesto di un progetto di ricerca internazionale nato nel 2001 nell'ambito del Dagstuhl Seminar on Interoperabiliy of Re-engineering Tools.

La caratteristica principale del modello, è quella di aver basato la propria definizione sulla distinzione, all'interno della rappresentazione di un sistema software, di tre macro-aree distinte e ben separate: riferimenti al codice sorgente (`SourceObject`), elementi del codice (`ModelObject`) e relazioni tra elementi del codice (`Relationship`).

L'obiettivo generale di Dagstuhl è quello di definire un meta-modello che astragga dai dettagli di basso livello (dettagli del codice specifici di un dato linguaggio di programmazione) e allo stesso tempo ignori caratteristiche comunemente contemplate da modelli di alto livello (es: componenti, pipes, filtri, ..).

Dagstuhl non nasce come un modello completo e omni-comprensivo, ma si propone piuttosto come un mezzo agile volto all'impiego pratico in strumenti di reverse engineering.

Tra gli elementi volontariamente ignorati da Dagstuhl troviamo:

- Riferimenti di chiamata a procedura/metodo.
- Assegnamenti e statement di controllo.
- Variabili locali.

Le conseguenze di queste scelte sono traducibili nelle seguenti affermazioni:

- Dagstuhl non possiede dettagli sufficienti a ricostruire il codice sorgente originale in seguito all'estrazione del modello.

- Dagstuhl non possiede dettagli sufficienti a generare un diagramma di flusso dati.

Analisi della struttura

Come precedentemente accennato, Dagstuhl pone l'accento sulla distinzione netta e senza sovrapposizioni tra elementi `SourceObject`, `ModelObject` e `Relationship` (vedi figura 3.11). La specifica completa del meta-modello prevede una ricca gerarchia di elementi che estendono ognuno dei tre elementi citati.

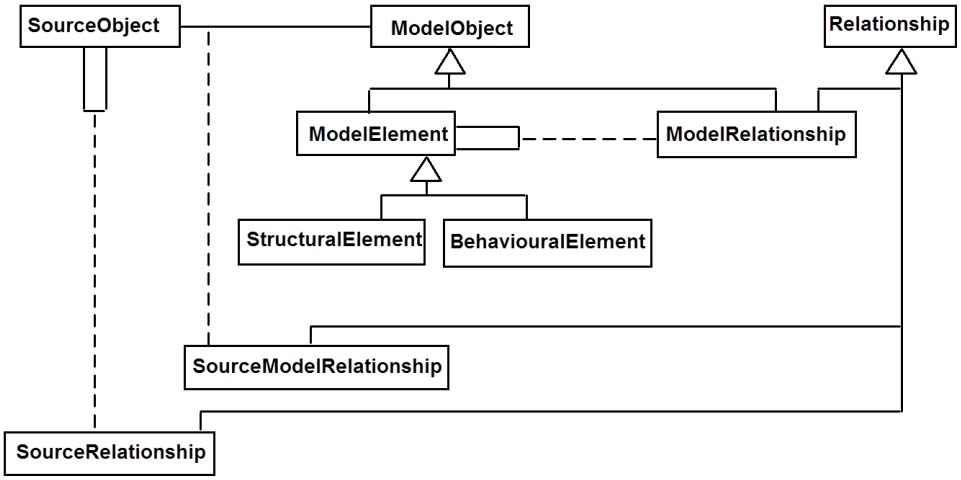

Figura 3.11: Dagstuhl: rappresentazione grafica del meta-modello

Il primo può essere interpretato come un semplice riferimento al codice sorgente. Grazie alle entità che lo estendono, è possibile contestualizzare in modo sufficientemente chiaro la rappresentazione di un `ModelObject` all'interno del codice sorgente.

`ModelObject` può rappresentare un qualsiasi tipo di costrutto presente nel codice di un sistema software scritto in un linguaggio di programmazione object-oriented o procedurale. Le entità rappresentabili possono variare da metodi e routine a classi, variabili e altri costrutti significativi nell'ottica di un'analisi statica di tipo strutturale o comportamentale.

`Relationship`, infine, codifica un discreto numero di relazioni che possono sussistere tra elementi di tipo `SourceObject-SourceObject`, `ModelObject-ModelObject` e `SourceObject-ModelObject`.

Gli elementi sommariamente descritti in questo paragrafo e previsti dalla specifica possono essere estesi e arricchiti a proprio piacimento. L'interoperabilità del meta-

modello è in ogni caso garantita dalla presenza di superclassi comuni che costituiscono un vocabolario comune a tutti i tool che supportano il meta-modello.

Ogni modello può essere codificato in qualsiasi formato che supporti i concetti di classe, istanza e associazioni tra istanze (es: GXL, TA, XMI, RDF).

Diffusione

Il meta-modello Dagstuhl è stato sperimentato nell'ambito di un progetto per il calcolo di metriche software a partire da codice sorgente Java [72].

Documentazione

Il meta-modello Dagstuhl è stato documentato nella tesi di dottorato dell'autore [71] e in un documento pubblicato da Timothy C. Lethbridge et al. [70]. Il progetto non dispone di alcun sito web di riferimento e di conseguenza è lecito supporre che sia stato abbandonato.

Considerazioni finali

Il meta-modello Dagstuhl si presenta come un'interessante variante semplificata di altri meta-modelli più completi quali KDM (*Program elements layer*) e FAMIX. Il suo punto di forza sta nell'estrema semplificazione dei concetti e nella loro chiara distinzione.

Rispetto a FAMIX, è possibile osservare:

- L'assenza di una gestione esplicita delle eccezioni (non sono presenti entità dedicate alla descrizione delle classi utilizzate per la gestione delle eccezioni e le entità di tipo `method` non referenziano alcuna classe esterna in caso lancio di un'eccezione).
- La presenza di entità in grado di gestire costrutti di tipo Collection o Enumeration (non trattati in modo esplicito in FAMIX).
- La presenza di una gerarchia più ampia e logicamente organizzata di relazioni fra entità.
- La presenza di un discreto numero di attributi aggiuntivi nell'entità `method` (es: isConstructor; isDestructor, isDinamicallyBound, isOverridable, ecc. . .).
- L'assenza di un'entità che rappresenti i namespace.

Purtroppo l'assenza di documentazione esaustiva e di implementazioni di riferimento riduce drasticamente il valore del modello. L'estensibilità dello schema è garantita dalla conformità alla specifica GXL.

3.3.4 Meta-modello Marple

Panoramica generale

Un ulteriore meta-modello per la rappresentazione delle entità di codice è stato sviluppato nel contesto del progetto Marple, sviluppato dal gruppo di ricerca Essere[3] presso l'Università di Milano-Bicocca.

Marple usa un meta-modello concepito per accomodare le esigenze pratiche di strumenti attivi nel settore della DPD, della software architecture reconstruction (SAR) e della anti-pattern detection.

La sua definizione è molto essenziale e orientata verso l'uso pratico. Al suo interno è possibile distinguere entità associate a componenti fisiche/logiche ed entità rappresentanti meta-informazioni derivate da processi di analisi più approfonditi.

Tra le meta-informazioni contemplate da MARPLE troviamo:

- Metriche (es: NOC, LOC, ecc...).
- Micro strutture (clues, EDP, micro-pattern) [7].

Analisi della struttura

Analizzando un sistema esistente è possibile distinguere tre tipi di informazioni (vedi figura 3.12):

- Informazioni relative alla struttura logica del progetto: composta da entità proprie del paradigma di programmazione ad oggetti (es: classi, metodi, attributi, ecc...).
- Informazioni relative alla struttura fisica del progetto: collezione di file e cartelle che contengono il codice sorgente del sistema analizzato.
- Meta-dati associati a entità logiche e/o fisiche: informazioni quali metriche e micro-architetture.

Tali informazioni sono state modellate in un unico meta-modello, che, in maniera semplice e pratica, mette in relazione gli uni con gli altri.

L'elemento centrale di questa rappresentazione è l'elemento `CodeEntity`. Tale elemento può essere specializzato in funzione della sua natura per meglio descrivere il proprio ruolo.

Le `CodeEntity` sono unità misurabili e, in quanto tali, possono essere associate ad una o più metriche aventi un proprio valore e utilizzabili ai fini dell'analisi. Le `CodeEntity` possono anche essere associate a diversi `BasicElement` e, secondo le esigenze, anche ad altri tipi di entità (es: `DesignPatternInstance`).

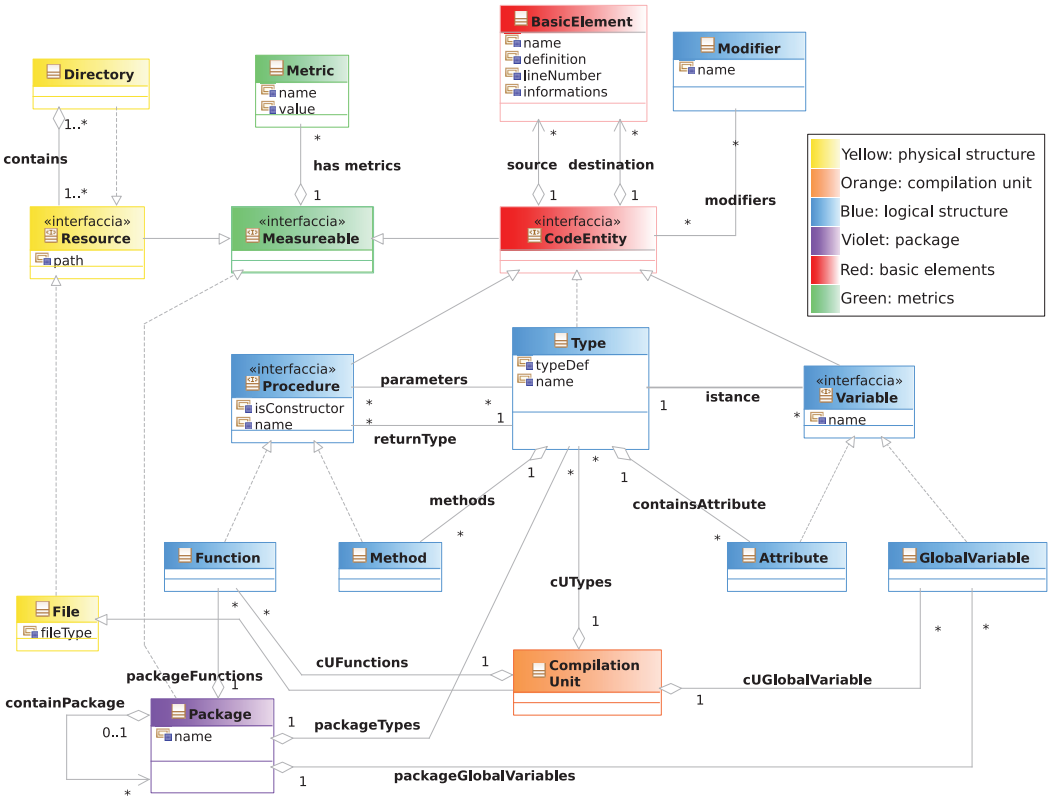

Figura 3.12: Marple: rappresentazione UML del meta-modello

Diffusione

Il meta-modello descritto è attualmente adottato nel contesto del progetto MARPLE [9], facente capo al gruppo di ricerca ESSeRE [3] dell'Università degli Studi di Milano-Bicocca [26].

Documentazione

Il meta-modello Marple è stato documentato in un articolo di Arcelli et al. [11].

Considerazioni finali

Il meta-modello MARPLE rappresenta un'ottima base di definizione per altri meta-modelli aventi obiettivi simili a quelli citati all'inizio della trattazione.

Risulta interessante l'inclusione all'interno del meta-modello di informazioni di carattere quantitativo e qualitativo che contribuiscono ad arricchire la rappresentazione globale del sistema e forniscono un valido supporto al processo di analisi. Inoltre risulta molto conveniente e immediata anche la navigazione dei contenuti rappresentati nel modello grazie alla presenza di sole associazioni bidirezionali e di utili relazioni di contenimento che permettono di mantenere il giusto livello di dettaglio in fase di consultazione.

Rispetto a FAMIX e Dagstuhl, è possibile osservare:

- Una maggiore precisione nella descrizione della gerarchia dei files analizzati (es: strutturazione delle directories).
- L'assenza di relazioni comunemente contemplate da altri meta-modelli (es: uses, calls, is-a, ecc. . .).
- L'assenza di un'entità che rappresenti i commenti.
- L'assenza di un'entità che rappresenti le annotazioni.

L'utilizzo del meta-modello è limitato al linguaggio di programmazione Java.

3.3.5 Altri modelli

Columbus [36]: Sviluppato da R. Ferenc nel 2002 e implementato nel sistema omonimo. La sua complessità, la mancanza di documentazione e il fatto che supporti solamente C/C++, lo rende obsoleto e decisamente poco interessante sotto il profilo dell'interoperabilità e dello scambio di informazioni nell'ambito della reverse engineering.

Datrix [60]: Sviluppato da Bell Canada Inc. nel 2000. Il meta-modello supporta C, C++ e Java ma, a causa della sua totale mancanza di documentazione e dell'assenza di una qualsiasi implementazione di riferimento, può essere considerato ormai obsoleto e di scarso interesse ai fini della presente trattazione.

Dynamix [45]: Un meta-modello, compatibile con MOF [49], creato per descrivere il comportamento dinamico di un'applicazione software. Di scarso rilievo nel contesto analizzato per la tipologia di informazioni rappresentate.

3.3.6 Possibilità di integrazione con DPB

Di seguito verranno analizzate alcune possibilità di integrazione tra il meta-modello DPB e gli altri meta-modelli esaminati nella presente sezione.

KDM

KDM è un meta-modello molto ricco ed estensibile. Analogamente a quanto discusso nella sezione 3.2.2, sarebbe possibile garantire un adeguato supporto agli strumenti basati su tale meta-modello definendone un'opportuna estensione e creando un repository di definizioni specificate secondo esso. Tale soluzione è illustrata in dettaglio nel diagramma rappresentato nella figura 3.13. Essa prevede l'introduzione di 8 nuove entità (rappresentate nella parte bassa dell'immagine) definite come specializzazioni di elementi presenti nel meta-modello originale di KDM. Tali entità sono:

- `KdmSourceRef` (specializzazione di `CodeItem`): rappresenta un riferimento ad una generica entità di codice (classe, metodo o attributo).
- `KdmLevelDef`, `KdmLevel` e `KdmLevelInstance` (specializzazioni di `Conceptual Container`): rappresentano degli aggregatori in grado di contenere entità generiche supportate dal meta-modello KDM.
- `KdmRole` e `KdmRoleDef` (specializzazioni di `ConceptualRole`): rappresentano descrittori associabili a elementi concettuali astratti supportati dal meta-modello KDM.
- `KdmLevelAssociation` e `KdmRoleAssociation` (specializzazioni di `Abstract ConceptualRelationship`): rappresentano relazioni generiche tra elementi concettuali astratti supportati dal meta-modello KDM.

Le entità sopra citate, sono delle pure estensioni del meta-modello che possono essere facilmente combinate per ottenere una rappresentazione equivalente al meta-modello DPB. Nella figura 3.14 si dimostra la fattibilità di tale tesi accostando il meta-modello composto dalle entità descritte al meta-modello adottato dalla piattaforma. Nello specifico, le entità `KdmLevelDef` e `KdmRoleDef` assumerebbero lo stesso ruolo delle entità `LevelDef` e `RoleDef`, mentre le entità `KdmLevel`, `KdmLevelInstance` e `KdmRole` consentirebbero di sostituire `Level`, `LevelInstance` e `Role`. Le relazioni tra gli elementi che costituiscono la definizione e quelli che rappresentano l'istanza, sono modellati attraverso l'utilizzo dell'entità `KdmRoleAssociation`. L'entità

`KdmSourceRef` è utilizzata per specificare il riferimento presente nel ruolo al codice sorgente.

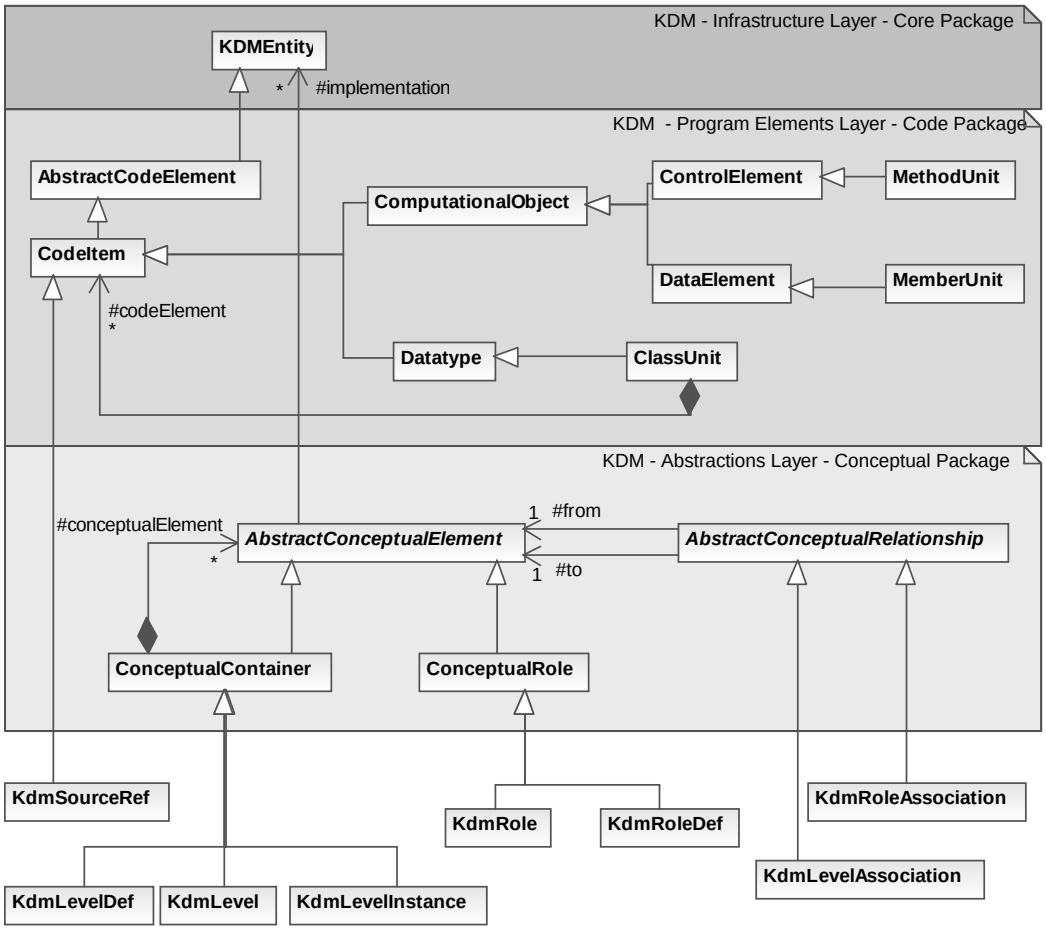

Figura 3.13: Integrazione KDM-DPB: estensione del modello KDM

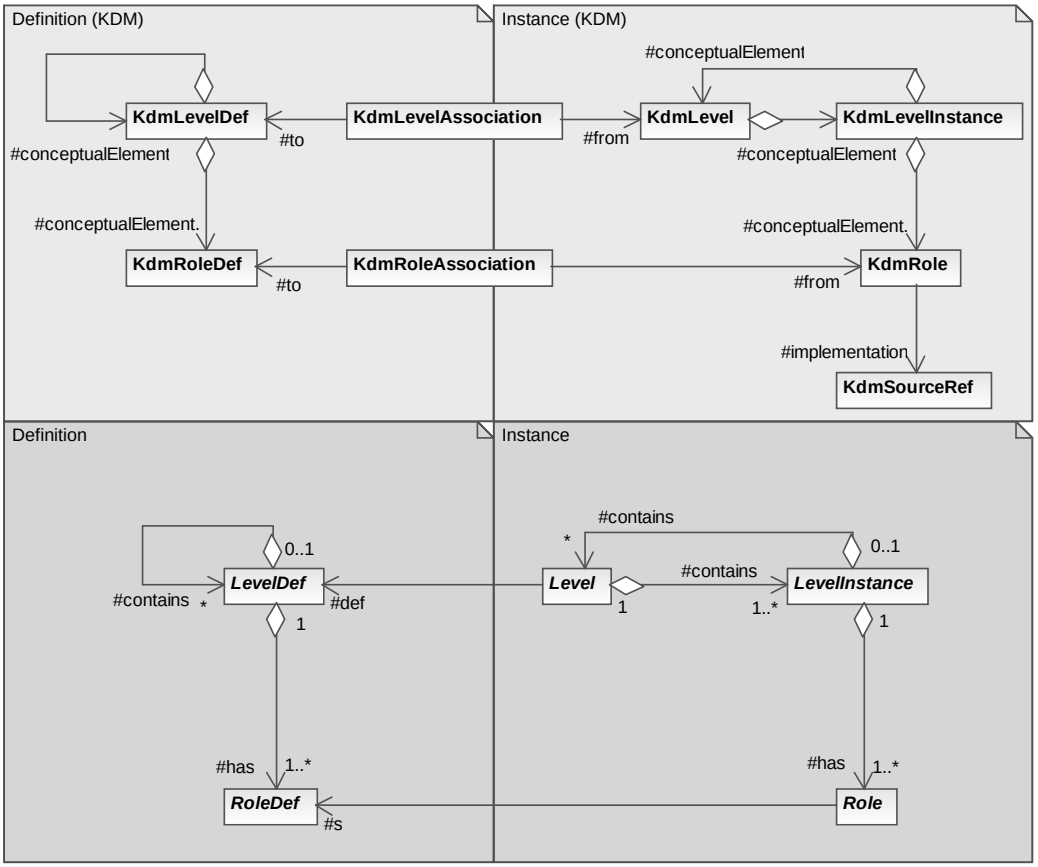

Figura 3.14: Integrazione KDM-DPB: confronto tra estensione del meta-modello KDM e meta-modello DPB

FAMIX, Marple, Dagstuhl

FAMIX, Marple e Dagstuhl sono dei modelli pratici utili alla rappresentazione del codice sorgente e dei costrutti presenti in esso. Ognuno dei meta-modelli citati presenta una superclasse astratta che rappresenta un generico elemento del codice. Tale elemento, se referenziato dall'entità `Role` presente nel meta-modello DPB, potrebbe fungere da anello di congiunzione tra ognuno dei tre meta-modelli trattati e il meta-modello DPB (vedi figura 3.15). L'estensione di FAMIX, Marple e Dagstuhl attraverso le modalità appena descritte, offrirebbe una capacità rappresentativa più ricca (e compatibile con la piattaforma DPB) a tutti quegli strumenti di DPD che adottino uno di questi meta-modelli per la rappresentazione interna della struttura del codice.

Di seguito è specificato, per ciascun meta-modello, il nome dell'elemento corrispondente all'entità `SourceCodeEntity` rappresentata nel diagramma 3.15:

- FAMIX: `Entity`.
- Marple: `CodeEntity`.
- Dagstuhl: `SourceObject`.

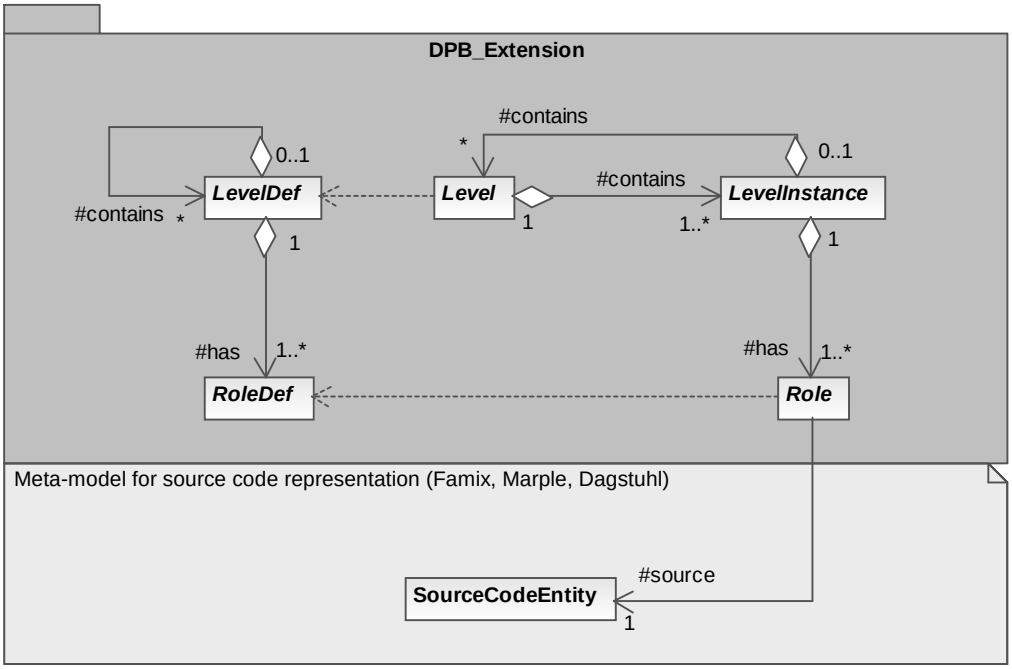

Figura 3.15: Integrazione FAMIX/Marple/Dagstuhl - DPB: estensione applicabile al modello FAMIX/Marple/Dagstuhl

Capitolo 4

Una piattaforma per la valutazione delle istanze di design pattern

In questo capitolo saranno illustrate le caratteristiche principali della piattaforma DPB. Nell'ordine, verranno analizzati: l'architettura tecnologica su cui essa è basata, il macro-processo che prevede e alcune delle più rilevanti funzionalità offerte.

Per sperimentare direttamente gli scenari descritti in questo capitolo, è possibile visitare il seguente indirizzo: `http://essere.disco.unimib.it/DPB`.

4.1 Tecnologie adottate

DPB è un'applicazione web accessibile via internet attraverso un qualsiasi browser che supporti le tecnologie HTML, JavaScript e Java applet. La maggior parte delle funzionalità erogate sono gestite dall'application server Glassfish[21], mentre altre funzionalità di minor rilievo (es: visualizzazione del codice sorgente), basate su tecnologia PHP, sono eseguite su Apache HTTP Server[40].
Il codice eseguito in ambiente Glassfish è stato strutturato secondo una classica architettura a tre livelli:

- Livello presentazione: contiene files Java e JSP. Ogni richiesta del client è gestita dal Web application framework JSF[77], un'implementazione Java-based del pattern architetturale MVC[41]. JSF associa una classe Java ad ogni pagina invocabile, specificandone i dettagli di presentazione in un file JSP separato contenente markup HTML, JSP e JSF. Tale soluzione garantisce una chiara separazione tra presentazione e logica di controllo, aumentando la leggibilità del

codice e riducendo al minimo la complessità e la quantità di codice necessaria ad implementare strutture di gestione dinamica delle informazioni.

- Livello logica di business: contiene files EJB. All'interno di questo tier sono presenti le entità di dominio e le classi che codificano le procedure di business più complesse e alcune politiche di gestione della persistenza.
- Livello dati: RDBMS Mysql[78]. Quest'ultimo livello contiene le informazioni persistenti dell'applicazione. L'accesso alla base dati, in contesto Glassfish, è mascherato dalla libreria TopLink Essentials[22].

La figura 4.1 presenta uno schema illustrativo delle tecnologie adottate nel progetto.

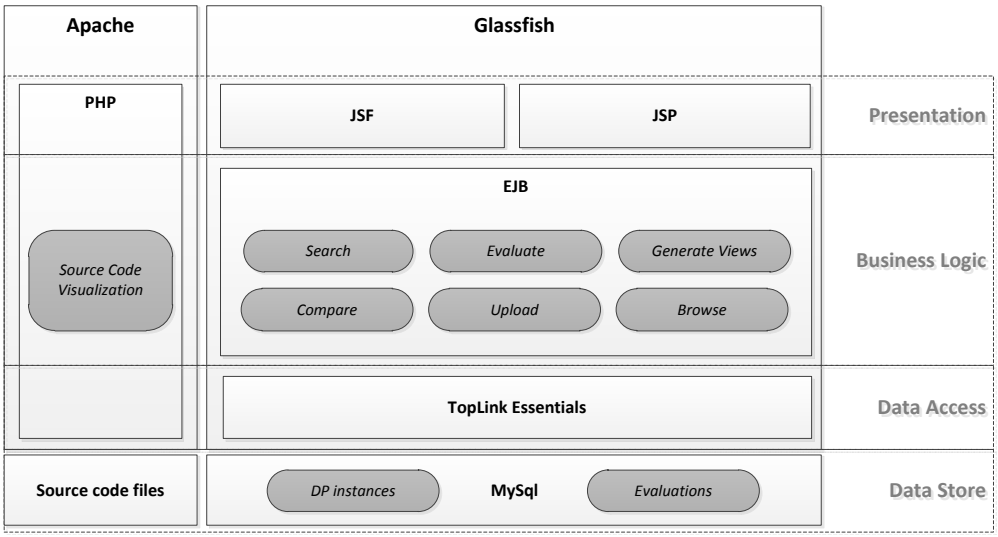

Figura 4.1: Tecnologie utilizzate nella piattaforma DPB

4.2 Modalità di interazione

La piattaforma DPB offre diverse modalità di interazione. Come mostrato nel diagramma in figura 4.2, è possibile individuare diversi attori aventi ognuno un ruolo specifico nella definizione dei contenuti della piattaforma.
Di seguito verrà presentato un elenco completo delle tipologie di utenti previste e delle azioni a loro consentite:

- Team di sviluppo di uno strumento di DPD: Se interessato al progetto, può contribuire ad alimentare la base dati della piattaforma sottomettendo i risultati del proprio strumento. I dati inviati dovranno essere strutturati secondo

le specifiche del modello descritto nella sezione 3.1 e codificati in conformità allo schema descritto nell'appendice A. L'attività di sottomissione delle analisi di sistema prodotte è supportata dalla piattaforma attraverso delle procedure interattive rese disponibili all'interno dell'area amministrativa della piattaforma. Ogni analisi di sistema è immediatamente resa disponibile al giudizio della comunità di utenti coinvolti nel progetto.

- Comunità: É composta da tutti gli utenti registrati alla piattaforma. Ad essa spetta il compito fondamentale di valutare le istanze che compongono le analisi di sistema presenti sulla piattaforma. Il loro giudizio è registrato in forma di valutazioni sintetiche numeriche accompagnate da un commento testuale. Ogni valutazione può essere supportata o meno dal resto della comunità e può essere a sua volta commentata da parte di coloro che non vi concordano. Per supportare questo processo di valutazione sono state rese disponibili delle funzionalità di ricerca e confronto, utili all'individuazione delle istanze e all'analisi complessiva dei risultati.

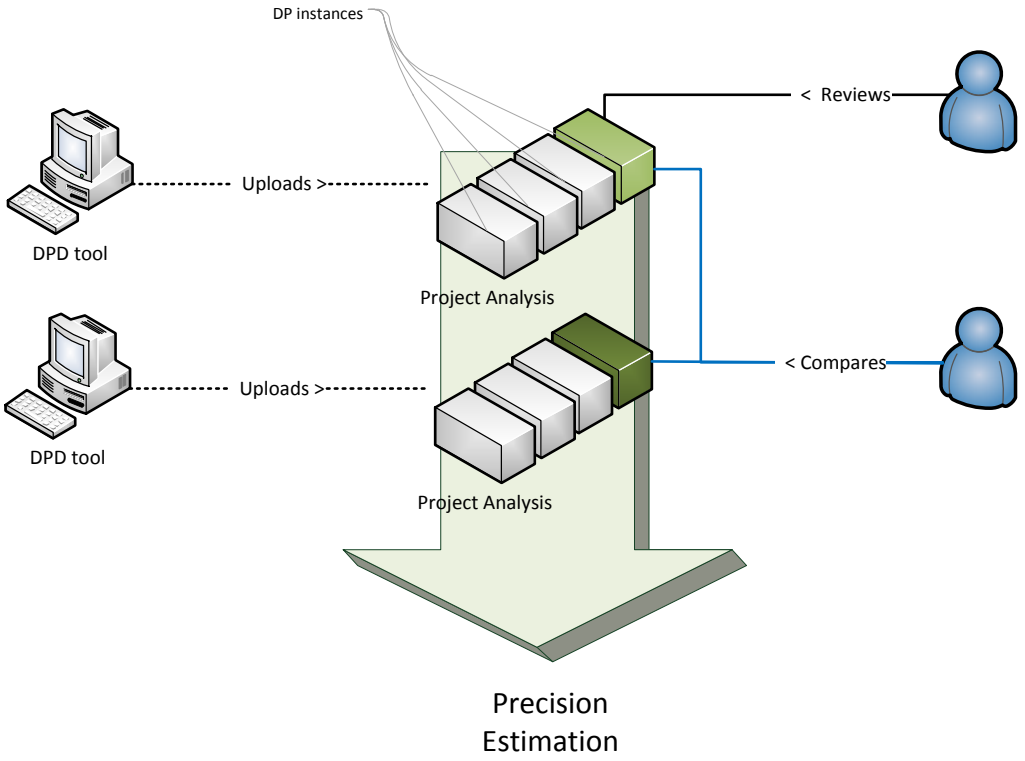

Figura 4.2: Schema generale di interazione tra utente e dati della piattaforma

Nella figura 4.3 è riportata una sintesi dello schema di navigazione della piattaforma, che mostra come le funzionalità appena descritte possano essere reperite e fruite in un normale contesto di utilizzo.

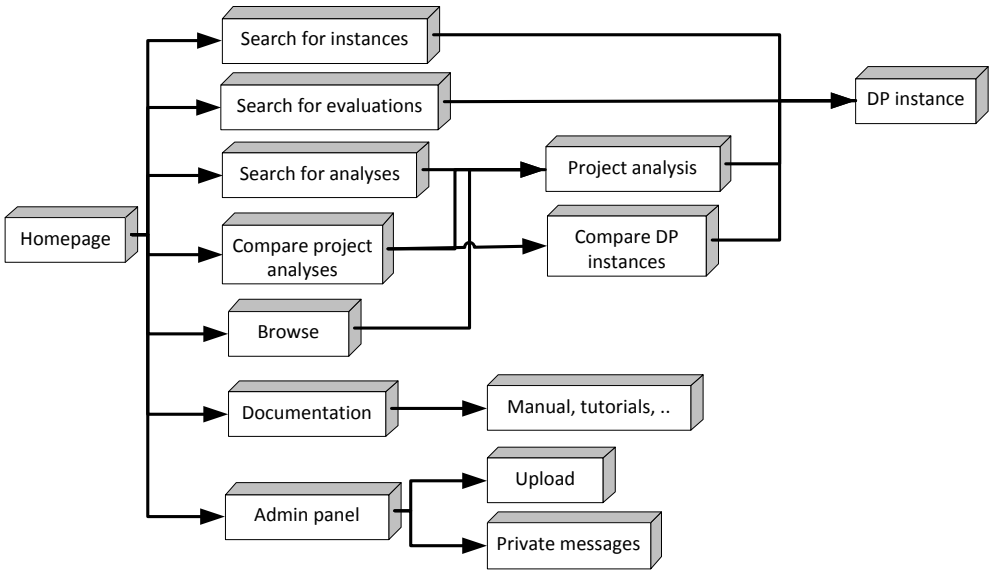

Figura 4.3: Schema di navigazione della piattaforma

4.3 Funzionalità più rilevanti

Nelle sezioni seguenti verranno descritte ed analizzate in dettaglio le funzionalità principali della piattaforma DPB.

4.3.1 Visualizzazione

Ogni istanza di DP presente sulla piattaforma è descritta da un insieme di informazioni ricco (si veda la descrizione del modello, sezione 3.1) ma poco fruibile da un utente umano. Al fine di consentire un'analisi rapida e semplice, è stato necessario progettare una pagina intuitiva e completa che fosse in grado di fornire il maggior numero di informazioni possibile senza confondere l'utente. L'obiettivo è stato raggiunto grazie alla creazione di un ampio numero di viste indipendenti (riquadro in alto nella figura 4.4) richiamabili attraverso la selezione di "tab" posti in cima alla pagina, e alla separazione netta e ben identificabili di informazioni descrittive,

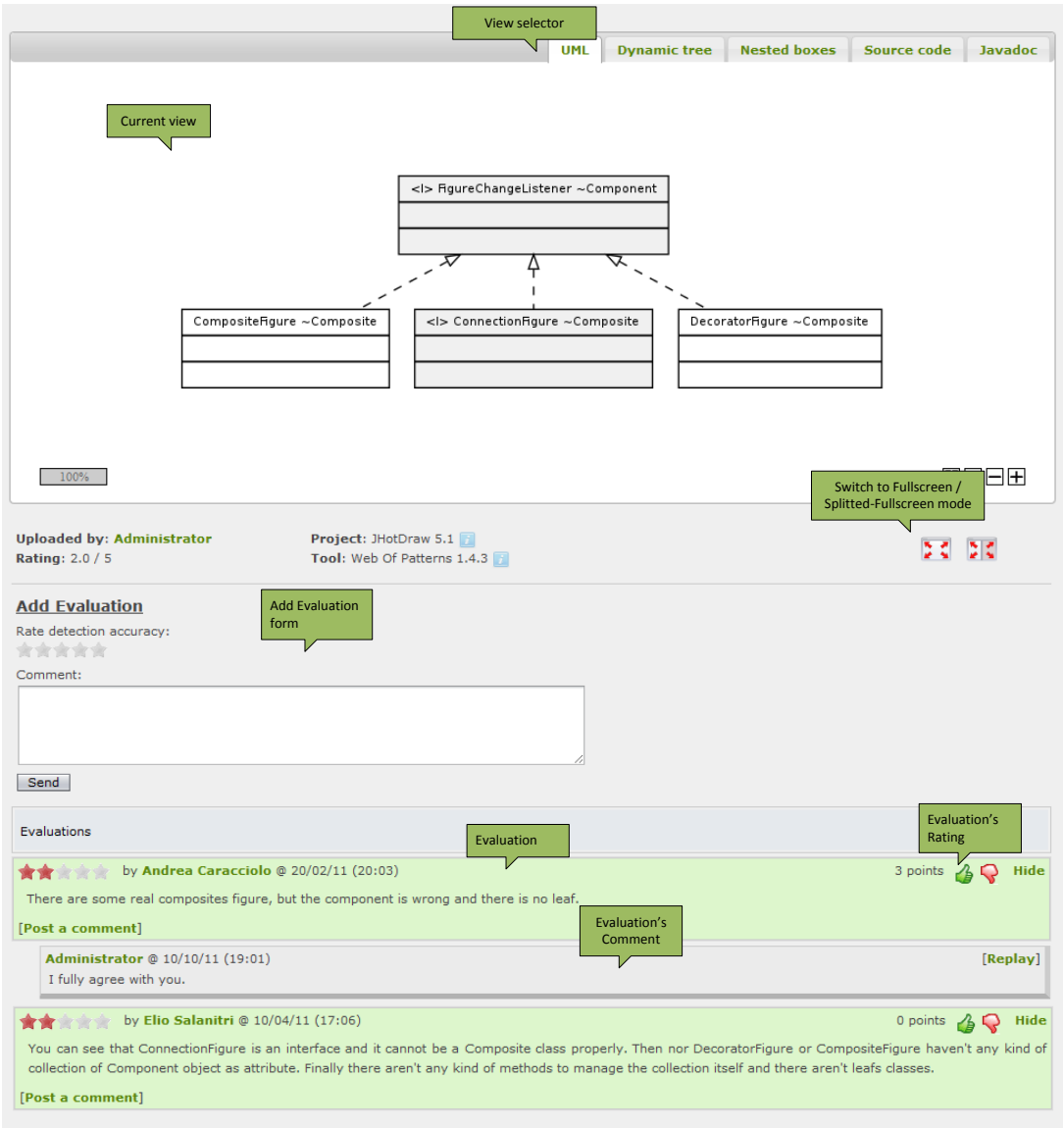

Figura 4.4: Pagina di visualizzazione

meta-dati e valutazioni. Come suggerito da alcuni utenti[1], è stata inoltre anche introdotta una modalità di visualizzazione a pieno schermo, che consente di visionare una singola vista o due viste affiancate (vedi figura 4.5) in un comodo riquadro che occupa l'intera dimensione della finestra. Tale modalità di visualizzazione consente di confrontare fonti informative diverse in modo rapido ed efficace offrendo all'utente la possibilità di formulare una valutazione più consapevole e immediata.

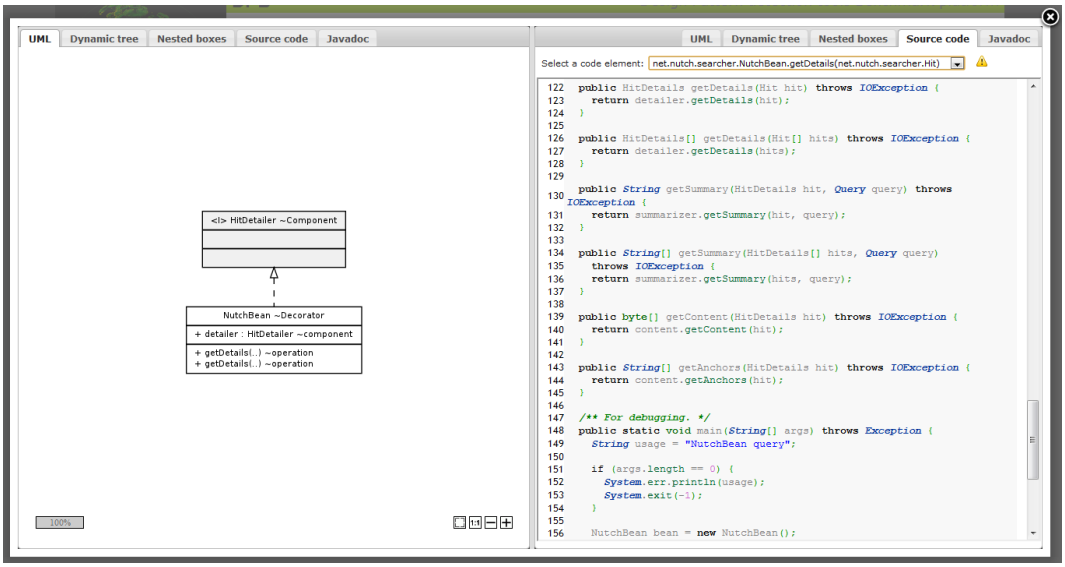

Figura 4.5: Pagina di visualizzazione: modalità a schermo intero

Di seguito verranno descritte in maggiore dettaglio le parti che compongono la pagina di visualizzazione:

- Viste: Ogni DP, se corredato delle informazioni sufficienti, può essere analizzato attraverso la consultazione delle cinque viste descritte nelle sotto-sezioni presentate di seguito. Ogni vista può essere richiamata attraverso la selezione di uno dei tab presenti nella parte alta del riquadro di presentazione.
- Meta-dati: Alla base del riquadro contenente le viste, sono presenti alcune informazioni secondarie utili a ricordare le proprietà fondamentali dell'istanza (sistema di appartenenza e strumento utilizzato per la sua scoperta), l'identità dell'utente responsabile della sua pubblicazione e il punteggio ottenuto dalla media delle valutazioni pesata rispetto ai voti assegnati ad ogni singola valutazione. Se presenti, sono anche riportati i riferimenti ad istanze "gemelle" aventi un grado di similarità del 100% rispetto all'istanza selezionata.
- Pulsanti per il passaggio a modalità di visualizzazione a pieno schermo: A destra del riquadro contenente i meta-dati, sono presenti due bottoni che per-

[1] Günter Kniesel e altri.

mettono di abilitare la visualizzazione a tutto schermo. Le modalità previste sono due: singola vista o coppia di viste affiancate in verticale (vedi figura 4.5). La visualizzazione a tutto schermo è ottenuta generando un nuovo riquadro "fluttuante" sovrapposto alla pagina.

- Modulo per aggiunta valutazione: Ogni utente registrato che desideri esprimere la sua opinione riguardo al grado di correttezza dell'istanza scoperta, può compilare il modulo presente a metà pagina sotto il riquadro dei meta-dati. I campi previsti sono due: una valutazione numerica in termini di *stelle* (dove una stella corrisponde ad una valutazione negativa e cinque stelle ad una positiva), e un commento a supporto della propria valutazione. Per evitare confusione nella formulazione delle valutazioni, la selezione delle stelle è accompagnata dalla comparsa di un commento testuale che specifica la condizione necessaria all'attribuzione di un tale giudizio (ad esempio, selezionando due stelle, compare il testo "Molti ruoli importanti sono sbagliati o non specificati").

- Valutazioni: Come spiegato sopra, le valutazioni sono composte da un valore numerico, rappresentato in forma di stelle, e da un commento testuale. Agli utenti della comunità è data la possibilità di votare una valutazione al fine di aumentare o diminuire il peso della stessa durante il calcolo del punteggio complessivo dell'istanza. Oltre a votare una valutazione, è inoltre possibile esprimere la propria opinione riguardo ad una valutazione creando un commento. Il commento è stato pensato come uno strumento di discussione a supporto del voto. Attraverso di esso è possibile giustificare il proprio voto, promuovere discussioni e influenzare l'opinione della comunità.

Il grado di correttezza di un'istanza è pari alla media ponderata delle valutazioni ad essa associate:

$$rating(instance) = \frac{\sum_{i=1}^{|evals|} eval_i \cdot votesBalance_i}{\sum_{i=1}^{|evals|} votesBalance_i} \tag{4.1}$$

dove:

$$votesBalance_i = max(def + votes_i^+ - votes_i^-, 0) \tag{4.2}$$

Segue la descrizione delle singole variabili:

- $evals$ è l'insieme delle valutazioni associate all'istanza di DP presa in esame.

- $eval_i$ è la valutazione i-esima associata all'istanza presa in esame.

- def è un valore costante attribuito ad ogni valutazione al fine di aumentare il numero minimo di consensi negativi necessari prima di considerare tale

valutazione inconsistente. Il valore def è stato attualmente stabilito pari a tre[2].

- $votes_i^+ / votes_i^-$ sono rispettivamente i voti favorevoli e contrari associati all'i-esima valutazione dell'istanza presa in esame.

A titolo d'esempio, verrà di seguito descritta l'applicazione della formula rispetto ad un'istanza specifica associata alle seguenti valutazioni:

- valutazione 1: 4 stelle (3 voti favorevoli / 1 voto contrario).
- valutazione 2: 3 stelle (8 voti favorevoli / 0 voto contrario).
- valutazione 3: 1 stelle (0 voti favorevoli / 8 voti contrari).
- valutazione 4: 4 stelle (1 voto favorevole / 4 voti contrari).

L'applicazione della formula porta ai seguenti risultati:

- $votesBalance_1 = 3 + 3 - 1 = +5$
- $votesBalance_2 = 3 + 8 - 0 = +11$
- $votesBalance_3 = 3 + 0 - 8 = -5$ (minore di 0, quindi $votesBalance_3 = 0$)
- $votesBalance_4 = 3 + 1 - 4 = 0$ (minore di 0, quindi $votesBalance_4 = 0$)

$$rating(instance) = \frac{4 \cdot 5 + 3 \cdot 11 + 1 \cdot 0 + 4 \cdot 0}{5 + 11 + 0 + 0} = \frac{20 + 33}{16} = 3.31 \qquad (4.3)$$

Il grado di correttezza stimata dell'istanza presa in esame, è pari a 3.31 su 5 (circa 66%).

Nelle prossime sotto-sezioni verranno descritte in dettaglio le singole viste fruibili dalla pagina di visualizzazione.

Vista *UML class diagram*

Questa vista presenta tutte le entità che concorrono alla definizione dell'istanza in forma di diagramma delle classi UML (vedi figura 4.6). Il diagramma presenta alcuni comandi interattivi che consentono di ingrandire/ridurre l'immagine.

La generazione del diagramma è ottenuta in maniera dinamica attraverso un programma sviluppato nel contesto di questo progetto. Tale programma riceve in input il codice del sistema analizzato (in formato JAR) e una lista delle entità da rappresentare nel diagramma (classi, metodi e campi). A partire da questi dati è in grado di esplorare il codice sorgente, riconoscere le entità specificate, ricostruire le

[2]Valore stabilito sulla base di considerazioni empiriche legate al numero di utenti facenti parte della comunità e al numero medio di voti associati alle singole valutazioni

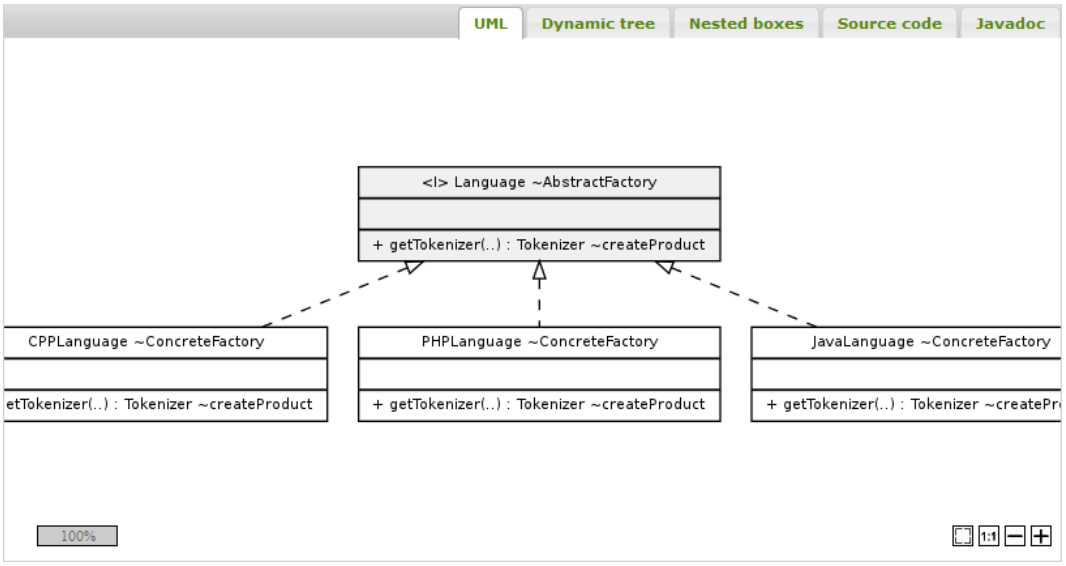

Figura 4.6: Pagina di visualizzazione: vista "UML class diagram"

relazioni presenti tra tali entità e generare un output grafico in formato PNG contenente tutte (e sole) le entità specificate in input e le relazioni presenti tra di esse. Tutte queste azioni vengono effettuate in tempo reale al momento della richiesta della pagina senza introdurre ritardi significativi nella fase di caricamento della stessa. Per ottimizzare il processo è stato inoltre introdotto un meccanismo di caching che consente di limitare il processo di generazione del grafico alla prima richiesta utente.

Vista *Dynamic tree*

Si tratta di una rappresentazione grafica interattiva che presenta le singole entità che compongono l'istanza di DP secondo una struttura ad albero (vedi figura 4.7). Ogni entità è rappresentata da un rettangolo (caratterizzato da un colore di sfondo definito in base alla classe di appartenenza), collegato attraverso delle frecce alle altre componenti dell'istanza. Il risultato finale è un grafo che riproduce fedelmente la struttura originale dell'istanza in conformità al meta-modello descritto nel capitolo 3. Il diagramma risultante può essere ingrandito o ridotto ed esportato come immagine. I singoli nodi che compongono il grafo possono essere ri-collocati manualmente o in modo automatico selezionando uno dei quattro layout preimpostati.

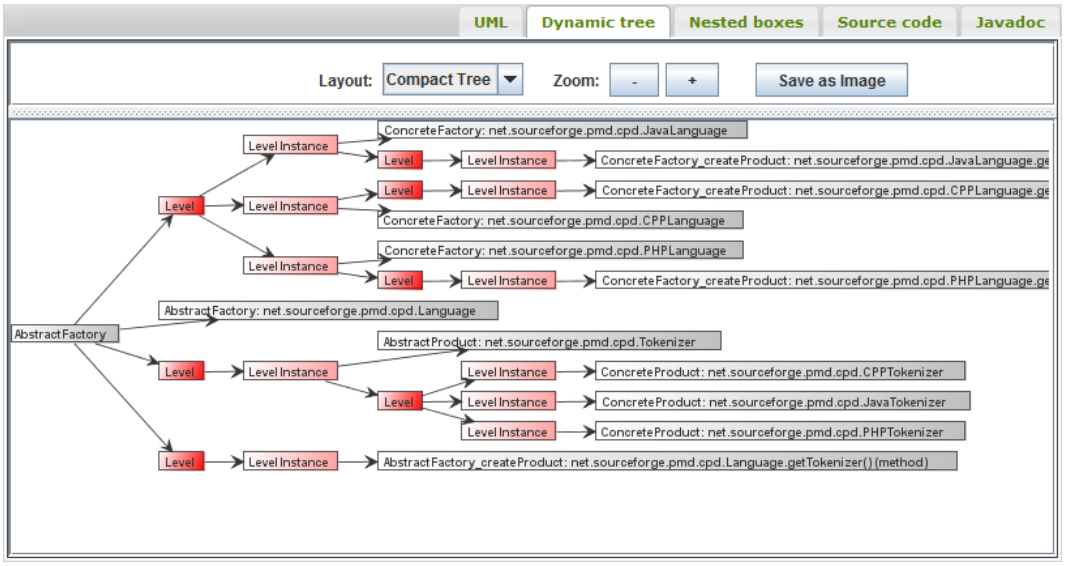

Figura 4.7: Pagina di visualizzazione: vista “Dynamic tree”

Vista *Nested boxes*

La vista nested boxes propone lo stesso modello rappresentato nella vista dynamic tree in forma di riquadri annidati in cui ogni nodo figlio è contenuto nel riquadro rappresentante il nodo genitore (vedi figura 4.8). Ogni riquadro è caratterizzato dal nome dell'entità rappresentata, dal valore dei ruoli contenuti e da un collegamento alla porzione di codice sorgente cui fanno riferimento. I riquadri sono categorizzati per colore secondo la tipologia di entità (classe, metodo, attributo) associata.

Vista *Source code*

Questa vista permette di consultare le porzioni di codice sorgente associate (in maniera implicita o esplicita) alle entità di tipo ruolo presenti nell'istanza (vedi figura 4.9). Il codice sorgente in questione è ospitato su un server SVN installato sulla stessa macchina su cui è collocato il progetto DPB. All'utente responsabile del caricamento di un'analisi di sistema, è offerta la possibilità di specificare, per ogni entità di tipo Ruolo, il percorso relativo del file contenente l'elemento individuato. Tale percorso deve essere compatibile con lo schema di indirizzamento adottato nella piattaforma, e potrà essere ulteriormente specializzato citando il numero della riga in cui compare la dichiarazione dell'elemento referenziato. Questa vista supporta l'evidenziazione sintattica del codice e consente di navigare l'intera base di codice del sistema ispezionato attraverso collegamenti ipertestuali. Per l'implementazione di questa vista si è fatto uso del programma WebSVN [12].

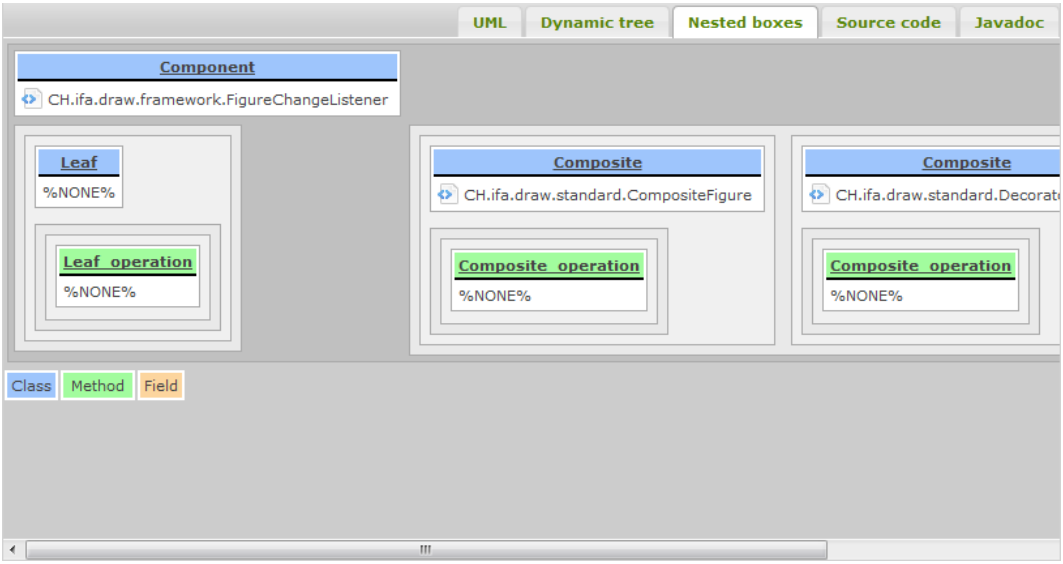

Figura 4.8: Pagina di visualizzazione: vista “Nested boxes”

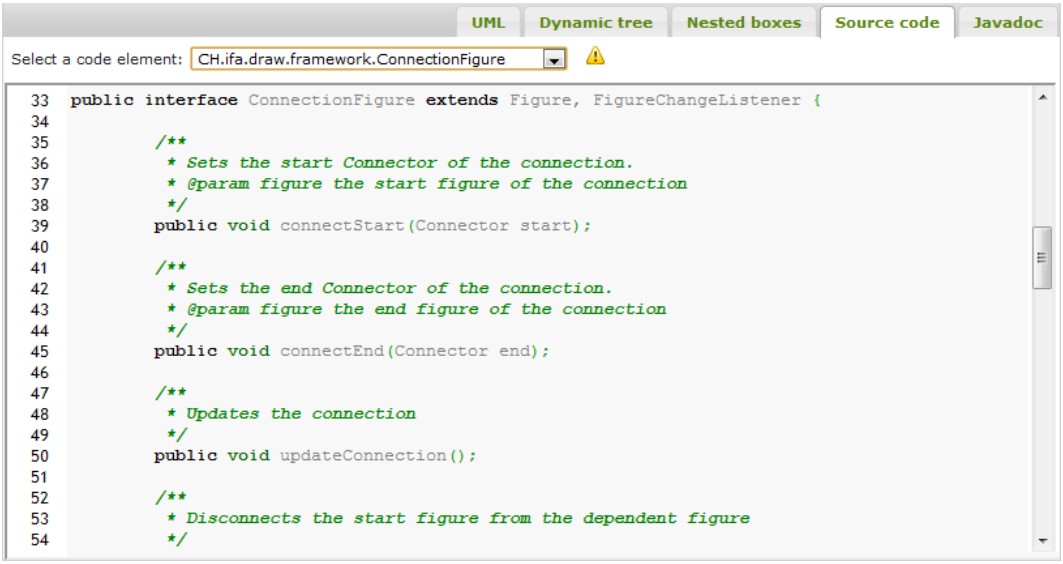

Figura 4.9: Pagina di visualizzazione: vista “Source code”

Vista *Javadoc*

Nel caso in cui l'istanza visualizzata appartenga ad un sistema Java, è possibile visualizzare una vista che, in maniera analoga alla precedente, offre una rappresentazione a livello di codice di tutte le entità di tipo ruolo presenti nell'istanza (vedi figura 4.10). Le informazioni presentate in questa vista sono il risultato dell'esecuzione del programma `Javadoc`, fornito nell'SDK (Software development kit) di Java.

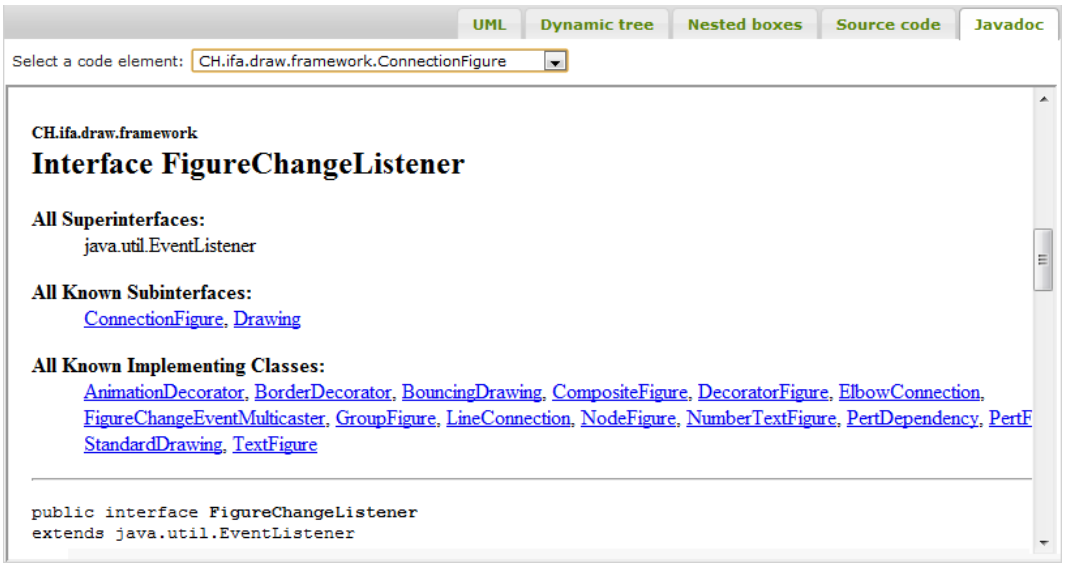

Figura 4.10: Pagina di visualizzazione: vista "Javadoc"

4.3.2 Ricerca

Se l'utente desidera ricercare delle analisi di sistema o delle istanze di DP presenti all'interno della piattaforma, può farlo utilizzando la funzionalità di ricerca (vedi figura 4.11). Tale funzionalità produce il proprio risultato sulla base della compilazione dei seguenti campi di ricerca:

- Linguaggio di programmazione/Sistema/Strumento di DPD/DP: L'insieme di questi campi di ricerca consente di limitare lo spazio di ricerca alle sole analisi di sistema associate ai valori selezionati. I campi di ricerca sono popolati dinamicamente secondo il valore selezionato nel campo immediatamente precedente. Se, ad esempio, selezionassimo gli strumenti A e B e cliccassimo sulla freccetta posizionata a destra del campo, otterremmo una lista di tutti e soli i DP presenti nelle analisi associate a tali strumenti. Tale modalità di selezione,

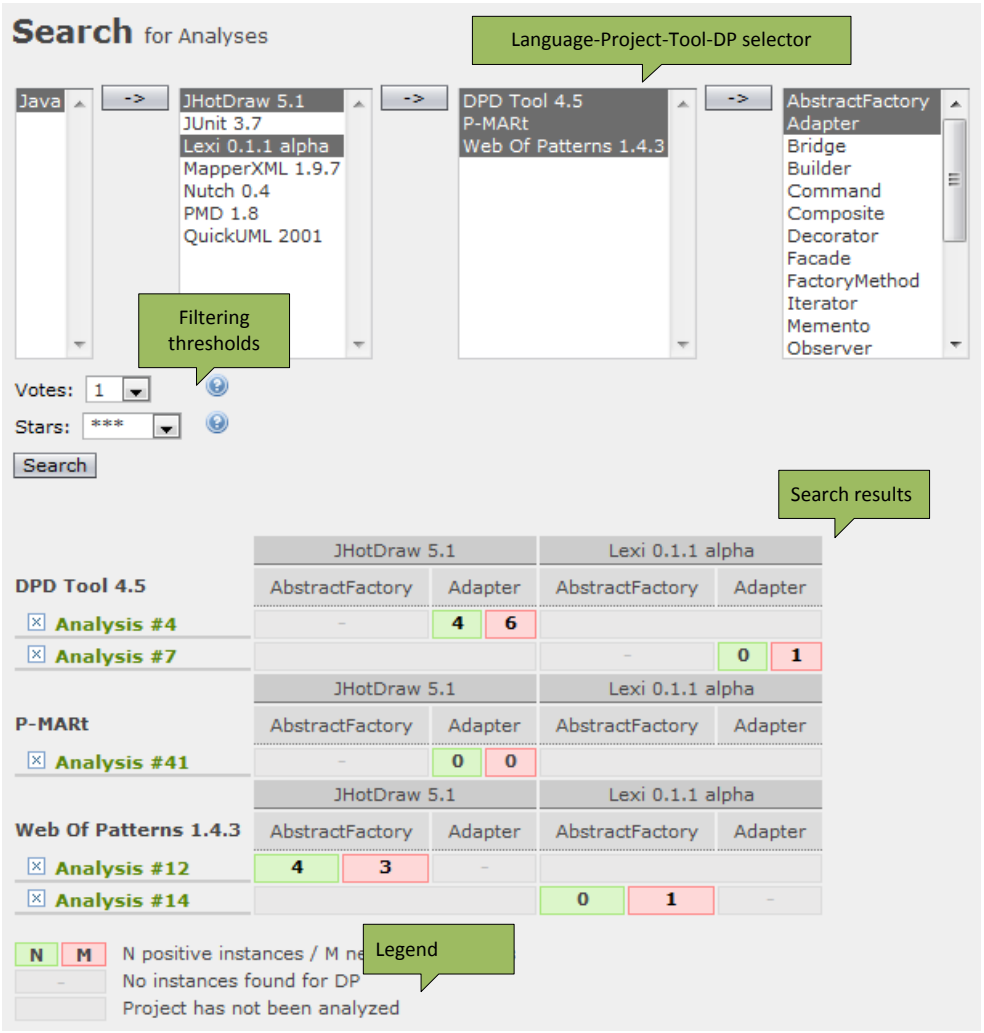

Figura 4.11: Funzionalità di ricerca

unita alla possibilità di selezionare più voci della stessa categoria, permette di ottenere risultati molto precisi e personalizzati.

- Soglia minima di opinioni: Definisce il numero minimo di valutazioni e voti necessari affinché un'istanza possa essere considerata come "valutata". Se un'istanza presenta un numero cumulativo di valutazioni e voti inferiore a tale soglia, tale istanza verrà semplicemente ignorata ai fini del conteggio del numero di istanze positive/negative presenti in una data analisi di sistema. L'utilità di questo campo è data dalla possibilità di poter ignorare tutte quelle istanze che non hanno ancora maturato un numero di giudizi sufficiente da poter trarre una valutazione soddisfacente e condivisa relativa al loro grado di correttezza.
- Soglia di punteggio: Definisce il punteggio minimo richiesto ad un'istanza per poter essere considerata positiva. Se, ad esempio, tale valore è pari a 3, verranno conteggiate come positive tutte le istanze aventi un numero di valutazioni e voti maggiori della soglia stabilita dal campo precedente e un punteggio medio maggiore o uguale di 3.

I risultati derivanti dalla compilazione dei campi appena descritti, sono rappresentati in forma tabellare. Sulle righe troviamo tutte le analisi di sistema che rispondono ai criteri definiti, raggruppate per strumento. Sulle colonne, sono specificati i sistemi e i DP selezionati durante la fase di definizione della ricerca. Nelle celle interne è possibile trovare due valori (su sfondo verde e rosso), un trattino o nessun valore. Nel primo caso, i valori rappresentati corrispondono rispettivamente al numero di istanze positive e negative trovate all'interno dell'analisi di sistema specificata nella riga, per il DP e il sistema identificati dalla colonna. Nel secondo caso invece, il simbolo utilizzato serve ad indicare che non è stato trovata alcuna istanza del pattern identificato dalla colonna. Nell'ultimo caso, l'assenza di un qualsiasi valore suggerisce che il sistema specificato nella colonna non è stato analizzato nel contesto dell'analisi riportata sulla riga.

Considerazioni sui risultati della ricerca

Oltre a consentire di individuare analisi di sistema, la funzionalità di ricerca offre la possibilità di ottenere una panoramica semplice ed efficace dei risultati ottenuti dai vari strumenti sui singoli sistemi analizzati rispetto a determinati DP. Questa seconda modalità di utilizzo è una valida strategia di confronto basata su parametri variabili e limitata ad un insieme di elementi definiti dall'utente. Invece di proporre una semplice classifica dei punteggi ottenuti dai vari strumenti, si è deciso di lasciare all'utente l'onere del raffronto, al fine sottrarsi alla logica competitiva e di rendere più trasparenti gli schemi di punteggio adottati.

Di seguito verranno descritte due chiavi di lettura dei risultati che rispondono alla strategia sopra descritta.

- Confronto tra diversi strumenti sulla stessa analisi: Selezionando lo stesso sistema, lo stesso DP e un numero variabile di strumenti, è possibile confrontare i risultati ottenuti da diversi strumenti nell'ambito dello stesso contesto operativo. Nell'esempio proposto nella figura 4.12, è ad esempio possibile osservare che la precision dello strumento DPD tool, pari al 100%, è superiore a quella dello strumento Web of Patterns (60%), e P-MARt (0%). Ovviamente la significatività dei risultati sarà molto spesso proporzionale al numero di istanze positive e negative individuate nel contesto dell'analisi.

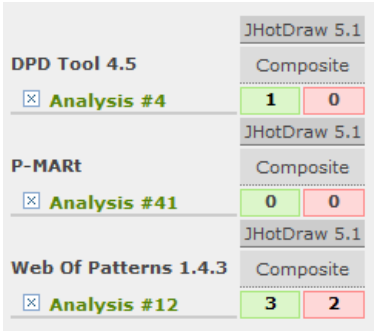

Figura 4.12: Esempio di ricerca 1

- Confronto tra diversi DP nell'ambito della stessa analisi: Per valutare le performance di uno strumento rispetto ai vari pattern che è in grado di individuare, è possibile selezionare un sistema, uno strumento e un certo insieme di DP. I risultati ottenuti, come mostrato in figura 4.13, mostrano come il valore di precision possa variare a secondo del pattern analizzato. Nell'esempio vediamo che il pattern Factory Method (precision pari al 100%) viene individuato con maggiore successo rispetto ai pattern Adapter (40%) e Decorator (33%).

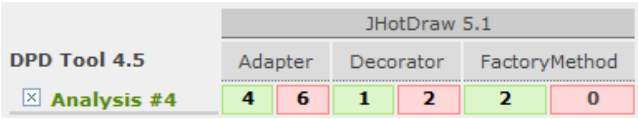

Figura 4.13: Esempio di ricerca 2

4.3.3 Confronto

Lo strumento di confronto messo a disposizione degli utenti della piattaforma, consente di ottenere una visione sintetica del grado di similarità presente tra due analisi di sistema. La granularità del confronto può variare a seconda dell'interesse dell'utente passando dalla modalità di confronto per analisi di sistema a quella per singole istanze di DP. L'interfaccia della funzionalità qui presentata, riportata nella figura 4.14, si compone di una parte superiore dedicata ai campi di ricerca delle analisi di

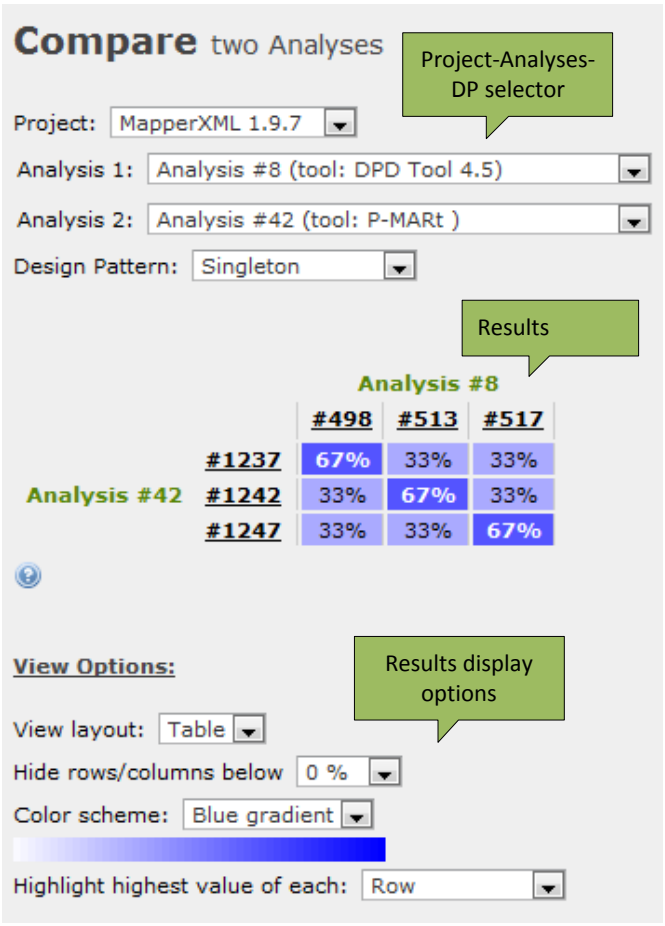

Figura 4.14: Pagina di confronto tra due analisi di sistema

sistema da confrontare, una parte intermedia per i risultati, e una parte inferiore che permette di modificare alcune opzioni di presentazione. Una volta selezionato il sistema di riferimento, due analisi di tale sistema e un DP, verrà presentata una tabella contenente i risultati del confronto. Sull'asse delle Y sono presenti tutte le istanze rilevate nel contesto della prima delle due analisi selezionate, mentre sull'asse delle X vi sono le istanze della seconda. Nelle celle interne della tabella troviamo la percentuale di similarità tra le due istanze presenti sulla stessa riga e sulla stessa colonna. Tale percentuale è calcolata applicando l'algoritmo descritto nel capitolo 5. Nella parte bassa della pagina sono presenti alcune opzioni che permettono all'utente di modificare alcune caratteristiche di presentazione dei risultati. É possibile passare da una presentazione tabellare ad una semplice lista ordinata dei risultati, nascondere le righe/colonne che presentano solamente valori al di sotto di una data soglia, modificare lo schema di colore utilizzato per mettere in evidenza le celle con valori più alti, o abilitare l'evidenziazione del valore più alto presente in ogni riga/colonna.

Confronto tra due istanze di DP

Cliccando su uno dei valori riportati nella tabella dei risultati, è possibile visionare un confronto dettagliato tra le due istanze che hanno prodotto tale valore. Come è possibile vedere nella figura 4.15, le due istanze confrontate sono rappresentate attraverso un'unica vista a scatole annidate, del tutto simile a quella discussa nella sezione 4.3.1, in cui ogni ruolo della prima istanza è contenuto nella stessa scatola dove sono presenti i ruoli dello stesso tipo appartenenti alla seconda istanza. Questa rappresentazione unificata consente di vedere affiancati i ruoli che si desidera mettere a confronto. L'uso dei colori verde e rosso nella presentazione dei valori associati ai singoli ruoli rende ancora più facile l'individuazione dei punti di somiglianza tra le due istanze. Cliccando sui singoli valori è inoltre possibile applicare dei filtri selettivi che permettono di agevolare la lettura della vista limitando la presentazione dei ruoli alle sole entità presenti nei nodi figlio dell'entità selezionata.

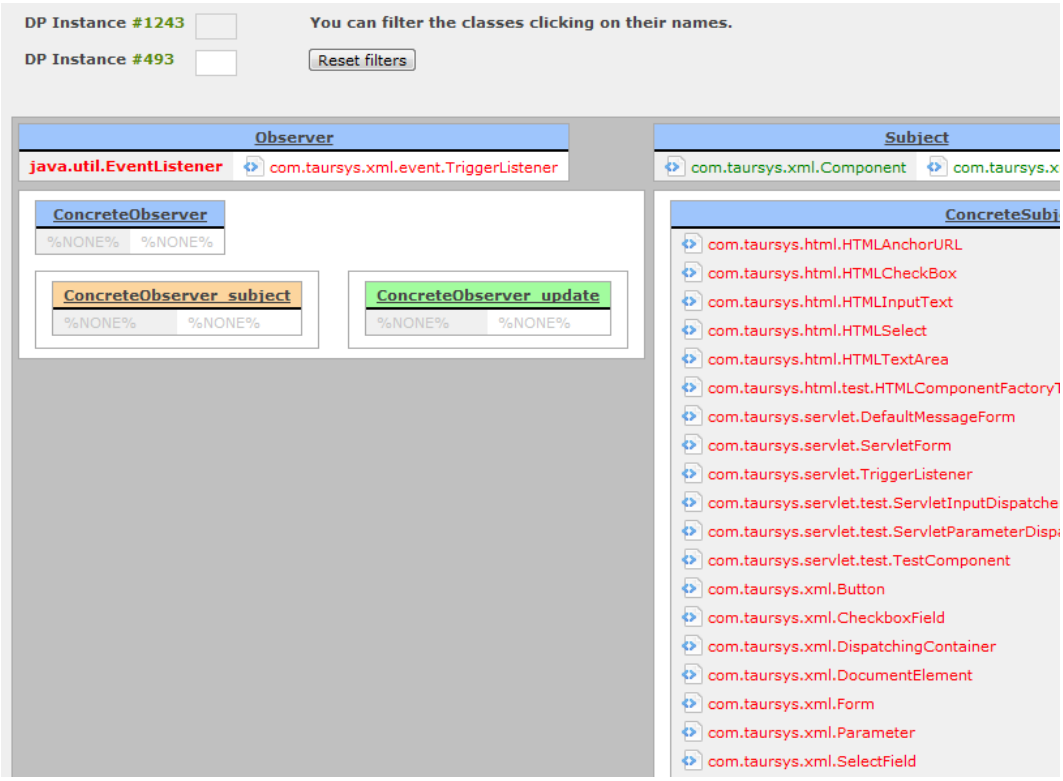

Figura 4.15: Pagina di confronto tra due istanze di DP

Capitolo 5

Un algoritmo per il confronto di istanze di design pattern

Data l'esigenza di analizzare grandi quantità di istanze al fine di trovare dei punti di contatto tra analisi effettuate da strumenti diversi, si è reso necessario lo sviluppo di un algoritmo in grado di calcolare automaticamente il grado di similarità tra coppie di istanze di DP.

Tale soluzione è stata integrata nel contesto dell'applicazione secondo le modalità descritte nella sezione 4.3.3.

5.1 Fasi di definizione dell'algoritmo

In questa sezione verranno descritte le varie fasi che hanno portato alla definizione dell'algoritmo. La lettura di questo capitolo presuppone la comprensione dei concetti discussi nel capitolo 3.1.

5.1.1 Versione 1: prima definizione dell'algoritmo

La prima versione dell'algoritmo di confronto è stata formulata durante le prime fasi di realizzazione del progetto, in concomitanza con la definizione del meta-modello descritto nel capitolo 3. La validità dell'approccio ha successivamente trovato conferma nella letteratura di ambiti di studio correlati. In particolare è stata riscontrata una notevole somiglianza rispetto all'approccio descritto nell'articolo di Vanneschi et al. [92], che introduce una funzione per la definizione del coefficiente di correlazione della distanza di fitness basata sul calcolo della distanza strutturale presente tra due alberi.

Funzione di confronto principale

La prima versione dell'algoritmo preso in esame può essere descritto come segue.

$$sim(inst_1, inst_2) = \begin{cases} weight_0 + \sum_{i=1}^{n} simL(subL_{1,i}, subL_{2,i}, 1) & \text{se } root_2 \text{ è equiv. a } root_1 \\ 0 & \text{altrimenti} \end{cases} \tag{5.1}$$

dove:

- $inst_1$ e $inst_2$ sono due istanze distinte di DP.
- $root_i$ è il nodo radice di $inst_i$.
- $sim(inst_1, inst_2)$ è una funzione che restituisce un valore (compreso tra 0 e 1) che rappresenta il grado di similarità tra $inst_1$ e $inst_2$.
- $weight_i$ (con $0 \leq i \leq MaxHeight$ dove $MaxHeight$ è l'altezza massima tra quelle degli alberi associati a $inst_1$ e $inst_2$) è il peso assegnato ai nodi del livello i-esimo degli alberi associati a $inst_1$ e $inst_2$. La definizione dei pesi associati ai vari livelli è esplicitata nella sezione 5.1.1.
- n è il numero dei sotto-livelli del nodo radice presenti in $inst_1$.
- $subL_{1,i}$ è il nodo figlio i-esimo di $root_1$.
- $subL_{2,i}$ è il nodo figlio i-esimo di $root_2$ "equivalente" a $subL_{1,i}$. La nozione di equivalenza è esplicitata nella sezione 5.1.1.
- $simL(l_1, l_2, depth)$ è una funzione ricorsiva che calcola il grado di similarità tra i sotto-alberi associati a l_1 e l_2. Per maggiori dettagli riguardo a tale funzione, si faccia riferimento alla sezione 5.1.1.

Per una migliore comprensione della nomenclatura attribuita ai vari nodi degli alberi associati a $inst_1$ e $inst_2$, si invita il lettore a confrontare la formula 5.1 con le due istanze prese ad esempio nella figura 5.1.

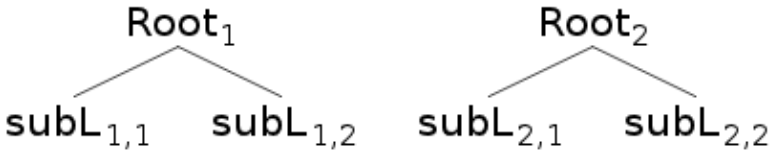

Figura 5.1: Esempio di due istanze $inst_1$ e $inst_2$ a due livelli

Il listato 5.1 presenta l'implementazione Java della funzione specificata nella formula 5.1. I metodi aggiuntivi, riportati in maniera parziale nella parte finale del codice, verranno trattati nelle sezioni seguenti.

```
1  public class CompareAlgorithm {
2    double[] weights;
3    final static int ROOT_DEPTH = 0;
4    final static int FIRST_LEVEL_DEPTH = 1;
5
6    public double compare(DpInstance dp1, DpInstance dp2) {
7      DpDef def1 = dp1.getDpDef();
8      DpDef def2 = dp2.getDpDef();
9
10     // pre-condition: must have same definition
11     if (!def1.equals(def2)) {
12       return 0;
13     }
14
15     // ** phase 1: calculate weights
16     this.calculateWeights(def1);
17
18     // ** phase 2: do comparison
19     double result = this.calculateSimilarity(dp1, dp2);
20
21     return this.round(result, 2);
22   }
23
24   protected double calculateSimilarity(DpInstance dp1, DpInstance dp2)
       {
25     LevelInstance root1 = dp1.getRoot();
26     LevelInstance root2 = dp2.getRoot();
27     double similarity = 0;
28
29     if (root1.isEquivalentOf(root2)) {
30       similarity += 1 * weights[ROOT_DEPTH];
31
32       for (Level l : root1.getSubLevels()) {
33         Level twinL = l.searchEquivalentIn(root2.getSubLevels());
34         similarity += simL(l, twinL, FIRST_LEVEL_DEPTH);
35       }
36     }
37
38     return similarity;
39   }
40
41   protected void calculateWeights(DpDef def) {..}
42   protected double simL(Level l1, Level l2, int depth) {..}
43   private double round(double d, int decimalPlace) {..}
44 }
```

Listing 5.1: metodo principale (versione 1)

Assegnazione dei pesi

Come precedentemente accennato, la funzione di similarità calcola il proprio valore assegnando un peso differente ad ogni livello analizzato. La funzione di assegnamento pesi definita nella prima versione dell'algoritmo è lineare, decrescente e a valori razionali. Di seguito verrà esplicitata la sua definizione teorica.

$$weight_i = \frac{depthScore_i}{totalpoints} \tag{5.2}$$

dove:

- i è il livello dell'albero preso in considerazione (con $0 \leq i \leq treeHeight$ dove $treeHeight$ è l'altezza dell'albero analizzato).
- $depthScore_i = treeHeight - i$
- $totalpoints = \sum_{j=1}^{treeHeight} depthScore_i \cdot numLevels_i$ (dove $numLevels_i$ è il numero di nodi di tipo `Level` presenti all'i-esimo livello di profondità).

Il listato di codice 5.2 presenta l'implementazione Java della formula 5.2.

```
1  protected void calculateWeights(DpDef def) {
2    int treeHeight = def.getTreeHeight();
3    this.weights = new double[treeHeight];
4    int totalPoints = 0;
5
6    for (int depth = 0; depth < treeHeight; depth++) {
7      int nodePoints = (treeHeight - depth);
8      totalPoints += nodePoints * def.getNumLevelsByDepth(depth);
9    }
10
11   for (int depth = 0; depth < treeHeight; depth++) {
12     int nodePoints = (treeHeight - depth);
13     this.weights[depth] = (double) nodePoints / (double) totalPoints;
14   }
15 }
```

Listing 5.2: definizione dei pesi per singolo livello di profondità (versione 1)

Similarità tra livelli e istanze di livello

Verrà di seguito definita la funzione *simL*, che consente di ottenere una misura del grado di somiglianza tra due nodi di tipo `Level`. Di seguito verrà esplicitata la sua

definizione teorica.

$$simL(l_1, l_2, depth) = \frac{|equivLis| \cdot weight_{depth}}{\max(|subLis_1|, |subLis_2|)} + \sum_{i=1}^{m} simL(subL_{1,i}, subL_{2,i}, depth + 1) \tag{5.3}$$

dove:

- $simL(l_1, l_2, depth)$ è una funzione che restituisce un valore (compreso tra 0 e 1) che rappresenta il grado di similarità tra due nodi di tipo `Level` (l_1 e l_2) collocati ad un livello di profondità *depth*.
- *equivLis* è un insieme che comprende i nodi figlio di l_1 (nodi di tipo `LevelInstance`) per i quali esiste almeno un nodo equivalente figlio di l_2. Per maggiori dettagli riguardo alla nozione di equivalenza, si faccia riferimento alla sezione 5.1.1.
- $subLis_i$ è l'insieme dei nodi figlio di l_i.
- m è il numero dei sotto-livelli del nodo l_1 (o, equivalentemente, del nodo l_2).

Il listato 5.3 presenta l'implementazione Java della funzione contenuta nella formula 5.3.

```
1  protected double simL(Level l1, Level l2, int depth) {
2    Collection<LevelInstance> subLis1 = l1.getSubLevelInstances();
3    Collection<LevelInstance> subLis2 = l2.getSubLevelInstances();
4    double maxSubLiCount = Math.max(subLis1.size(), subLis2.size());
5    int equivalentLis = 0;
6    double subLevelsScore = 0;
7
8    for (LevelInstance subLi1 : subLis1) {
9      LevelInstance twinsubLi = subLi1.searchEquivalentIn(subLis2);
10
11     if(twinsubLi!=null){
12       equivalentLis++;
13       for (Level l : subLi1.getSubLevels()) {
14         Level twinSubL = l.searchEquivalentIn(twinsubLi.getSubLevels())
               ;
15         subLevelsScore += simL(l, twinSubL, depth + 1);
16       }
17     }
18   }
19
20   double currentLevelScore = (equivalentLis * weights[depth]) /
       maxSubLiCount;
21   return subLevelsScore + currentLevelScore;
22 }
```

Listing 5.3: calcolo della similarità tra nodi di tipo "Level" (versione 1)

Equivalenza tra livelli, istanze di livello e ruoli

Nella presente sezione verranno descritti i criteri secondo cui due entità di tipo `Level`,`LevelInstance` o `Role` possono essere considerate equivalenti.

Il listato di codice 5.4 riporta i metodi utilizzati per individuare nodi equivalenti dato un nodo modello e un insieme di candidati dello stesso tipo.

```
1  class LevelInstance {
2    public LevelInstance searchEquivalentIn(Collection<LevelInstance> lis
         ) {
3      for (LevelInstance li : lis) {
4        if (this.isEquivalentOf(li)) {
5          return li;
6        }
7      }
8      return null;
9    }
10 }
11
12 class Level {
13   public Level searchEquivalentIn(Collection<Level> ls) {
14     for (Level l : ls) {
15       if (this.isEquivalentOf(l)) {
16         return l;
17       }
18     }
19     return null;
20   }
21 }
22
23
24 class Role {
25   public Role searchEquivalentIn(Collection<Role> rs) {
26     for (Role r : rs) {
27       if (this.isEquivalentOf(r)) {
28         return r;
29       }
30     }
31     return null;
32   }
33 }
```

Listing 5.4: ricerca di un nodo equivalente su un certo insieme di nodi di tipo “Level”, “LevelInstance” e “Role” (versione 1)

Nel listato 5.5 sono esplicitati i metodi utilizzati dalle singole classi al fine di verificare l’effettiva equivalenza tra una propria istanza e un’altra dello stesso tipo.

Come è possibile vedere, il concetto di equivalenza si declina in maniera diversa a seconda della classe di nodi che si va a considerare.

```
1  class LevelInstance {
2    public boolean isEquivalentOf(LevelInstance li) {
3      for (Role r1 : this.getRoles()) {
4        Role twinRole = r1.searchEquivalentIn(li.getRoles());
5        if (twinRole == null) {
6          return false;
7        }
8      }
9      return true;
10   }
11 }
12
13 class Level {
14   public boolean isEquivalentOf(Level l) {
15     boolean sameDef = this.getDefinition().equals(l.getDefinition());
16
17     if (sameDef) {
18       return true;
19     }
20     return false;
21   }
22 }
23
24 class Role {
25   public boolean isEquivalentOf(Role r) {
26     boolean sameValue = this.getRoleValue().equals(r.getRoleValue());
27     boolean sameDef = this.getDefinition().equals(r.getDefinition());
28     boolean sameLevel = this.getLevel().isEquivalentOf(r.getLevel());
29
30     if (sameValue && sameDef && sameLevel) {
31       return true;
32     }
33     return false;
34   }
35 }
```

Listing 5.5: calcolo della similarità tra nodi di tipo “Level”, “LevelInstance” e “Role” (versione 1)

Nello specifico:

- un oggetto `LevelInstance` A è considerato equivalente ad un oggetto `LevelInstance` B se:
  - per ogni `Role` contenuto in A, esiste un `Role` equivalente in B.
- un oggetto `Level` A è considerato equivalente ad un oggetto `Level` B se:
  - A e B sono associati allo stesso `LevelDef`.
- un oggetto `Role` A è considerato equivalente ad un oggetto `Role` B se:
  - il valore associato ad A è identico al valore associato a B.
  - A e B sono associati allo stesso `RoleDef`.
  - A e B sono associati a dei `Level` tra loro equivalenti.

5.1.2 Versione 2: ottimizzazione dello schema di assegnamento pesi

Nella seconda fase di definizione dell'algoritmo si è deciso di modificare la funzione di attribuzione dei pesi passando da una definizione a crescita lineare ad una a crescita logaritmica. Tale modifica ha consentito di ottenere una variazione più graduale dei pesi, una penalizzazione minore dei nodi foglia e dei risultati globali più verosimili. La modifica introdotta nella formula 5.2, può essere sintetizzata come segue:

$$depthScore_i = \log_{10}(treeHeight - i) + 1 \tag{5.4}$$

Il listato di codice 5.6 consente di visionare in dettaglio le modifiche apportate alla funzione di assegnamento pesi originariamente specificata nel frammento di codice 5.2.

Per comprendere in maniera più concreta la modifica apportata, invitiamo il lettore a visionare la figura 5.2, che riporta il grafico della funzione di assegnazione dei pesi nell'algoritmo versione 1, e la stessa funzione nell'algoritmo versione 2, entrambe applicate a due istanze a 5 livelli, aventi un solo nodo `Level` per ogni livello di profondità.

```
protected void calculateWeights(DpDef def) {
  int treeHeight = def.getTreeHeight();
  this.weights = new double[treeHeight];
  double totalPoints = 0;

  for (int depth = 0; depth < treeHeight; depth++) {
    double nodePoints = Math.log10(treeHeight - depth)+1;
    totalPoints += nodePoints * def.getNumLevelsByDepth(depth);
  }

  for (int depth = 0; depth < treeHeight; depth++) {
    double nodePoints = Math.log10(treeHeight - depth)+1;
    this.weights[depth] = (double) nodePoints / (double) totalPoints;
  }
}
```

Listing 5.6: definizione dei pesi per singolo livello di profondità (versione 2)

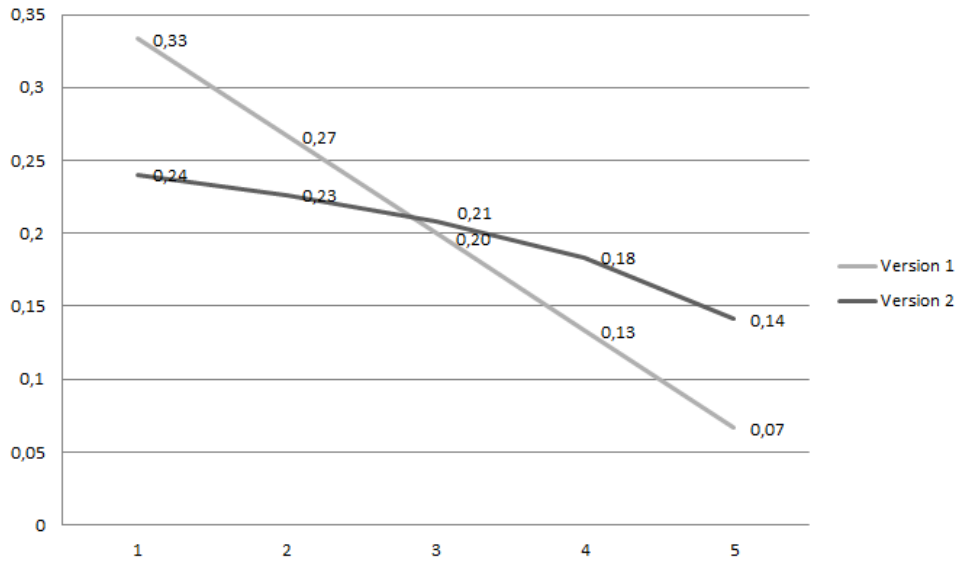

Figura 5.2: confronto tra i valori delle funzioni di assegnazione dei pesi dell'algoritmo in versione 1 e 2. Sull'asse delle ordinate sono riportati i valori, ottenuti dall'applicazione della funzione, associati ai ruoli presenti nel livello indicato sull'asse delle ascisse.

5.1.3 Versione 3: comparazione simmetrica

La terza versione dell'algoritmo è nata dall'esigenza di rendere l'algoritmo simmetrico rispetto ai propri operandi ($sim(inst_1, inst_2) = sim(inst_2, inst_1)$). Per far ciò

è stato necessario apportare delle modifiche al metodo `isEquivalentOf` della classe `LevelInstance`.

La nuova implementazione risultante da tale modifica può essere visionata nel listato di codice 5.7.

```
1  class LevelInstance {
2    public boolean isEquivalentOf(LevelInstance li) {
3      for (Role r1 : this.getRoles()) {
4        Role twinRole = r1.searchEquivalentIn(li.getRoles());
5        if(twinRole == null){
6          return false;
7        }
8      }
9      for (Role r2 : li.getRoles()) {
10       Role twinRole = r2.searchEquivalentIn(this.getRoles());
11       if(twinRole == null){
12         return false;
13       }
14     }
15     return true;
16   }
17 }
```

Listing 5.7: nuova definizione del metodo che consente di calcolare il grado di similarità presente tra due nodi di tipo "LevelInstance"

Seguendo l'esempio illustrato nella figura 5.3, in cui due nodi di tipo `LevelInstance` sono associati a due insiemi di ruoli di cardinalità diversa, e in cui un insieme di ruoli e sotto-insieme dell'altro, è possibile osservare la seguente variazione nei risultati:

- versione 2 (e precedenti): $simL(l_1, l_2) = 1$, $simL(l_2, l_1) = 0$
- versione 3: $simL(l_1, l_2) = simL(l_2, l_1) = 0$

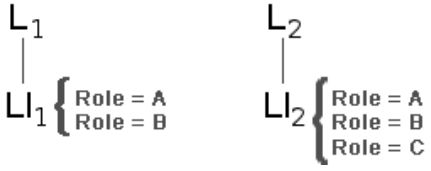

Figura 5.3: Due istanze di LevelInstance associate a insiemi di ruoli di diversa cardinalità

5.1.4 Versione 4: calcolo della similarità in base a singoli ruoli

L'ultima variazione apportata all'algoritmo, è stata introdotta al fine di rendere più accurato il confronto tra singoli nodi di tipo `LevelInstance`. La modifica introdotta in questa fase segna il passaggio dall'utilizzo di una modalità di confronto tra nodi di tipo `LevelInstance` ad esito booleano, all'utilizzo di una funzione di confronto a valori razionali ($simLI(li_1, li_2) \in \mathbb{Q}$).

Se, ad esempio, prendessimo in considerazione due nodi di tipo `LevelInstance` (vedi figura 5.4), rispettivamente associati alle coppie di ruoli aventi valore A e B (nodo li_1) e A e C (nodo li_2), potremmo osservare le seguenti differenze nei risultati della funzione $simL$:

- versione 3 (e precedenti): $simL(l_1, l_2) = 0$
- versione 4: $simL(l_1, l_2) = 0.5$

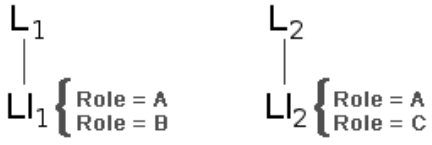

Figura 5.4: Due istanze di LevelInstance associate a insiemi di ruoli eterogenei

Di seguito verranno discusse in dettaglio le differenze introdotte in questa fase rispetto alle versioni precedenti dell'algoritmo.

La prima variazione introdotta riguarda la funzione $simLI$, che in questa versione può essere sintetizzata nel modo seguente:

$$sim(inst_1, inst_2) = \begin{cases} simLI(root_1, root_2) \cdot weight_0 \\ + \sum_{i=1}^{n} simL(subL_{1,i}, subL_{2,i}, 1) & \text{se } simLI(root_1, root_2) > 0 \\ \\ 0 & \text{altrimenti} \end{cases} \tag{5.5}$$

La formula differisce dalla 5.1 nei seguenti punti:

- $simLI(li_1, li_2)$ è una funzione che calcola il grado di similarità tra due nodi di tipo `LevelInstance`. Per maggiori dettagli riguardo a tale funzione, si faccia riferimento alla formula 5.7.

```
1  protected double calculateSimilarity(DpInstance dp1, DpInstance dp2) {
2    LevelInstance root1 = dp1.getRoot();
3    LevelInstance root2 = dp2.getRoot();
4    double similarity = 0;
5
6    if (simLI(root1, root2) > 0) {
7      similarity += simLI(root1, root2) * weights[ROOT_DEPTH];
8
9      for (Level l : root1.getSubLevels()) {
10       Level twinL = l.searchEquivalentIn(root2.getSubLevels());
11       similarity += simL(l, twinL, FIRST_LEVEL_DEPTH);
12     }
13   }
14
15   return similarity;
16 }
```

Listing 5.8: comparazione ricorsiva (versione 4)

Il listato 5.8 presenta l'implementazione Java della funzione contenuta nella formula 5.5.

Anche la formula 5.3, subisce delle modifiche:

$$\begin{aligned} simL(l_1, l_2, depth) &= \sum_{i=1}^{n} simLI(subLi_{1,i}, mostSim(subLi_{1,i}, subLis_2)) \\ &\quad \cdot \frac{weight_{depth}}{n} + \sum_{i=1}^{m} simL(subL_{1,i}, subL_{2,i}, depth + 1) \end{aligned} \tag{5.6}$$

La formula differisce dalla 5.3 nei seguenti punti:

- $n = \max(|subLis_1|, |subLis_2|)$ dove $subLis_i$ è l'insieme di nodi figlio di tipo `LevelInstance` del nodo l_i.
- $mostSim(subLi_{1,i}, subLis_2)$ è il nodo di tipo `LevelInstance`, appartenente a $subLis_2$, più "simile" a $subLi_{1,i}$. La similarità tra i due nodi è calcolata attraverso la funzione $simLI$ descritta nella formula 5.7.

Il listato 5.9 presenta l'implementazione Java della funzione contenuta nella formula 5.6.

A supporto delle formule discusse in questa sezione è stata inoltre introdotta la

```
1  protected double simL(Level l1, Level l2, int depth) {
2    Collection<LevelInstance> subLis1 = l1.getSubLevelInstances();
3    Collection<LevelInstance> subLis2 = l2.getSubLevelInstances();
4    double maxSubLiCount = Math.max(subLis1.size(), subLis2.size());
5    double subLevelsScore = 0;
6    double currentLevelScore = 0;
7    Vector<LevelInstance> checkedLis = new Vector<LevelInstance>();
8
9    for (LevelInstance subLi1 : subLis1) {
10     LevelInstance mostSimilarSubLi = mostSimilarOf(subLi1, subLis2,
           checkedLis);
11
12     if(mostSimilarSubLi!=null){
13       double mostSimilarSubLiScore = simLI(subLi1, mostSimilarSubLi);
14       currentLevelScore += (mostSimilarSubLiScore * weights[depth]) /
             maxSubLiCount;
15
16       for (Level l : subLi1.getSubLevels()) {
17         Level twinSubL = l.searchEquivalentIn(mostSimilarSubLi.
               getSubLevels());
18         subLevelsScore += simL(l, twinSubL, depth + 1);
19       }
20     }
21   }
22
23   return (subLevelsScore + currentLevelScore);
24 }
25
26 protected LevelInstance mostSimilarOf(LevelInstance levelInstance,
       Collection<LevelInstance> lis, Vector<LevelInstance> checkedLis) {
27   LevelInstance mostSimilar = null;
28   double bestScore = 0;
29
30   for (LevelInstance li : lis) {
31     double simScore = simLI(levelInstance, li);
32     if (simScore > bestScore) { // better similarity score ?
33       if(!checkedLis.contains(li)){ // already used as mostSimilarLi ?
34         mostSimilar = li;
35         bestScore= simScore;
36       }
37     }
38   }
39   checkedLis.add(mostSimilar);
40   return mostSimilar;
41 }
```

Listing 5.9: calcolo della similarità tra nodi di tipo “Level” (versione 4)

seguente nuova funzione:

$$simLI(li_1, li_2) = \frac{|sharedRoles|}{\max(|subRoles_1|, |subRoles_2|)} \tag{5.7}$$

Il listato 5.10 presenta l'implementazione Java della funzione contenuta nella formula 5.7.

```
1  protected double simLI(LevelInstance li1, LevelInstance li2) {
2    Collection<Role> li1Roles = li1.getRoles();
3    Collection<Role> li2Roles = li2.getRoles();
4    int maxRoles = Math.max(li1Roles.size(), li2Roles.size());
5    int sharedRoles = 0;
6
7    for (Role r1 : li1Roles) {
8      Role twinRole = r1.searchEquivalentIn(li2Roles);
9
10     if(twinRole != null){
11       sharedRoles++;
12     }
13   }
14
15   return (double)sharedRoles / (double)maxRoles;
16 }
```

Listing 5.10: calcolo della similarità tra nodi di tipo "LevelInstance" (versione 4)

5.2 Esempio di applicazione dell'algoritmo

Per meglio comprendere il comportamento dell'algoritmo di confronto finora descritto, verranno di seguito descritti i passi effettuati per la valutazione della coppia di istanze rappresentate nei listati 5.11 e 5.12.

```
1  ==========================================================
2  instance 1
3  ==========================================================
4  LI:root11  => Roles: {AF:a}
5    L:l_11 => ID: 1
6      LI:li_11 => Roles: {AP:b, AP:c}
7        L:l_12 => ID: 2
8          LI:li_12 => Roles: {CP:d, CP:e, CP:f}
9          LI:li_13 => Roles: {CP:g}
10     LI:li_14 => Roles: {AP:h, AP:i}
11       L:l_13 => ID: 2
12         LI:li_15 => Roles: {CP:l, CP:m, CP:n}
13     LI:li_16 => Roles: {AP:o}
14       L:l_14 => ID: 2
15         LI:li_17 => Roles: {CP:p}
16   L:l_15 => ID: 3
17     LI:li_18 => Roles: {CF:q}
```

Listing 5.11: istanza 1 di Abstract Factory (esempio)

```
1  ==========================================================
2  instance 2
3  ==========================================================
4  LI:root21 => Roles: {AF:a}
5    L:l_21 => ID: 1
6      LI:li_21 => Roles: {AP:b, AP:c}
7        L:l_22 => ID: 2
8          LI:li_22 => Roles: {CP:d, CP:e}
9          LI:li_23 => Roles: {CP:x}
10     LI:li_24 => Roles: {AP:h}
11       L:l_23 => ID: 2
12         LI:li_25 => Roles: {CP:l, CP:y}
13
14
15
16   L:l_24 => ID: 3
17     LI:li_26 => Roles: {CF:z}
```

Listing 5.12: istanza 2 di Abstract Factory (esempio)

Ogni nodo di tipo `LevelInstance` è formattato come segue:

```
LI:NodeLabel => Roles:{RoleName:RoleValue, ..}
```

Gli elementi elencati nella parte destra sono i ruoli associati all'istanza di livello. Due ruoli sono considerati equivalenti se possiedono lo stesso nome e valore.

Ogni nodo di tipo `Level` è specificato come segue:

```
L:NodeLabel => ID:IDValue
```

Due livelli sono considerati equivalenti se possiedono lo stesso valore specificato dall'attributo `IDValue`.

Le due istanze considerate presentano le seguenti caratteristiche:

- Entrambe si sviluppano su tre livelli.
- L'istanza 1 è composta da 9 nodi di tipo `LevelInstance`.
- L'istanza 2 è composta da 7 nodi di tipo `LevelInstance`.
- I nodi `root11, li_11` dell'istanza 1 sono equivalenti ai nodi `root21, li_21` dell'istanza 2.
- I nodi `li_12, li_14, li_15` dell'istanza 1 sono parzialmente equivalenti ai nodi `li_22, li_24, li_25` dell'istanza 2.
- I nodi `li_13, li_16, li_17, li_18` dell'istanza 1 non trovano equivalenti nell'istanza 2.
- I nodi `li_23, li_26` dell'istanza 2 non trovano equivalenti nell'istanza 1.

La procedura di confronto impiegata dall'algoritmo adottato nel progetto DPB, si compone dei seguenti passi:

1. assegnazione dei pesi ad ogni livello.
2. computazione ricorsiva della similarità dei singoli livelli.

Di seguito verranno applicati i passi enumerati alla coppia di istanze rappresentate nei listati 5.11 e 5.12.

5.2.1 Assegnazione dei pesi ad ogni livello

Applicando la funzione descritta nella formula 5.2, aggiornata secondo la modifica introdotta dalla formula 5.4, otteniamo i seguenti valori:

- $depthScore_0 = \log_{10}(3 - 0) + 1 = 1.48$
- $depthScore_1 = \log_{10}(3 - 1) + 1 = 1.3$
- $depthScore_2 = \log_{10}(3 - 2) + 1 = 1$

Da cui:
$totalpoints = 1.48 \cdot 1 + 1.3 \cdot 2 + 1 \cdot 1 = 5.08$

Otteniamo il risultato:

- $weight_0 = \dfrac{1.48}{5.08} = 0.29$
- $weight_1 = \dfrac{1.3}{5.08} = 0.26$
- $weight_2 = \dfrac{1}{5.08} = 0.2$

5.2.2 Computazione ricorsiva della similarità dei singoli livelli

Applicando la funzione descritta in 5.1, aggiornata secondo le modifiche introdotte nelle versioni successive dell'algoritmo, otteniamo i seguenti valori:

$sim(inst1, inst2) = simLI(root_{11}, root_{21}) \cdot weight_0 + simL(l_{11}, l_{21}, 1) + simL(l_{15}, l_{24}, 1)$

applicando la funzione ricorsiva simL, otteniamo:

$$\begin{aligned} simL(l_{11}, l_{21}, 1) &= (simLI(li_{11}, li_{21}) + simLI(li_{14}, li_{24}) \\ &\quad + simLI(li_{16}, null)) \cdot weight_1 \cdot 1/3 \\ &\quad + (simL(l_{12}, l_{22}, 2) + simL(l_{13}, l_{23}, 2) + simL(l_{14}, null, 2)) \\ simL(l_{15}, l_{24}, 1) &= simLI(li_{18}, li_{26}) \cdot weight_1 \cdot 1/1 \end{aligned} \tag{5.8}$$

proseguendo al livello di profondità 2, otteniamo:

$$\begin{aligned} &simLI(li_{11}, li_{21}) = 1 \\ &simLI(li_{14}, li_{24}) = 0.5 \\ &simLI(li_{16}, null) = 0 \\ &simL(l_{12}, l_{22}, 2) = (simLI(li_{12}, li_{22}) + simLI(li_{13}, li_{23})) \cdot weight_2 \cdot 1/2 \\ &simL(l_{13}, l_{23}, 2) = (simLI(li_{15}, li_{25})) \cdot weight_2 \cdot 1/1 \\ &simL(l_{14}, null, 2) = 0 \\ &simLI(li_{18}, li_{26}) = 0 \end{aligned} \tag{5.9}$$

dove:

$$\begin{aligned} simLI(li_{12}, li_{22}) &= 0.66 \\ simLI(li_{13}, li_{23}) &= 0 \\ simLI(li_{15}, li_{25}) &= 0.33 \end{aligned} \tag{5.10}$$

Risolvendo le equazioni in 5.9 otteniamo:

$$\begin{aligned} &simLI(li_{11}, li_{21}) = 1 \\ &simLI(li_{14}, li_{24}) = 0.5 \\ &simLI(li_{16}, null) = 0 \\ &simL(l_{12}, l_{22}, 2) = (0.66 + 0) \cdot 0.2 \cdot 1/2 = 0.066 \\ &simL(l_{13}, l_{23}, 2) = (0.33) \cdot 0.2 \cdot 1/1 = 0.066 \\ &simL(l_{14}, null, 2) = 0 \\ &simLI(li_{18}, li_{26}) = 0 \end{aligned} \tag{5.11}$$

e sostituendo in 5.8 otteniamo:

$$\begin{aligned} simL(l_{11}, l_{21}, 1) &= (1 + 0.5 + 0) \cdot 0.26 \cdot 1/3 + (0.066 + 0.066 + 0) = 0.262 \\ simL(l_{15}, l_{24}, 1) &= 0 \cdot 0.26 \cdot 1/1 = 0 \end{aligned} \tag{5.12}$$

per cui:

$sim(inst1, inst2) = 1 \cdot 0.29 + 0.262 + 0 = 0.552$

Secondo il risultato dell'algoritmo descritto in questo capitolo, il grado di somiglianza tra l'istanza 1 e l'istanza 2 è pari al **55.2%**.

5.3 Sperimentazione degli Algoritmi

In questa sezione verranno descritti i risultati sperimentali ottenuti dall'applicazione dei quattro algoritmi descritti nella prima parte di questo capitolo. Le figure contenute in questa sezione (5.5–5.14) rappresentano le dieci coppie di istanze di DP sulle quali sono stati applicati gli algoritmi citati.

Ogni istanza è rappresentata in forma compatta secondo uno schema ad albero, in cui ogni nodo rappresenta un oggetto `LevelInstance` a cui è associato un numero

variabile di oggetti di tipo `Role` pari al numero di lettere che compongono l'etichetta testuale (es: "OO", "OX", ecc...) ad esso associata. Ogni lettera rappresenta il valore associato all'oggetto di tipo `Role` che rappresenta. Per semplicità sono sempre state prese in considerazione istanze simmetriche nella struttura. Le coppie rappresentate nelle figure 5.13 e 5.14, a differenza di tutte le altre, presentano un'asimmetria nel numero di ruoli associati al nodo radice. Tale variazione è stata introdotta per dimostrare il supporto alla comparazione simmetrica della funzione di similarità.

La tabella 5.1 presenta i risultati ottenuti dall'applicazione delle quattro versioni dell'algoritmo di comparazione descritte in questo capitolo rispetto ai casi di studio qui delineati. L'ultima riga della tabella riporta la capacità di ogni algoritmo di effettuare un confronto simmetrico tra due istanze di DP (per maggiori informazioni si faccia riferimento alla sezione 5.1.3).

Dai risultati ottenuti nel corso di questa sperimentazione, è emerso un graduale miglioramento dei risultati al progredire dello sviluppo delle singole evoluzioni. I valori ottenuti dall'applicazione dell'algoritmo in versione 4, rispondono pienamente ai valori attesi, pronosticati dagli utenti coinvolti nella sperimentazione. É pertanto possibile concludere che i risultati prodotti dall'ultima variazione dell'algoritmo trattato in questo capitolo, siano sufficientemente verosimili da giustificare l'adozione dello stesso algoritmo all'interno dell'applicazione.

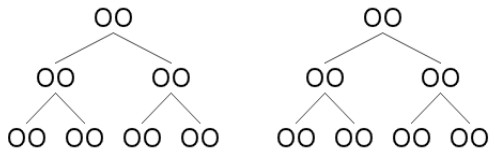

Figura 5.5: caso 1: istanze identiche. Valore di similarità atteso: 100%

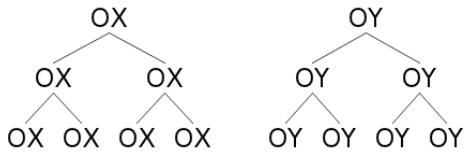

Figura 5.6: caso 2: Il 50% dei ruoli associati ad ogni nodo dell'albero di ciascuna istanza non trova equivalente nell'istanza reciproca. Valore atteso: 50%

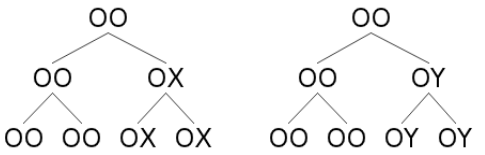

Figura 5.7: caso 3: Le istanze possiedono un ampio sotto-albero parzialmente dissimile. Valore atteso: circa 75%

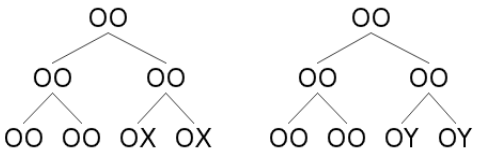

Figura 5.8: caso 4: 50% dei nodi foglia parzialmente dissimili. Valore atteso: circa 85-90%

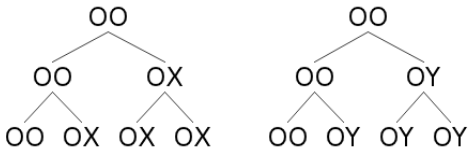

Figura 5.9: caso 5: Le istanze possiedono un ampio sotto-albero parzialmente dissimile e un nodo foglia parzialmente dissimile. Valore atteso: meno del 75%

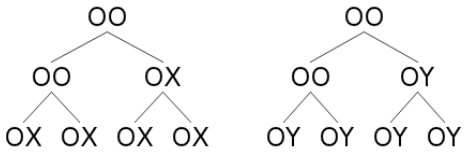

Figura 5.10: caso 6: tutti i nodi foglia e un nodo al secondo livello parzialmente dissimili. Valore atteso: tra 60% e 75%

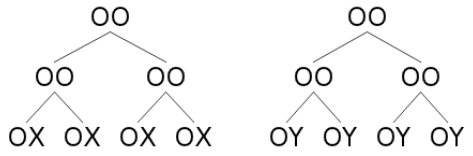

Figura 5.11: caso 7: tutti i nodi foglia parzialmente dissimili. Valore atteso: minore di 85% e maggiore di 70%

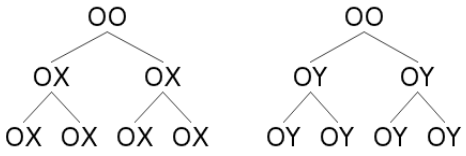

Figura 5.12: caso 8: 6 nodi su 7 parzialmente dissimili. Valore atteso: minore di 70% maggiore di 50%

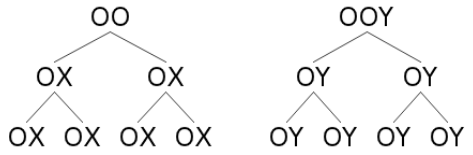

Figura 5.13: caso 9: istanze asimmetriche e parzialmente dissimili. radice dell'istanza 2 equivalenti al 66% alla radice dell'istanza 1. Valore atteso: maggiore del 50%

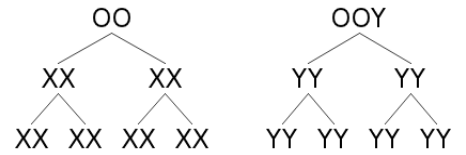

Figura 5.14: caso 10: istanze asimmetriche e quasi completamente dissimili (ad eccezione della radice). Valore atteso: maggiore dello 0%

	Versione 1	Versione 2	Versione 3	Versione 4	Valore atteso
Caso 1	1	1	1	1	1
Caso 2	0	0	0	0.50	0.50
Caso 3	0.64	0.59	0.59	0.80	0.75
Caso 4	0.82	0.75	0.75	0.88	0.85-0.90
Caso 5	0.55	0.47	0.47	0.73	< 0.75
Caso 6	0.45	0.34	0.34	0.67	0.60-0.75
Caso 7	0.64	0.50	0.50	0.75	0.70-0.85
Caso 8	0.27	0.18	0.18	0.59	0.50-0.70
Caso 9	0.27	0.18	0	0.53	> 0.50
Caso 10	0.27	0.18	0	0.12	> 0
Simmetrico	No	No	Si	Si	Si

Tabella 5.1: Risultati ottenuti su 9 casi di studio

Capitolo 6

Sperimentazione della piattaforma

Il ciclo di sviluppo della piattaforma DPB è stato caratterizzato da un'ampia sperimentazione interna e da varie fasi di "beta testing". In questo capitolo verranno sinteticamente descritti i contributi più significativi che hanno portato al rilascio dell'applicazione e i risultati ottenuti durante il suo periodo di esercizio.

6.1 Beta-testing e feedback

Per conoscere al meglio le esigenze degli utenti fruitori della piattaforma, si è scelto di coinvolgere sin dall'inizio un insieme di ricercatori attivi nel settore della DPD. Tali persone sono state invitate ad utilizzare la piattaforma nel corso del suo sviluppo. Buona parte di essi ha fornito suggerimenti, analisi e commenti che, nel loro insieme, hanno fornito un apporto indispensabile alla buona riuscita del progetto.

Tra le persone che hanno maggiormente contribuito alla fase sperimentale iniziale, è possibile citare:

- **Nikos Tsantalis**[91]: Assegnista di ricerca presso l'Università di Alberta (Edmonton, Canada). Ideatore e sviluppatore dello strumento DPD-tool (vedi sezione 1.2.2), co-ideatore di DPDX (vedi sezione 3.2.1) e ricercatore attivo nell'ambito della DPD.
- **Günter Kniesel**[65]: "senior lecturer" presso l'Università di Bonn (Germania). Co-ideatore di DPDX (vedi sezione 3.2.1), responsabile del progetto di sviluppo dello strumento DPJF (vedi sezione 1.2.1) e autore di numerose pubblicazioni relative all'ambito della DPD e della reverse engineering.

- **Yann-Gaël Guéhéneuc**[52]: Professore associato presso l'École Polytechnique di Montréal (Canada). Una delle personalità più attive nell'ambito della DPD. Co-ideatore di DPDX (vedi sezione 3.2.1), co-ideatore dello strumento Ptidej (vedi sezione 1.2.1), ideatore di P-MARt (vedi sezione 2.2) e prolifico autore di un gran numero di pubblicazioni relative alla DPD.

Il processo di sperimentazione è stato portato avanti attraverso il rilascio graduale di aggiornamenti e nuove funzionalità. Ad ogni rilascio sono state inviate delle email di notifica agli utenti coinvolti nel programma di "beta testing".

Oltre alle persone citate, si è anche scelto di coinvolgere nel progetto un gruppo di 18 studenti iscritti al corso "Ingegneria del software complementi" erogato nell'ambito del corso di laurea in informatica attivo presso l'Università degli studi di Milano-Bicocca. Agli studenti coinvolti è stato chiesto di valutare un certo insieme di istanze, fornendo una valida giustificazione del proprio giudizio. Gli spunti raccolti nel corso di questa esperienza sono risultati molto utili per comprendere al meglio il tipo di interazione ricercata dall'utente. I commenti e le domande poste dagli utenti sono inoltre servite ad individuare ulteriori possibilità di miglioramento a beneficio dell'usabilità dell'applicazione.

6.2 Traduttori

Al fine di ridurre la distanza tra gli strumenti di DPD attualmente resi pubblici e gli utenti della piattaforma DPB, è stato considerato opportuno sviluppare dei traduttori software che fossero in grado di trasformare l'output di un insieme scelto di strumenti di DPD in un file di input compatibile con il meccanismo di importazione della piattaforma DPB.

Gli strumenti di DPD scelti in questa fase sono:

- Web of Patterns (vedi sezione 1.2.1).
- DPD-tool (vedi sezione 1.2.2).

In aggiunta agli strumenti sopra citati, si è ritenuto interessante costruire un ulteriore traduttore per il repository P-MARt (vedi sezione 2.2). L'apporto dei risultati contenuti in P-MARt può essere considerato di notevole valore se valutato nell'ottica di un confronto rispetto ad istanze non verificate prodotte da altri strumenti.

L'introduzione e l'applicazione di questi traduttori ha consentito di popolare la piattaforma con un considerevole numero di analisi di sistema, offrendo agli utenti un vasto numero di istanze da visionare e valutare.

Si pensa che la pubblicazione del codice sorgente di questi programmi, consentirà anche ad altri sviluppatori di strumenti di DPD di creare un traduttore compatibile

con il proprio strumento senza dover stravolgere lo schema utilizzato per esportare i dati da esso prodotti.

6.3 Collaborazioni

In seguito alla prima fase di sperimentazione dell'applicazione, è stato possibile avviare una serie di collaborazioni, nate dall'interesse spontaneo delle persone coinvolte nella fase di beta-testing. Nel seguito di questa sezione verranno citate le più significative.

Günter Kniesel[65] (ottobre 2011 - presente): Con l'obiettivo di supportare il processo di sperimentazione dello strumento di DPD, da loro recentemente presentato, DPJF (vedi sezione 1.2.1), è stato definito un processo di integrazione bilaterale basato sui seguenti punti:

- estensione dello strumento DPJF al fine di supportare l'esportazione dei risultati in un formato compatibile con la piattaforma DPB.
- implementazione di una funzionalità di esportazione che consenta agli utenti della comunità di scaricare una copia di tutti i dati presenti nella piattaforma DPB nei formati XML e Prolog.
- sviluppo di un modulo di importazione che consenta a DPJF di utilizzare i dati esportati da DPB per calcolare i valori di precision e recall ottenuti dai risultati prodotti dallo strumento rispetto alle valutazioni presenti in DPB.
- estensione della funzionalità di importazione di DPB al fine di garantire il supporto del formato DPDX.

La realizzazione degli obiettivi concordati è attualmente in fase di attuazione.

Yann-Gaël Guéhéneuc[52] (ottobre 2010 - novembre 2011): Al fine di impiegare al meglio tutte le risorse rese disponibili dalla comunità scientifica attiva nell'ambito della DPD, è stato proposto di integrare i risultati contenuti nel data-set P-Mart (vedi sezione 2.2) all'interno della piattaforma DPB. La richiesta è stata accolta e discussa, portando all'effettiva integrazione delle due parti.

6.4 Stato di avanzamento

Di seguito verranno sinteticamente riportati alcuni dati che testimoniano le dimensioni attuali del progetto.

La piattaforma è popolata da:

- 2 strumenti di DPD: WOP (vedi sezione 1.2.1) e DPD-tool (vedi sezione 1.2.2).

- 1 dataset di istanze verificate: P-Mart (vedi sezione 2.2).
- più di 20 analisi di sistema.
- più di 700 istanze di DP.
- più di 160 valutazioni.

Sono presenti 36 utenti registrati.

Secondo le statistiche di accesso, sono stati registrati:

- più di 900 visite e 360 visitatori unici.
- più di 13.000 visualizzazioni di pagina.
- 15 minuti di tempo medio di permanenza sul sito.

Capitolo 7

Conclusioni e sviluppi futuri

In questa tesi è stata descritta un'applicazione in grado di fornire una misura della qualità di uno strumento di riconoscimento di DP sulla base delle valutazioni espresse in merito alle singole istanze che compongono i suoi risultati. Le valutazioni utilizzate come base della misurazione sono fornite dai membri di una comunità di utenti esperti aderenti al progetto. Ciò consente di ottenere una misurazione equilibrata e condivisa, basata sul principio della votazione democratica e del confronto.

L'applicazione presenta numerose caratteristiche innovative e può essere considerata unica nel suo genere sia per la natura delle funzionalità offerte sia per la considerevole usabilità che la contraddistingue.

La piattaforma è fondata su un robusto meta-modello per la rappresentazione dei DP. Tale meta-modello è stato concepito per essere sufficientemente generico e flessibile da poter essere adottato dalla maggior parte degli strumenti di DPD per la rappresentazione di qualsiasi DP. Esso garantisce un'ampia espressività, consente di definire delle strutture coerenti e di facile comprensione e potrebbe anche essere integrato ad altri meta-modelli di analoga concezione presentati in letteratura.

L'applicazione offre ai propri utenti gli strumenti sufficienti per analizzare e valutare le istanze di DP condivise dalla comunità, ricercare informazioni di proprio interesse, trarre conclusioni sull'effettiva qualità dei risultati prodotti da un determinato strumento e confrontare singole istanze di DP. Quest'ultima funzionalità è supportata da un algoritmo di nuova concezione sviluppato nell'ambito di questo progetto. Esso consente di quantificare il grado di somiglianza presente tra due istanze dello stesso DP, sulla base di un confronto analitico delle strutture ad albero adottate per la loro rappresentazione.

La soluzione finale sviluppata nell'ambito di questo progetto, è stata ampiamente testata e raffinata. Le persone coinvolte nelle prime fasi di sperimentazione del prodotto, si sono dimostrate interessate e attive. Grazie alle loro critiche e suggerimenti, è stato possibile costruire un servizio efficace e allo stesso tempo usabile e di facile

fruizione. Alcune collaborazioni, nate dall'interesse spontaneo di alcuni ricercatori esperti di fama internazionale appartenenti al contesto scientifico di riferimento del progetto, hanno dato luogo ad interessanti contributi che hanno reso l'applicazione più efficace e vicina alle esigenze degli utenti.

Sulla base dei dati raccolti e dei commenti ricevuti, si ritiene che sia importante dirigere il processo di evoluzione verso i seguenti obiettivi:

1. **Semplificare il processo di importazione dei risultati da strumenti di DPD**.
   Questo obiettivo potrebbe essere raggiunto attraverso un'opportuna combinazione di alcune delle seguenti strategie:

   - Estensione della compatibilità verso altri meta-modelli attualmente diffusi e impiegati.
   - Implementazione di un "web service" che consenta di automatizzare il processo di caricamento dei risultati di uno strumento di DPD.
   - Definizione di un meta-modello più generale, comune a tutta la comunità attiva nell'ambito della DPD (ad esempio: DPDX-esteso, descritto nella sezione 3.2.2).
   - Sviluppo di un modulo di traduzione in grado di interpretare meta-modelli non supportati dalla piattaforma attraverso l'applicazione di regole di "mapping" definite dall'utente. La produzione di tali regole potrebbe essere resa possibile grazie all'uso di una procedura semplificata che supporti il soggetto interessato durante il processo di definizione e convalida.

2. **Utilizzare i dati provenienti da valutazioni precedenti per produrre inferenze statistiche**.
   La conoscenza derivata dalle valutazioni associate ad istanze presenti sulla piattaforma potrebbe essere utilizzata nella formulazione di ulteriori valutazioni circa il grado di correttezza attribuibile ad altre istanze dello stesso tipo. Tale valutazione potrebbe essere inferita sulla base del grado di similarità presente tra le istanze prese in considerazione.

3. **Ideare nuove forme di presentazione dei dati a supporto dell'analisi**.
   Al fine di semplificare l'interazione tra utente e applicazione e ridurre i tempi necessari ad interpretare la grande quantità di informazioni presenti nella piattaforma, potrebbe essere utile studiare nuovi metodi per la fruizione e la rappresentazione dei dati presenti sulla piattaforma. Per supportare questo obiettivo sarebbe interessante riprogettare, ottimizzare o coniugare alcune delle viste utilizzate per la rappresentazione delle istanze di DP. Inoltre si pensa che possa essere di grande aiuto supportare l'utente nella propria analisi fornendo accesso ai dati presenti in DPB attraverso un plug-in per Eclipse[39]. Tale plug-in dovrebbe essere in grado di trasferire tutti i dati presenti nella

piattaforma DPB, in un ambiente più ricco di funzionalità e familiare all'utente. Il risultato ottenuto da una simile evoluzione, consentirebbe un analisi più efficace e veloce delle istanze di DP. La maggiore interattività dell'ambiente adottato e la disponibilità di strumenti di analisi complementari, porterebbe ad ottenere effetti positivi anche sulla precisione delle valutazioni espresse.

4. **Migliorare l'algoritmo di similarità descritto nel capitolo 5**.
   Per consentire una corretta valutazione delle istanze e garantire un adeguato supporto agli utenti che intendano confrontare diverse analisi di progetto, è indispensabile garantire un meccanismo di comparazione che esprima al meglio le differenze presenti tra coppie di istanze di DP. Il codice sorgente dell'algoritmo attualmente utilizzato, è stato già condiviso con diversi ricercatori attivi nell'ambito di ricerca della DPD. Per raggiungere l'obiettivo di creare un algoritmo di comprovata qualità, si intende documentare e diffondere tali informazioni ad un campione più ampio di utenti esperti. I commenti e suggerimenti raccolti forniranno validi spunti di miglioramento e interessanti argomenti di discussione.

Oltre alle modifiche sopra discusse, si prevede inoltre di estendere e arricchire la documentazione attualmente presente a supporto dell'utente e di fornire ulteriori specifiche tecniche a chiunque sia interessato ad approfondire il funzionamento dell'applicazione.

Appendice A

Il formato DPBXS

Per consentire un corretto inserimento dei dati nella piattaforma, è stato necessario definire un formato standard per la specifica delle informazioni che compongono un'analisi di sistema.

La specifica formale di tale formato è sintetizzata nel file di definizione XSD riportato nel listato A.1.

```
1  <!--?xml version="1.0" encoding="UTF-8" ?-->
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3      <xs:element name="analysis"> <xs:complextype>
4        <xs:sequence>
5            <xs:element ref="pattern" minoccurs="1" maxoccurs="unbounded"
                 />
6          </xs:sequence>
7      </xs:complextype> </xs:element>
8      <xs:element name="pattern"> <xs:complextype>
9        <xs:sequence>
10           <xs:element ref="patternInstance" minoccurs="0" maxoccurs="
                 unbounded"/>
11         </xs:sequence>
12
13         <!-- Name of the pattern. Must match one of the names listed at
                 : DPBWeb/faces/Doc_DpDef.jsp -->
14         <!-- example: "AbstractFactory" -->
15         <xs:attribute name="name" type="xs:string" use="required"></xs:
               attribute>
16     </xs:complextype> </xs:element>
17     <xs:element name="patternInstance"> <xs:complextype>
18       <xs:sequence>
19           <xs:element ref="role" minoccurs="1" maxoccurs="unbounded"/>
20           <xs:element ref="level" minoccurs="0" maxoccurs="unbounded"/>
21       </xs:sequence>
22     </xs:complextype> </xs:element>
23     <xs:element name="level"> <xs:complextype>
```

```
24        <xs:sequence>
25            <xs:element ref="levelInstance" minoccurs="1" maxoccurs="
                  unbounded"/>
26        </xs:sequence>
27      </xs:complextype> </xs:element>
28      <xs:element name="levelInstance"> <xs:complextype>
29        <xs:sequence>
30            <xs:element ref="role" minoccurs="1" maxoccurs="unbounded"/>
31            <xs:element ref="level" minoccurs="0" maxoccurs="unbounded"/>
32          </xs:sequence>
33    </xs:complextype> </xs:element>
34    <xs:element name="role"> <xs:complextype>
35        <xs:sequence>
36        <xs:element ref="location" minoccurs="0" maxoccurs="1"/>
37            <xs:element ref="comment" minoccurs="0" maxoccurs="1"/>
38        </xs:sequence>
39
40      <!-- Role name. Must match one of the role names associated to the
            chosen design pattern. -->
41      <!-- To find the list of all legal names, go to DPBWeb/faces/
            Doc_DpDef.jsp and select a pattern. -->
42      <!-- All roles are required. If your tool does not provide that
            kind of information, just set role.value="NONE". -->
43      <!-- example: "AbstractProduct" -->
44      <xs:attribute name="name" type="xs:string" use="required"></xs:
            attribute>
45
46      <!-- Role value. Can be a class, method or field. -->
47      <!-- Must be specified using a full reference identifier and cannot
             contain spaces. -->
48      <!-- If no value can be specified, use "NONE" -->
49      <!-- example(class): "org.test.MyClass" -->
50      <!-- example(nested class): "org.test.MyClass.MySubClass" -->
51      <!-- example(method): "org.test.MyClass.method(int,java.lang.String
            )" -->
52      <!-- example(field): "org.test.MyClass.field" -->
53      <!-- example(none): "NONE" -->
54      <xs:attribute name="value" type="xs:string" use="required"></xs:
            attribute>
55      </xs:complextype> </xs:element>
56    <!-- describes where the specified role.value element can be found in
          the code base -->
57    <xs:element name="location"> <xs:complextype>
58      <!-- unix formatted file path. Will be combined with the [Svn base
            URI] value of the chosen project. -->
59      <!-- [Svn base URI] can be found in the upload page, selecting the
            desiered project. -->
60      <!-- [Svn base URI] + [Role.filePath] must be a valid source code
            reference. -->
61      <!-- example: "a/b/c/Test.java" -->
62      <xs:attribute name="file" type="xs:string" use="required"></xs:
            attribute>
63
```

```
64      <!-- code line where the role value element can be found in the
            above specified file. -->
65      <!-- example: "987" -->
66      <xs:attribute name="line" type="xs:integer" use="optional"></xs:
            attribute>
67    </xs:complextype> </xs:element>
68
69      <!-- free annotation field. Should be used to provide more
            information in order to help users in the task of reviewing the
            instance. -->
70      <!-- example: "Accuracy = 0.9" -->
71      <xs:element name="comment" type="xs:string"/>
72  </xs:schema>
```

Listing A.1: Schema XSD associato al formato DPBXS

Ogni analisi di sistema si compone delle seguenti entità:

- `analysis`: nodo radice.
- `pattern`: aggregatore di istanze dello stesso tipo di pattern. Il nome del pattern è specificato dall'attributo `name`. Tale nome deve corrispondere ad uno dei nomi associati alle definizione di DP supportate dalla piattaforma. Una lista completa delle definizioni attualmente supportate, è consultabile nella sezione Documentation della piattaforma [19].
- `patternInstance`: rappresenta una singola istanza di design pattern.
- `level`: corrisponde all'entità `Level` descritta nella sezione 3.1. Deve contenere uno o più elementi di tipo `levelInstance`.
- `levelInstance`: corrisponde all'entità `LevelInstance` descritta nella sezione 3.1. Deve contenere uno o più elementi di tipo `role`, e può contenere un numero variabile di elementi di tipo `Level`.
- `role`: corrisponde all'entità `Role` descritta nella sezione 3.1. Può contenere al più un elemento di tipo `location` e uno di tipo `comment`. É caratterizzato da un attributo `name`, che descrive il tipo di ruolo rappresentato, e un attributo `value`, che descrive il nome dell'entità di codice a cui il ruolo fa riferimento. L'attributo `name` deve corrispondere al nome di uno dei ruoli presenti nella definizione del pattern specificato nell'elemento `pattern`. L'attributo `value` deve essere codificato secondo il formato standard adottato per la denominazione delle entità di codice nel linguaggio di programmazione Java. I valori ammessi possono essere dei seguenti tipi: classe, metodo, attributo. Se necessario, è possibile indicare come valore dell'attributo la stringa `%NONE%`.
- `location`: descrive la posizione dell'entità di codice referenziata dal ruolo, attraverso i valori dei propri attributi (`file` e `line`). L'attributo `file`, combinato con il prefisso specificato in fase di caricamento (SVN base URI), de-

ve formare un indirizzo URL valido che consenta di referenziare la risorsa specificata.

- `comment`: aggiunge un'informazione testuale sintetica a beneficio dell'analisi. Può essere, ad esempio, utilizzato per esprimere il grado di confidenza riportato dallo strumento nell'attribuzione del ruolo.

Nel listato A.3, si mostra un esempio di analisi di sistema contenente un'istanza di DP. L'istanza rappresentata, è conforme alla definizione del pattern Abstract Factory, sintetizzata nel listato A.2.

```
1 AbstractFactory
2   - AbstractProduct
3     -- ConcreteProduct
4   - ConcreteFactory
5     -- ConcreteFactory_createProduct
6   - AbstractFactory_createProduct
```

Listing A.2: Struttura della definizione del pattern Abstract Factory

```
<analysis><pattern name="AbstractFactory"><patterninstance>
  <level>
      <levelinstance>
      <role name="AbstractFactory" value="com.foo.MyClass1">
        <location file="com/foo/MyClass1.java" line="13"/>
        <comment>Some text</comment>
      </role>
      <level>
        <levelinstance>
          <role name="AbstractProduct" value="com.foo.MyClass2">
            <location file="com/foo/MyClass2.java" line="53"/>
            <comment>Some text</comment>
          </role>
          <level>
            <levelinstance>
              <role name="ConcreteProduct" value="com.foo.MyClass3">
                <location file="com/foo/MyClass3.java" line="93"/>
                <comment>Some text</comment>
              </role>
            </levelinstance>
          </level>
        </levelinstance>
      </level>
      <level>
        <levelinstance>
          <role name="ConcreteFactory" value="com.foo.MyClass4">
            <location file="com/foo/MyClass4.java" line="93"/>
            <comment>Some text</comment>
          </role>
          <level>
            <levelinstance>
              <role name="ConcreteFactory_createProduct" value="com.foo
                  .MyClass.myMethod1(java.lang.String,int)">
                <location file="com/foo/MyClass5.java" line="133"/>
                <comment>Some text</comment>
              </role>
            </levelinstance>
          </level>
        </levelinstance>
      </level>
      <level>
        <levelinstance>
          <role name="AbstractFactory_createProduct" value="com.foo.
              MyClass.myMethod2(java.lang.String,int)">
            <location file="com/foo/MyClass6.java" line="133"/>
            <comment>Some text</comment>
          </role>
        </levelinstance>
      </level>
    </levelinstance>
  </level>
</patterninstance></pattern></analysis>
```

Listing A.3: Esempio di istanza specificata secondo il formato DPBXS

Bibliografia

[1] Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, and Narendra Jussien. Instantiating and detecting design patterns: Putting bits and pieces together. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, page 166, Washington, DC, USA, 2001. IEEE Computer Society.

[2] Giuliano Antoniol, Roberto Fiutem, and Luca Cristoforetti. Design pattern recovery in object-oriented software. In *IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension*, page 153, Washington, DC, USA, 1998. IEEE Computer Society.

[3] Francesca Arcelli Fontana. Essere. `http://essere.disco.unimib.it/`.

[4] Francesca Arcelli Fontana, Andrea Caracciolo, and Marco Zanoni. A benchmark for design pattern detection tools: a community driven approach. *ERCIM News*, January 2012.

[5] Francesca Arcelli Fontana, Andrea Caracciolo, and Marco Zanoni. Dpb: A benchmark for design pattern detection tools. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering CSMR 2012*, Szeged, Hungary, March 2012.

[6] Francesca Arcelli Fontana and Luca Cristina. Enhancing software evolution through design pattern detection. In *Software Evolvability, 2007 Third International IEEE Workshop on*, pages 7–14, Oct. 2007.

[7] Francesca Arcelli Fontana, Stefano Maggioni, and Claudia Raibulet. Design patterns: a survey on their micro-structures. *Journal of Software Maintenance and Evolution: Research and Practice*, pages n/a–n/a, 2011.

[8] Francesca Arcelli Fontana, Cristian Tosi, and Marco Zanoni. A benchmark proposal for design pattern detection. In *FAMOOSr 2008: Proceedings of 2nd Workshop on FAMIX and Moose in Reengineering*, Oct. 2008.

[9] Francesca Arcelli Fontana and Marco Zanoni. Marple project. `http://essere.disco.unimib.it/reverse/Marple.html`.

[10] Francesca Arcelli Fontana, Marco Zanoni, and Andrea Caracciolo. A benchmark platform for design pattern detection. In *Proceedings of The Second International Conferences on Pervasive Patterns and Applications PATTERNS 2010*, pages 42–47, Lisbon, Portugal, November 2010. IARIA, Think Mind.

[11] Francesca Arcelli Fontana, Marco Zanoni, Riccardo Porrini, and Mattia Vivanti. A model proposal for program comprehension. In *Proceedings of the 16th International Conference on Distributed Multimedia Systems, DMS 2010, October 14-16, 2010, Hyatt Lodge at McDonald s Campus, Oak Brook, Illinois, USA*, pages 23–28, 2010.

[12] Tim Armes, Dirk Thomas, and Quinn Taylor. Websvn. `http://www.websvn.info/`.

[13] Angel Asencio, Sam Cardman, David Harris, and Ellen Laderman. Relating expectations to automatically recovered design patterns. In *WCRE '02: Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02)*, page 87, Washington, DC, USA, 2002. IEEE Computer Society.

[14] Jagdish Bansiya. Automating design-pattern identification. *Dr Dobbs Journal*, 1998.

[15] Ian Bayley and Hong Zhu. Formalising design patterns in predicate logic. In *Software Engineering and Formal Methods, 2007. SEFM 2007. Fifth IEEE International Conference on*, pages 25 –36, sept. 2007.

[16] Dirk Beyer, Andreas Noack, and Claus Lewerentz. Simple and efficient relational querying of software structures. In *WCRE '03: Proceedings of the 10th Working Conference on Reverse Engineering*, page 216, Washington, DC, USA, 2003. IEEE Computer Society.

[17] Marcel Birkner. Objected-oriented design pattern detection using static and dynamic analysis in java software. Master's thesis, university of applied sciences bonn-rhein-sieg, sankt augustin, germany, 2007.

[18] Alex Blewitt, Alan Bundy, and Ian Stark. Automatic verification of java design patterns. In *ASE '01: Proceedings of the 16th IEEE international conference on Automated software engineering*, page 324, Washington, DC, USA, 2001. IEEE Computer Society.

[19] Andrea Caracciolo. Dpb: list of supported dp definitions. `http://essere.disco.unimib.it:8080/DPBWeb/faces/Doc_DpDef.jsp`.

[20] Elliot J. Chikofsky and James H. Cross II. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[21] Glassfish community. Glassfish. `http://glassfish.java.net`.

[22] Glassfish community. Toplink essentials. http://glassfish.java.net/javaee5/persistence.

[23] Jikes community. Jikes. http://jikes.sourceforge.net/.

[24] Gennaro Costagliola, Andrea De Lucia, Vincenzo Deufemia, Carmine Gravino, and Michele Risi. Design pattern recovery by visual language parsing. In *CSMR '05: Proceedings of the Ninth European Conference on Software Maintenance and Reengineering*, pages 102–111, Washington, DC, USA, 2005. IEEE Computer Society.

[25] Andrea De Lucia, Vincenzo Deufemia, Carmine Gravino, and Michele Risi. An eclipse plug-in for the detection of design pattern instances through static and dynamic analysis. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–6, Washington, DC, USA, 2010. IEEE Computer Society.

[26] Università degli Studi di Milano-Bicocca. Homepage. http://www.unimib.it.

[27] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. *Object-Oriented Reengineering Patterns*. Square Bracket Associates, 2008.

[28] Jens Dietrich and Chris Elgar. Owl ontology used by wop. http://www-ist.massey.ac.nz/wop/20050204/owldoc/index.html.

[29] Jens Dietrich and Chris Elgar. A formal description of design patterns using owl. In *Software Engineering Conference, 2005. Proceedings. 2005 Australian*, pages 243 – 250, march-1 april 2005.

[30] Jens Dietrich and Chris Elgar. Towards a web of patterns. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5:108–116, June 2007.

[31] Jing Dong, Dushyant S. Lad, and Yajing Zhao. Dp-miner: Design pattern discovery using matrix. In *ECBS '07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pages 371–380, Washington, DC, USA, 2007. IEEE Computer Society.

[32] Jing Dong, Yajing Zhao, and Tu Peng. Architecture and design pattern discovery techniques - a review. In *Software Engineering Research and Practice*, pages 621–627, 2007.

[33] Gregoire Dupe and Hugo Bruneliere. Modisco. http://www.eclipse.org/MoDisco/.

[34] Félix Agustín Castro Espinoza, Gustavo Núñez Esquer, and Joel Suárez Cansino. Automatic design patterns identification of c++ programs. In *EurAsia-ICT '02: Proceedings of the First EurAsian Conference on Information and Communication Technology*, pages 816–823, London, UK, 2002. Springer-Verlag.

[35] Rudolf Ferenc, Arpad Beszedes, Lajos Jenő Fülöp, and Janos Lele. Design pattern mining enhanced by machine learning. In *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance*, pages 295–304, Washington, DC, USA, 2005. IEEE Computer Society.

[36] Rudolf Ferenc, Arpad Beszedes, Mikko Tarkiainen, and Tibor Gyimothy. Columbus - reverse engineering tool and schema for c++. In *Proceedings of the International Conference on Software Maintenance (ICSM'02)*, page 172, Washington, DC, USA, 2002. IEEE Computer Society.

[37] Lajos Jenő Fülöp. Deebee. `http://www.inf.u-szeged.hu/designpatterns/`.

[38] Lajos Jenő Fülöp. Deebee: Upload format. `https://www.sed.hu/projects/dpe/wiki/ExampleCsvFormatum`.

[39] Eclipse Foundation. Eclipse. `http://www.eclipse.org/`.

[40] The Apache Software Foundation. Apache http server. `http://httpd.apache.org`.

[41] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[42] Lajos Jenő Fülöp, Rudolf Ferenc, and Tibor Gyimothy. Towards a benchmark for evaluating design pattern miner tools. In *CSMR '08: Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, pages 143–152, Washington, DC, USA, 2008. IEEE Computer Society.

[43] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[44] Tudor Gîrba. *The Moose Book*. Self Published, 2010.

[45] Orla Greevy. *Enriching Reverse Engineering with Feature Analysis*. PhD thesis, University of Bern, 2007.

[46] Object Management Group. Catalog of omg modernization specifications. `http://www.omg.org/technology/documents/modernization_spec_catalog.htm`.

[47] Object Management Group. Kdm. `http://www.omg.org/technology/kdm/index.htm`.

[48] Object Management Group. Kdm 1.0 annotated reference. `http://kdmanalytics.com/kdmspec/index.php`.

[49] Object Management Group. Mof. `http://www.omg.org/mof/`.

[50] Object Management Group. Organization homepage. http://www.omg.org.

[51] Yann-Gaël Guéhéneuc, Jean-Yves Guyomarc'H, and Houari Sahraoui. Improving design-pattern identification: a new approach and an exploratory study, March 2010.

[52] Yann-Gaël Guéhéneuc. Personal homepage. http://www.yann-gael.gueheneuc.net/.

[53] Yann-Gaël Guéhéneuc. P-mart: Pattern-like micro architecture repository. In *Proceedings of the 1st EuroPLoP Focus Group on Pattern Repositories*, Jul. 2007.

[54] Yann-Gaël Guéhéneuc and Giuliano Antoniol. Demima: A multilayered approach for design pattern identification. *IEEE Transactions on Software Engineering*, 34:667–684, 2008.

[55] Yann-Gaël Guéhéneuc, Houari Sahraoui, and Farouk Zaidi. Fingerprinting design patterns. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering*, pages 172–181, Washington, DC, USA, 2004. IEEE Computer Society.

[56] Manjari Gupta, Akshara Pande, and A. K. Tripathi. Design patterns detection using sop expressions for graphs. *SIGSOFT Softw. Eng. Notes*, 36:1–5, January 2011.

[57] Shinpei Hayashi, Junya Katada, Ryota Sakamoto, Takashi Kobayashi, and Motoshi Saeki. Design pattern detection by using meta patterns. *IEICE - Trans. Inf. Syst.*, E91-D:933–944, April 2008.

[58] Dirk Heuzeroth, Thomas Holl, Gustav Hogstrom, and Welf Lowe. Automatic design pattern detection. In *Program Comprehension, 2003. 11th IEEE International Workshop on*, pages 94–103, 2003.

[59] Ric Holt, Andy Schürr, Susan Elliott Sim, and Andreas Winter. Gxl. http://www.gupro.de/GXL/.

[60] Richard C. Holt, Ahmed E. Hasan, Bruno Laguë, Sébastien Lapierre, and Charles Leduc. E/r schema for the datrix c/c++/java exchange format. In *Reverse Engineering, 2000. Proceedings. Seventh Working Conference on*, pages 284 –286, 2000.

[61] Mei Hong, Tao Xie, and Fuqing Yang. Jbooret: an automated tool to recover oo design and source models. In *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*, pages 71 –76, 2001.

[62] IBM. Rational software architect. http://www.ibm.com/developerworks/rational/products/rsa/.

[63] Olivier Kaczor, Yann-Gaël Guéhéneuc, and Sylvie Hamel. Efficient identification of design patterns with bit-vector algorithm. In *CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering*, pages 175–184, Washington, DC, USA, 2006. IEEE Computer Society.

[64] Rudolf K. Keller, Reinhard Schauer, Sébastien Robitaille, and Patrick Pagé. Pattern-based reverse-engineering of design components. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 226–235, New York, NY, USA, 1999. ACM.

[65] Günter Kniesel. Personal homepage. `http://www.iai.uni-bonn.de/~gk/gk/`.

[66] Günter Kniesel, Alexander Binun, Peter Hegedus, Lajos Jenő Fülöp, Alexander Chatzigeorgiou, Yann-Gaël Guéhéneuc, and Nikolaos Tsantalis. Dpdx–towards a common result exchange format for design pattern detection tools. In *Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering*, CSMR '10, pages 232–235, Washington, DC, USA, 2010. IEEE Computer Society.

[67] Günter Kniesel, Alexander Binun, Péter Hegedusy, Lajos Jeno Fülöp, Alexander Chatzigeorgiouz, Yann-Gaël Guéhéneuc, and Nikolaos Tsantalis. Dpdx project homepage. `http://sewiki.iai.uni-bonn.de/dpdx/start`.

[68] Günter Kniesel, Alexander Binun, Péter Hegedusy, Lajos Jeno Fülöp, Alexander Chatzigeorgiouz, Yann-Gaël Guéhéneuc, and Nikolaos Tsantalis. A common exchange format for design pattern detection tools. Technical report, 2009.

[69] Christian Kramer and Lutz Prechelt. Design recovery by automated search for structural design patterns in object-oriented software. In *WCRE '96: Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96)*, page 208, Washington, DC, USA, 1996. IEEE Computer Society.

[70] Timothy C. Lethbridge, Erhard Plödereder, Sander Tichelaar, Claudio Riva, Panos Linos, and Sergei Marchenko. The dagstuhl middle model (dmm). `http://www.site.uottawa.ca/~tcl/dmm/DMMDescriptionV0006.pdf`.

[71] Timothy C. Lethbridge, Sander Tichelaar, and Erhard Ploedereder. The dagstuhl middle metamodel: A schema for reverse engineering. *Electronic Notes in Theoretical Computer Science*, 94:7 – 18, 2004. Proceedings of the International Workshop on Meta-Models and Schemas for Reverse Engineering (ateM 2003).

[72] Jacqueline A. McQuillan and James F. Power. Experiences of using the dagstuhl middle metamodel for defining software metrics. In *Proceedings of the 4th international symposium on Principles and practice of programming in Java*, PPPJ '06, pages 194–198, New York, NY, USA, 2006. ACM.

[73] Hausi A. Müller, Jens H. Jahnke, Dennis B. Smith, Margaret-Anne Storey, Scott R. Tilley, and Kenny Wong. Reverse engineering: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 47–60, New York, NY, USA, 2000. ACM.

[74] Jörg Niere, Wilhelm Schäfer, Jörg P. Wadsack, Lothar Wendehals, and Jim Welsh. Towards pattern-based design recovery. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 338–348, New York, NY, USA, 2002. ACM.

[75] Oscar Nierstrasz, Stéphane Ducasse, and Tudor Gĭrba. Moose project. `http://www.moosetechnology.org/`.

[76] Oscar Nierstrasz, Stéphane Ducasse, and Tudor Gĭrba. The story of moose: an agile reengineering environment. *SIGSOFT Softw. Eng. Notes*, 30:1–10, September 2005.

[77] Oracle. Javaserver faces. `http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html`.

[78] Oracle. Mysql. `http://www.mysql.com`.

[79] Wolfgang Pree. *Design patterns for object-oriented software development*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[80] Ghulam Rasool, Ilka Philippow, and Patrick Mäder. Design pattern recovery based on annotations. *Adv. Eng. Softw.*, 41:519–526, April 2010.

[81] Jochen Seemann and Jürgen Wolff von Gudenberg. Pattern-based design recovery of java software. In *SIGSOFT '98/FSE-6: Proceedings of the 6th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 10–16, New York, NY, USA, 1998. ACM.

[82] Nija Shi and Ronald A. Olsson. Reverse engineering of design patterns from java source code. In *ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 123–134, Washington, DC, USA, 2006. IEEE Computer Society.

[83] Edgewall Software. Trac. `http://trac.edgewall.org/`.

[84] Krzysztof Stencel and Patrycja Wegrzynowicz. Detection of diverse design pattern variants. In *Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific*, pages 25 –32, dec. 2008.

[85] David Stotts, Jason McC. Smith, and Jason Mcc Smith. Spqr: Flexible automated design pattern extraction from source code. In *In 18th IEEE Intl Conf on Automated Software Engineering*, pages 215–224. IEEE Computer Society Press, 2003.

[86] Detlef Streitferdt, Christian Heller, and Ilka Philippow. Searching design patterns in source code. In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference*, pages 33–34, Washington, DC, USA, 2005. IEEE Computer Society.

[87] Sander Tichelaar. Famix 2.2. `http://www.moosetechnology.org/docs/others/famix2.2`.

[88] Sander Tichelaar. Famix 3.0. `http://www.moosetechnology.org/docs/others/famix3.0`.

[89] Sander Tichelaar. *Modeling Object-Oriented Software for Reverse Engineering and Refactoring*. PhD thesis, University of Bern, 2001.

[90] Nikolaos Tsantalis and Spyros T. Halkidis. Design pattern detection using similarity scoring. *IEEE Trans. Softw. Eng.*, 32(11):896–909, 2006. Member-Chatzigeorgiou, Alexander and Member-Stephanides, George.

[91] Nikos Tsantalis. Personal homepage. `http://java.uom.gr/~nikos/`.

[92] Leonardo Vanneschi, Marco Tomassini, Philippe Collard, and Manuel Clergue. Fitness distance correlation in structural mutation genetic programming. In *Proceedings of the 6th European conference on Genetic programming*, EuroGP'03, pages 455–464, Berlin, Heidelberg, 2003. Springer-Verlag.

[93] Wei Wang and Vassilios Tzerpos. Design pattern detection in eiffel systems. In *WCRE '05: Proceedings of the 12th Working Conference on Reverse Engineering*, pages 165–174, Washington, DC, USA, 2005. IEEE Computer Society.

[94] Lothar Wendehals. Improving design pattern instance recognition by dynamic analysis. In *Proc. of the ICSE 2003 Workshop on Dynamic Analysis (WODA), Portland, USA*, pages 29–32, 2003.

[95] Zhi-Xiang Zhang, Qing-Hua Li, and Ke-Rong Ben. A new method for design pattern mining. In *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, volume 3, pages 1755–1759 vol.3, Aug. 2004.