UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea in Informatica

UN PLUGIN PER ECLIPSE
EDP DETECTOR

SUPERVISORI: - Chiar.ma Prof.ssa Francesca ARCELLI
- Dott. Luigi UBEZIO

Relazione della prova finale di:
Stefano FANTIN
Matr. n. 041172

Anno Accademico 2004/2005

UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea in Informatica

UN PLUGIN PER ECLIPSE

EDP DETECTOR

SUPERVISORI: - Chiar.ma Prof.ssa Francesca Arcelli
- dott. Luigi Ubezio

Relazione della prova finale di:
Stefano Fantin
Matr. n. 041172

Anno Accademico 2004/2005

Dedica:
Ai miei genitori Mario
e Giancarla

Ringraziamenti:
alla professoressa Arcelli che mi ha dato la possibilità di sviluppare questa tesi, al mio correlatore dott. Luigi Ubezio che mi ha sempre seguito dandomi (a suo modo) la possibilità di essere arrivato dove sono ed a tutti gli stagisti e ricercatori perché mi hanno fatto passare un gioioso momento nel laboratorio SAL dell'Università degli Studi di Milano Bicocca.

Indice

Introduzione

Questa tesi verte sugli EDP (Elemental Design Pattern) e più in particolare su come gestirli all'interno dell'ambiente di lavoro Eclipse. Siccome tale lavoro ha avuto forti connotazioni pratiche si ritiene utile al fine che il lettore capisca a fondo tutti i passaggi effettuati svolgere un'adeguata introduzione sulla Reverse Engineering e Desing Pattern quindi nei primi due capitoli trattare la teoria che sta alla base del lavoro svolto per poi esaminare nell'ultimo capitolo la parte implementativa. Questa è stata la decisione dell'autore di questa tesi in quanto sostiene la seguente frase "bisogna conoscere per dominare".
Inizio la parte di teoria. Il Reverse Engineering (RE) è *un insieme di teorie, modelli, metodi, tecniche e tecnologie per:*

• il progetto e l'implementazione di processi di estrazione ed astrazione di informazioni da componenti di un sistema software esistente e la produzione di nuovi componenti ad un livello di astrazione maggiore e consistenti con quelli di partenza. Tale materia di studio è anche l'aggiunta ai componenti prodotti di conoscenza ed esperienza che non può essere ricostruita direttamente ed automaticamente dai componenti analizzati, ergo esisteranno problemi indecidibili: ad esempio, non è possibile, a partire dal solo codice, astrarre *il progetto* dal quale esso è stato prodotto ma non è invece indecidibile il problema di astrarre un progetto coerente con il codice; non è possibile, a partire dal solo programma oggetto, astrarre *il programma* sorgente dal quale esso è stato prodotto ma non è invece indecidibile il problema di astrarre un programma sorgente che generi il dato programma oggetto.

Per far comprendere al meglio cos'è la reverse engineering, introduco le attività di restructuring e di reengineering, dove per restructuring intendo un'attività che trasforma il codice esistente in codice equivalente dal punto di vista funzionale, ma migliorato dal punto di vista della sua qualità (un esempio è quello che trasforma il codice non strutturato nel flusso di controllo in codice strutturato con eliminazione delle istruzioni "GOTO"); mentre per reengineering intendo un' attività di ristrutturazione a livello della riorganizzazione dell'architettura modulare di un sistema; si parte da una certa organizzazione dei moduli del sistema e del relativo flusso dati, e se ne producono altre con migliori caratteristiche di qualità, come ad esempio il controllo dell'uso di variabili globali. Quindi sia un processo di restructuring che di reengineering si sviluppa sempre nell'ambito di un definito livello di astrazione rispettivamente codice e moduli, al contrario, un processo di RE, si sviluppa sempre tra livelli di astrazione diversi, ad esempio, costruire dal codice i documenti di progetto e di specifica di un modulo.

Gli scopi di un processo di Reverse engineering sono:
• individuare i componenti del software e le relazioni esistenti tra essi;
• creare rappresentazioni del prodotto ad un più alto livello di astrazione;
• comprendere e descrivere le funzioni svolte dal prodotto stesso e le modalità con cui esse sono state implementate.
Nell'ambito di questa attività, da circa un decennio si è formata una corrente di ricerca con lo scopo di migliorare il riuso delle soluzioni di progettazione dei sistemi.
L'intento principale è l'individuazione nei sistemi software di soluzioni a problemi ricorrenti e si procede ad una loro catalogazione come *pattern*, cioè come modelli i quali dovrebbero diventare strumenti che condividono "buone idee" ed essere riutilizzati con successo in altri contesti applicativi.
I Design Pattern (DP) sono i pattern più noti nell'ambito dello sviluppo software.
Da molti studi si evince come questi DP sono presenti in svariati sistemi software esistenti e che la loro semantica permetta di avere informazioni non estraibili direttamente da un'analisi del codice sorgente, allorché riconosciuto un DP è possibile supporre il tipo di problema individuato dai progettisti del sistema e quali criteri sono alla base delle scelte progettuali adottate.
Un Design Pattern è una descrizione di un problema e del nucleo della sua soluzione in modo che questa possa essere riusata in diversi ambiti, ma implementata sempre in forma diversa, in un certo contesto: da questo punto di vista un pattern è quindi una regola tripartita (contesto, problema, soluzione); non è una specifica dettagliata, quanto più un insieme di conoscenze ed esperienze ("a well-tried solution to a common problem"), fatte da definizioni di classi e dei loro ruoli. Un esempio caratteristico dei Design Pattern sono le code, gli stack, le liste: è universalmente riconosciuto che le operazioni su di uno stack sono la push e la pop.
Gli elementi fondamentali di un Design Pattern sono:
- Un nome significativo
- Una descrizione del problema (motivazione e applicabilità)
- Una soluzione astratta ("a template for a design solution that can be instantiated in different ways")
- Una lista di conseguenze nell'applicare il pattern, i compromessi, i vantaggi e gli svantaggi

Il formato scelto per la creazione di un pattern non è importante, purché la specifica sia completa e precisa (in genere è divisa in paragrafi). E' bene specificare che un pattern non è solo teoria, ma spesso, essendo tutto frutto di esperienza, alcuni punti vengono meglio spiegati con del codice concreto.
Un Design Pattern dovrebbe essere usato solo da programmatori esperti che possano riconoscere in quali ambiti applicare un certo pattern; usare un pattern aiuta a:
- Costruire software riusabile
- Evitare scelte che (si è visto per esperienza) possano influire negativamente sul prodotto finale
- Migliorare la documentazione
- Rendere il programma modulare e flessibile

Gli obiettivi di chi utilizza Design Pattern sono:
- Aiutare i programmatori a risolvere problemi già analizzati
- Catalogare i pattern in una letteratura nella comunità OO

- Definire un linguaggio per trasferire la conoscenza e l'esperienza
- Far vedere che la soluzione trattata è
  - Utilizzabile (in un certo contesto)
  - Utile (ricorrenza della soluzione)
  - Usata (consolidata)

I Pattern sono organizzati secondo un Linguaggio dei Pattern (dizionario di Pattern) che contiene istruzioni per utilizzarli o per combinarli; un esempio di catalogazione è il seguente:

- Crezionali

Pattern che forniscono un'astrazione per il processo di creazione degli oggetti: un Sistema è indipendente da come sono creati, rappresentati e composti gli oggetti al suo interno

- Comportamentali

Pattern dedicati all'assegnazione di responsabilità tra oggetti e alla definizione di funzionalità, sottoforma di algoritmi

- Strutturali

Pattern dedicati alla composizione di classi e oggetti, in modo da formare strutture complesse; è possibile definire classi che ereditano da più classi in modo da usarne le diverse proprietà

I pattern che questa tesi tratta sono Object Element e Type relation che fanno fanno parte degli Elemental Design Pattern (EDP).

La tesi è suddivisa nei seguenti capitoli: il capitolo 1 descriverà la Reverse Engineering, i Design Patterns ed i framework mentre il capitolo 2 cercherà di spiegare gli Elemental Design Pattern che vengono ricercati nel Plugin, infine il capitolo 3 esaminerà l'architettura di Eclipse e del plugin.

Capitolo 1

Reverse Engineering e Design Pattern

Prima di descrivere dettagliatamente l'argomento di questa tesi si ritiene utile definire in quale contesto operano gli strumenti di reverse engineering, quindi quali benefici si possono riscontrare con uno strumento di ausilio al riconoscimento dei Design Pattern.

1.1 Reverse Engineering dei sistemi software

Generalmente, un processo di Reverse Engineering (come il plugin di questa tesi) inizia con l'analisi statica del codice, (processo di estrazione) preferibilmente già ristrutturato; con esso si estrarranno informazioni che poi saranno oggetto di successivi processi di astrazione, il cui fine è quello di produrre una documentazione consistente con il codice analizzato.
I processi di astrazione, al contrario del processo di estrazione, non sono completamente automatizzabili (salvo casi particolari, come ad esempio, quando opera in ambienti di produzione software altamente stabili nella organizzazione, nelle metodologie e nel personale impiegato) inoltre, per la produzione dei documenti di più alto livello di astrazione si richiede in generale l'aggiunta di conoscenza ed esperienza umana.
Il processo di estrazione dal codice si basa su un'analisi statica del codice stesso, il cui fine è quello di descrivere il prodotto analizzato mediante modelli rappresentanti i suoi componenti e le relazioni esistenti tra essi. In questa fase ogni modulo viene analizzato

indipendentemente dagli altri, ovvero non si considera gli effetti prodotti dalle interazioni con gli altri moduli, che invece verranno considerati nella fase di astrazione; quindi in fase di estrazione esisteranno relazioni tra moduli del tipo *direct relation*, mentre nel secondo caso *summary relation*.
Le informazioni tipicamente identificate che vengono estratte da un modulo analizzato sono:

- i nomi, il tipo ed i parametri formali dei moduli dichiarati in esso;
- i nomi ed il tipo dei dati dichiarati in esso;
- i nomi dei moduli che vengono attivati ed i nomi dei parametri attuali che vengono scambiati ad ogni attivazione;
- i nomi dei dati che vengono definiti nel modulo;
- i nomi dei dati che vengono usati nel modulo;
- le strutture di controllo utilizzate per l'implementazione, il loro nesting ed il flusso di controllo da esso determinato.

Le informazioni estratte vanno opportunamente memorizzate in un repository.

Il processo di astrazione, vuole ricostruire documenti a più alto livello di astrazione quali quelli del progetto architetturale, che mostrano l'organizzazione in moduli di un sistema software, le relazioni esistenti tra di essi e i relativi flussi di dati che si scambiano, e i documenti delle specifiche funzionali.
In maniera consapevole, avendo descritto un' introduzione del processo di reverse engineering, di astrazione e estrazione delle informazioni ora introduco la definizione di *reverse engineering* (RE) data in [CI90] :
l'attività di *reverse engineering* è il processo di analisi di un sistema allo scopo di:
- identificarne gli elementi base e le interrelazioni che li legano;
- crearne una rappresentazione in una diversa forma o con un livello di astrazione più alto.

Tale termine, ha origine in ambito di analisi hardware perché è una pratica molto comune giungere al progetto di un sistema partendo dal prodotto finito, mentre in ambito di sistemi software, a rigor di logica gli scopi di questa attività, naturalmente sono ben diversi perché l'obiettivo (obbiettivo software) della reverse engineering è il raggiungimento di un sufficiente livello di comprensione del progetto per:
- supportare la manutenzione del sistema;
- consolidare i miglioramenti apportati al sistema;
- supportare la sostituzione di alcune parti del sistema.

Per quanto riguarda lo sviluppo object oriented, la parola reverse engineering potrebbe sembrare inopportuna, in quanto il termine viene usualmente riferito ai sistemi "legacy", cioè scritti in linguaggi procedurali, ma in realtà è altrettanto appropriato.
Secondo la definizione data nello standard ISO 12207-1995:
Il prodotto software subisce delle modifiche al codice e alla documentazione associata a causa di problemi o per la necessità di miglioramenti. Lo scopo è di modificare il software esistente preservandone la sua integrità.
Il contesto cui è applicata l'attività di reverse engineering prende parte in diversi processi riguardanti un sistema software:
- Reingegnerizzazione
- Ristrutturazione
- Costruzione di un nuovo sistema
- Sviluppo evolutivo

Precisando che il sistema in oggetto può essere sia un singolo programma che un frammento di codice, oppure un insieme complesso di programmi che interagiscono tra di loro.
Inoltre la reverse engineering può essere effettuata su qualsiasi livello di astrazione (requisiti, progettazione, implementazione) e in un qualsiasi momento del ciclo di vita del software.
Inoltre si tratta di un processo di investigazione e non di modifica o di riproduzione.

1.2 Fase di Design, Design Pattern e sviluppo software

La fase di design di un sistema è quella fase cruciale che si interpone tra la fase di analisi e di implementazione. Sebbene la fase di analisi possa essere affrontata con successo applicando principi metodologici, la fase di design tende a sfuggire ad ogni tentativo di inquadramento canonico.
Durante il design vengono introdotti nuovi oggetti, la distribuzione di responsabilità, le relazioni e le regole di collaborazione tra gli oggetti di analisi possono essere modificate per rispettare nuovi vincoli talvolta contrastanti: incapsulamento, granularità, dipendenza, flessibilità, prestazioni, evoluzione, riuso, ecc. Definire un insieme di regole certe per il design è pressoché impossibile.
E se fare design è difficile, ancora di più è fare un "buon" design, allora appare palese porsi la domanda: cosa si intende per "buon" design?
Innanzi tutto la condizione necessaria è che risolva il problema, ma non basta perché che esistono infiniti programmi che risolvono lo stesso problema, quindi, qual è il migliore? Il giudizio può essere: "questo è un programma fatto bene" perché risolve il problema in modo elegante, con meno linee di codice, con meno uso di risorse, perché si presta a essere facilmente modificato e riusato in altri contesti. Si potrebbe quasi dire che il design è una forma d'arte, che richiede doti difficili da catturare e razionalizzare. Ma una cosa sembra certa: un designer con esperienza riesce a fare un buon design perché applica l'esperienza per riusare soluzioni sperimentate con successo nel passato per risolvere problemi simili.
Il concetto di pattern corrisponde al modo con cui funziona il cervello e travalica ogni dominio particolare e quindi nel dominio dello sviluppo software ha senso parlare di **design pattern**.
Tra i primi a applicare il concetto di pattern al design a oggetti individuando i pattern più importanti sono stati Gamma, Helm, Johnson, Vlissides (detti GOF = Gang Of Four) [GOF1] con il libro "Design Patterns, Elements of Reusable Object-Oriented Software", considerato la bibbia sui pattern. Il libro [GOF1] è soprattutto un catalogo di 23 patterns, ciascuno descritto sottolineando quattro elementi essenziali:
1. **Nome**. Sembra ovvio, ma dare nomi significativi ai pattern è molto importante. Un nome è un modo immediato e veloce per individuare un problema di design. Avere questi nomi nel proprio vocabolario permette di comunicare con altri designers passando complesse soluzioni di design con il semplice scambio di una parola. Dire "qui ho usato il

Singleton perché dovevo essere sicuro che la risorsa venisse allocata una sola volta" comunica immediatamente la soluzione al problema e il problema stesso.
2. **Problema**. Per definizione un pattern serve per risolvere un problema.
3. **Soluzione**. Descrive la struttura del pattern: gli elementi partecipanti e le forme di collaborazione e interazione tra questi. La descrizione è sempre sufficientemente astratta da essere applicabile in diversi casi e situazioni reali.
4. **Conseguenze**. Ogni pattern, e quindi ogni modo per risolvere un problema, ha vantaggi e svantaggi. Proprio perché esistono infinite soluzioni a un problema, a seconda delle criticità certe volte un pattern è più adatto di un altro. Identificare esplicitamente le conseguenze di un pattern aiuta a determinare l'efficacia dello stesso nei vari casi.
Pur rimanendo sufficientemente generali e indipendenti dal linguaggio, i pattern [GOF1] vengono descritti con esempi C++ e Smalltalk. Le pagine di questa nota sono una mia interpretazione dei pattern [GOF1] in ambito Java, sottolineando gli idiomi particolari di questo linguaggio.
A cosa servono i pattern:
1. Introducono un vocabolario comune. La conoscenza dei designer esperti non è organizzata secondo le regole sintattiche di un linguaggio di programmazione, ma in strutture concettuali più astratte. I pattern di design offrono un vocabolario comune per comunicare, documentare ed esplorare alternative di design. Permettono di descrivere un sistema in termini più astratti che non le semplici righe di codice del linguaggio di programmazione. I designer possono comunicare con frasi del tipo: "usiamo l'Observer in questo caso", o "fattorizziamo uno Strategy da queste classi". Avere un vocabolario comune evita di dover descrivere un'intera soluzione di design: basta nominare i pattern usati.
2. Permettono di capire più facilmente il funzionamento dei sistemi a oggetti. Molti sistemi a oggetti complessi usano design pattern. Conoscere i pattern aumenta la comprensione dei sistemi perché risultano più chiare le scelte di design.
3. Accelerano la fase di apprendimento. Diventare esperti designer richiede molto tempo. Il modo migliore è lavorare a lungo con sistemi a oggetti e osservare molti sistemi fatti da designer esperti. Avere conoscenza dei concetti dei pattern riduce il tempo necessario per diventare esperti.
4. Sono un complemento alle metodologie ad oggetti. Le metodologie a oggetti cercano di risolvere la complessità del design standardizzando il modo con cui si affronta il problema. Ogni metodologia introduce una notazione (es. Object Modeling Technique) per modellare i vari aspetti del design e un insieme di regole che indicano come e quando usare ogni elemento della notazione. L'approccio ortodosso e standardizzato delle metodologie non è mai riuscito a catturare l'esperienza dei designer. I pattern sono un complemento, il tassello mancante alle metodologie a oggetti. In particolare, il passaggio dalla fase di analisi a quella di design è sempre la più critica. Molti oggetti di design non hanno una corrispettiva entità nel dominio di analisi e i patterns sono un mezzo essenziale per spiegare questi oggetti.
5. Permettono di anticipare i cambiamenti. Lo sviluppo del software passa in genere lungo tre fasi:
prototipazione, *espansione* e *consolidamento*. Nella fase di prototipazione viene definita l'applicazione in modo che rispetti i requisiti iniziali. Quando l'applicazione viene messa in esercizio possono nascere nuovi requisiti che richiedono l'estensione delle funzionalità offerte: è la fase di espansione. Il problema è che spesso le funzionalità da introdurre richiedono grossi interventi sul software perché non flessibile per accettare i

cambiamenti. Ecco che si entra nella fasi di consolidamento in cui il design viene rivisto per accomodare la flessibilità necessaria. Il consolidamento comporta separazione di classi in sotto componenti, muovere operazioni lungo la gerarchia di ereditarietà, razionalizzare le interfacce delle classi, ecc. Il processo di consolidamento è inevitabile e in genere costoso a meno che il design iniziale venga fatto in ottica evolutiva e i pattern aiutano in questo senso.

Vale la pena chiarire brevemente le differenze tra i successivi concetti.
Una libreria è un insieme di classi base che forniscono funzionalità generiche, es. liste, dizionari, ecc.
Un framework è un insieme di classi cooperanti che formano design concreto e riusabile per uno specifico dominio (es. interfacce grafiche). Il framework definisce l'architettura dell'applicazione e cattura le decisioni di design comuni nel dominio in cui si applica. A differenza di una libreria dove l'applicazione specifica il flusso principale del programma e richiama le classi di supporto, con un framework il flusso principale è già definito dal framework per cui è sufficiente specializzare il comportamento di alcune sotto componenti.
I pattern differiscono in almeno tre aspetti dai framework:
- sono concetti astratti mentre i framework sono reali. I framework in genere usano diversi patterns nella loro architettura.
- hanno una struttura molto più semplice e focalizzata a un particolare problema di design. I frameworks hanno una struttura complessa tipicamente composta da diversi patterns.
- sono generici e non legati a un particolare dominio. I frameworks sono sempre legati a un dominio particolare.

Sia patterns che idiomi descrivono soluzioni a problemi ricorrenti di design, ma mentre nei patterns soluzione e problema sono sufficientemente generici da essere indipendenti dal linguaggio di programmazione, gli idiomi fanno riferimento alle proprietà specifiche di un certo linguaggio e descrivono come implementare le soluzioni di design nel linguaggio.

1.3 Relazione tra Reverse Engineering e Design Patterns

Per iniziare questo paragrafo ho ritenuto opportuno, oltre che quanto scritto in precedenza a riguardo della RE anche inserire la definizione di redocumentation e di design recovery.

Quindi per redocumentation s'intende:
- la documentazione prodotta può essere grafica o testuale
- non esistono metodi consolidati di ridocumentazione
- tipicamente il processo inizia sottoponendo il codice a uno strumento di analisi statica, che produce in uscita:
- relazioni di chiamata tra componenti
- gerarchie delle classi
- tavole di interfaccia dei dati
- pseudocodice

Quindi la redocumentation è la creazione o revisione di una rappresentazione semantica equivalente con lo stesso relativo livello di astrazione.
Design Recovery consiste nell'arricchire l'osservazione del sistema oggetto dell'analisi con conoscenze sul dominio applicativo, informazioni esterne, deduzioni o ragionamenti fuzzy per identificare livelli di astrazione più alti che siano significativi.
Citando Ted Biggerstaff: "deve riprodurre tutte le informazioni necessarie ad una persona per capire pienamente cosa fa un programma, come lo fa, perché lo fa, e così via".
Svariati autori hanno discusso i vantaggi derivanti dal documentare i sistemi software per capire il fondamento logico dietro le decisioni progettuali che è altrettanto importante quanto comprendere i costituenti strutturali e logici del software stesso. La maggior parte dei comuni strumenti impiegati nella reverse engineering trascura completamente la ricostruzione del design. La rappresentazione astratta del codice sorgente spesso non riesce a mantenere traccia dei motivi che sono alla base delle scelte che hanno guidato il progetto.
Allora appare evidente che le principali sotto-aree della reverse engineering che mi porteranno a motivare un coinvolgimento in tale attività del riconoscimento dei pattern sono la Redocumentation e la Design Recovery
I Design Patterns riescono a comunicare il razionale delle scelte progettuali grazie al tipo di processo che solitamente si deve seguire quando si utilizza un pattern.
Riassumendo, un possibile scenario, è stato creato prendendo spunto da [Fow03]:
1. *capire il problema*: il problema è una parte vitale di un certo contesto, questo implica che abbiamo un contesto che deve essere stato ben definito a priori;
2. *cercare il pattern*: si cerca nei contesti dei design pattern quelli che sono più simili al nostro;
3. *scelta del pattern*: si valutano i pattern trovati al passo precedente ed eventualmente si decide di utilizzarne uno o più.
4. *verifica di completezza*: si valuta se è il caso di utilizzare alcuni pattern correlati.

Quindi un insieme di pattern, estratto da un sistema esistente, ci fornisce un contesto a partire dal quale possiamo supporre il perchè di certi aspetti del progetto. In alcuni casi si potrebbe stabilire, una volta compreso il sistema e il suo contesto, se il problema era stato affrontato correttamente e indagare su un possibile uso errato dei Design Patterns.
Purtroppo l'introduzione dei Design Patterns nella reverse engineering incontra delle resistenze in entrambe le comunità che si dedicano ai due argomenti. Le argomentazioni principali si basano sul fatto che i patterns possono essere implementati in molti modi e che la stessa struttura può ricorrere con intenti assai diversi.
Penso che se d'accordo con quanto detto sinora sull'utilità dei Design Pattern e si comprende che molte parti di sistemi software esistenti possono essere meglio comprese se si recuperano i pattern in essi contenuti, in modo sia manuale che automatico. Naturalmente quest'ultimo è il mio caso.

Siamo concordi anche sul fatto che prima di pensare ad un metodo per il riconoscimento dei patterns è importante stabilire quali sono i valori che ne motivano la realizzazione.
Il reverse engineering 'e un processo personale, del resto la progettazione e lo sviluppo di un software sono attività umane dove entra in gioco quella piccolissima percentuale di creatività che ci rende unici, pertanto solo un altro umano, trovandosi nel posto giusto, al momento giusto potrà capire il perché di certe scelte. Di conseguenza il riconoscimento dei Design Patterns è utile se è applicato nel posto giusto, al momento giusto, altrimenti si riduce, ad un opinione comune, ad un'attività speculativa che non porta a nessun risultato utile.

Capitolo 2

Gli Elemental Design Pattern che vengono ricercati nel plugin

Prima di introdurre i gli elemental design pattern che vengono utilizzati da questo plugin (Object Elements e Type Relation) si è ritenuto opportuno scrivere questa introduzione riguardante l'esistenza di una sostanziale relazione tra i design pattern e gli elemental design pattern in quanto quest'ultimi possono essere visti come elementi di base su cui costruire i più complessi DP. Nell'intento di capire se veramente esistono relazioni tra i design pattern e gli elemental design pattern è stata condotta un indagine articolata da una prima analisi teorica dei design pattern la quale prevede lo studio su carta per voler intuire quali EDP potrebbero fare parte della loro struttura. Dopodiché si procede ad un'analisi pratica ,che verifichi tramite EDP Detector gli EDP trovati siano effettivamente presenti nell'implementazione dei design pattern.

2.1 Object Elements

I patterns del gruppo Object Elements sono quelli che si occupano della creazione e della definizione degli oggetti.
Create Object descrive: come, quando e perché istanziamo gli oggetti; cosa li rende speciali rispetto ai sistemi procedurali.
Abstract Interface da una soluzione su come rimandare l'implementazione delle operazioni sugli oggetti ad un momento più opportuno.
Infine, Retrieve mostra come e perché usare gli oggetti come attributi di un altro oggetto.

2.1.1 Create Object

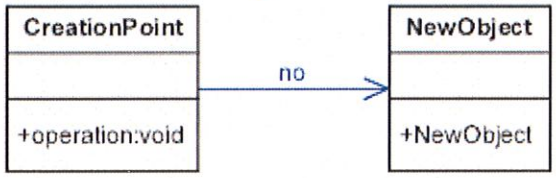
Intento

Assicurare che le strutture dati appena allocate siano conformi ad un insieme di asserzioni e precondizioni prima che vengano utilizzate dal resto del sistema e che si possa operare su di esse solo in modi predefiniti.

Conosciuto come

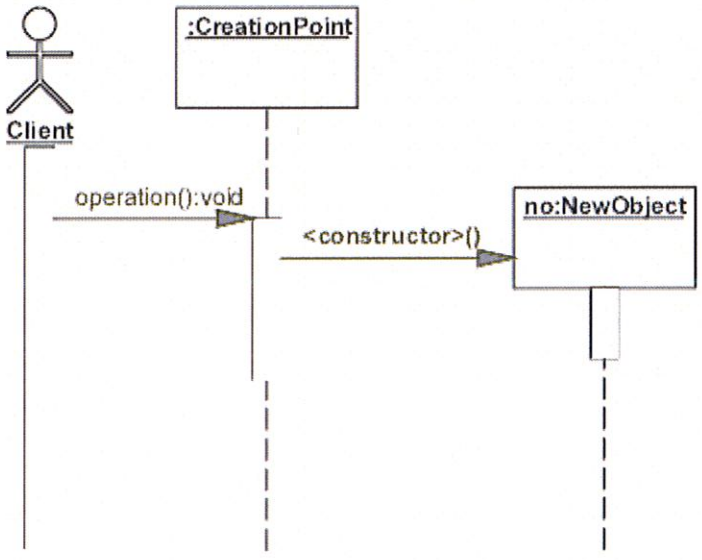
Instanzazione

Struttura



Collaborazione

Un'istanza di esempio richiede la creazione di un'istanza di NuovOggetto. I dettagli riguardanti la creazione e la modifica di un'istanza di NuovOggetto sono definiti internamente alla classe dell'oggetto stesso.



Implementazione

usare il nuovo oggetto: il costruttore invocato per la creazione dell'istanza di NuovOggetto ritorna un riferimento a tale istanza. Se il nuovo oggetto deve essere utilizzato localmente da Esempio il riferimento va assegnato ad un attributo che sia di tipo compatibile con il tipo di NuovOggetto.

Esempio di codice

```
public class Esempio {
private NuovOggetto no;
public void operation() { no = new NuovOggetto (); }
}
public class NuovOggetto {
public NuovOggetto () { // viene inizializzato lo stato della nuova istanza }
}
```

2.1.2 Abstract Interface

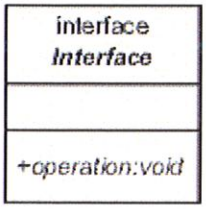
Intento

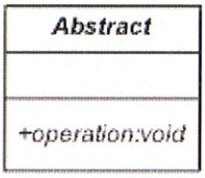
Avere un' interfaccia comune per operare su una famiglia di tipi di oggetti, posticipando la definizione delle reali operazioni da eseguire ad un momento successivo.

Conosciuto come

Virtual Method, Polymorphism, Defer Implementation

Struttura





Collaborazione

Un'interfaccia o una classe abstract definisce le interfacce di metodi che qualche sottoclasse implementerà

Implementazione

Interfacce. Se le classi che definiranno i metodi devono estendere altre classi (ereditarietà multipla delle interfacce) è necessario definire la classe base come interface.

Classi astratte. Se la classe base deve fornire l'implementazione di alcuni metodi o attributi che non siano costanti allora è necessario definire una classe abstract.

Esempio di codice

Una classe abstract deve indicare esplicitamente quali sono i metodi per cui ritardare la definizione. Se invece si utilizza un'interfaccia i metodi sono considerati implicitamente public e abstract, quindi il consiglio è di non esplicitare tali modificatori.

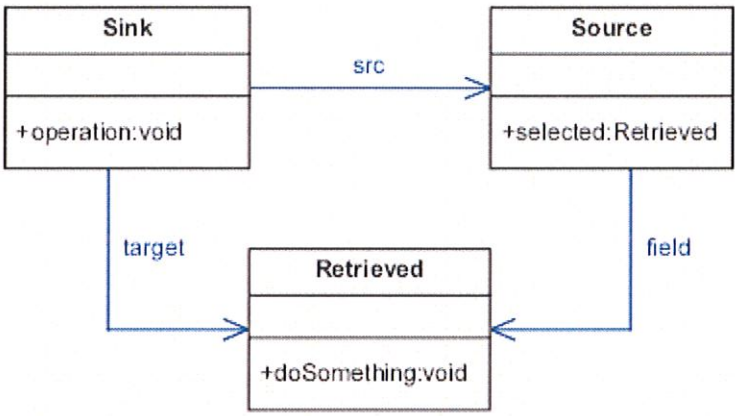
```
abstract public class AbstractOperations {
public abstract void operation();
}
public interface Interface {
void operation();
}
```

2.1.3 Retrieve

Intento

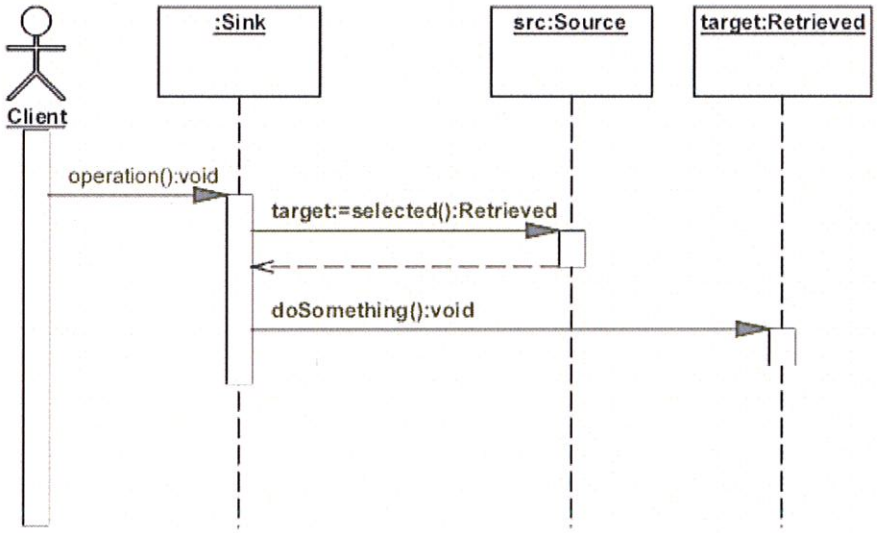
Usare, in ambito locale, un oggetto di una fonte non locale attraverso la creazione di una legame tra l'oggetto locale e quello remoto.

Struttura



Collaborazione

Il valore di ritorno di un metodo di Source è un riferimento ad un oggetto di tipo Retrieved il quale viene assegnato ad un attributo di Sink che sia di tipo compatibile con Retrieved.
Sink può ora operare sull'oggetto dall'ambito locale.



Implementation

1. *Valore di ritorno*. Il riferimento all'oggetto istanza di Retrieved viene fornito da Source mediante il valore di ritorno di un metodo.

2. *Attributo pubblico*. Source ha un attributo pubblico che punta all'istanza di Retrieved a cui altri oggetti possono accedere.

Esempio di codice

```
public class Sink {
private Source src;
private Retrieved target;
public void operation() {
// valore di ritorno
target = src.selected();
target.doSomething();
// attributo pubblico
src.field.doSomething();
}
}
public class Source {
private Retrieved field = new Retrieved();
public Retrieved selected() { return field; }
}
public class Retrieved {
public void doSomething(){}
}
```

2.2 Type Relation

Il gruppo Type Relation è formato da un solo pattern: Inheritance. Questo EDP presenta il principale metodo utilizzato nei sistemi orientati agli oggetti per permettere il riutilizzo delle informazioni di tipo e delle implementazioni dei metodi. L'ereditarietà è considerata la caratteristica fondamentale dei linguaggi ad oggetti per consentire la costruzione del software secondo un processo incrementale.

2.2.1 Inheritance

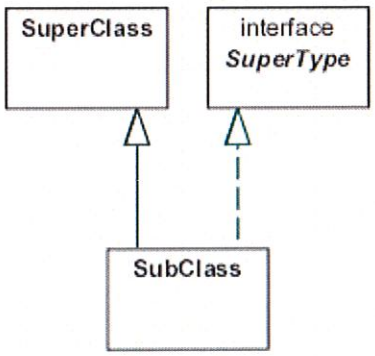
Intento

Usare l'interfaccia completa, attributi e metodi, di un'altra classe e usare interamente o solo in parte la sua implementazione.
Permettere ad un singolo oggetto di essere istanza di più di una classe.

Conosciuto come

Ereditarietà

Struttura



Collaborazione

- SuperClass fornisce un insieme di interfacce a metodo accompagnate, eventualmente, dalla relativa implementazione. Quelle mancanti devono necessariamente essere fornite da SubClass, altrimenti quest'ultima deve essere definita come classe abstract.
- SuperType può esclusivamente definire delle interfacce a metodo di cui SubClass deve obbligatoriamente fornire l'implementazione.
- SubClass eredita tutti i membri visibili di SuperClass (metodi e attributi) e tutte le eventuali implementazioni che SubClass potrà poi scegliere di ridefinire (*overriding*).

Implementation

1. *Tipi definiti dall'utente*. Le interfacce nella gerarchia delle classi aggiungono a Java l'ereditarietà multipla. Le classi estese e le interfacce implementate da una classe C sono dette i *supertipi*, o *tipi base*, di C, mentre C è un *sottotipo*, o *tipo derivato*. Le definizioni di classe e di interfaccia consentono di creare nomi di tipi da usare nella dichiarazione di variabili; ogni oggetto sottotipo di del tipo così definito può essere assegnato a quella variabile (*polimorfismo*).

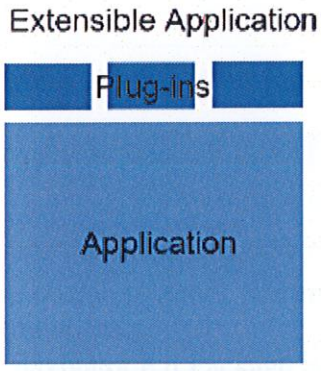
Esempio di codice

```
public class SubClass extends SuperClass implements SuperType {
// Implementare tutti i metodi dichiarati in SuperType
// Se necessario, ridefinire alcuni metodi di SuperClass
}
```

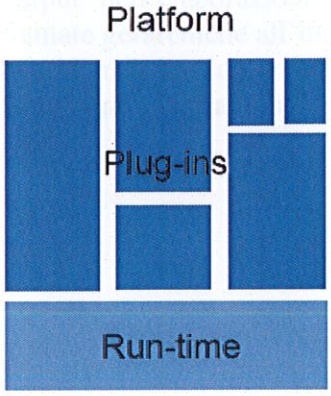
Capitolo 3

3.1 Architettura di Eclipse

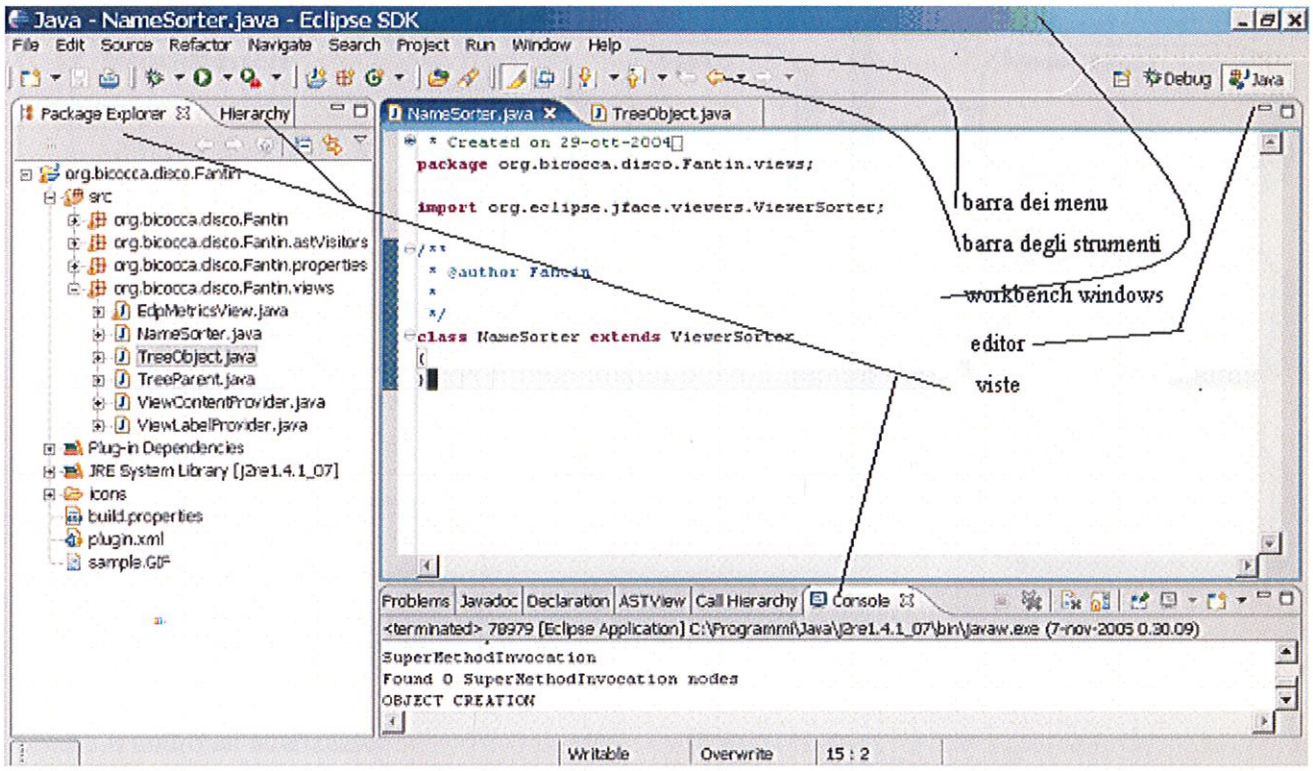
Innanzi tutto vorrei specificare come è costituito Eclipse, cioè se un applicazione estensibile e quindi è strutturabile nel seguente modo:



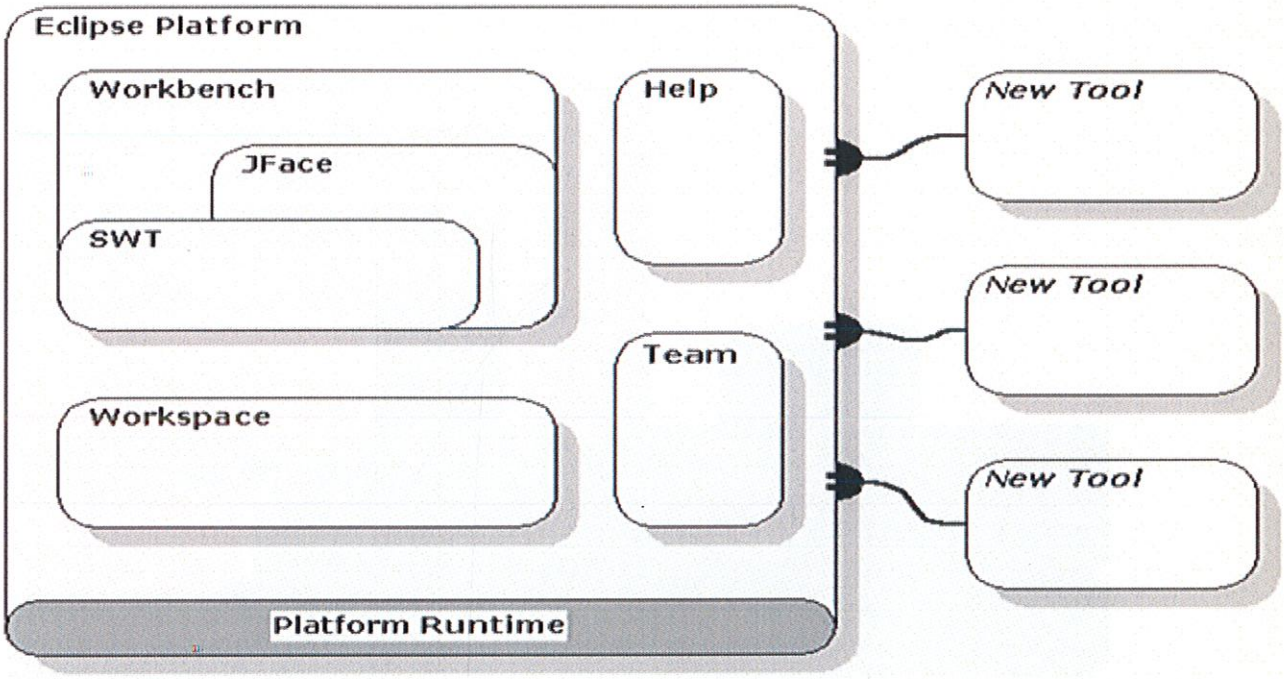
oppure se è una piattaforma.



Eclipse è una piattaforma! Essa non viene utilizzata per un particolare uso, bensì diverse applicazioni possono essere fatte "per girare" su di essa.
La figura seguente rappresenta come è costituito Eclipse, in questo caso la directory di lavoro è il mio plugin.

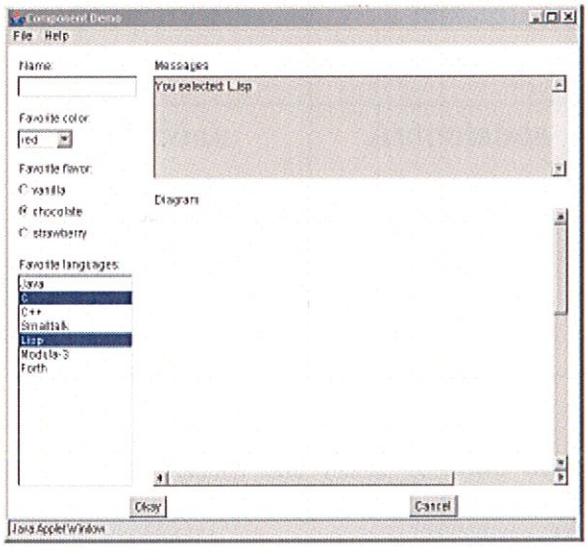


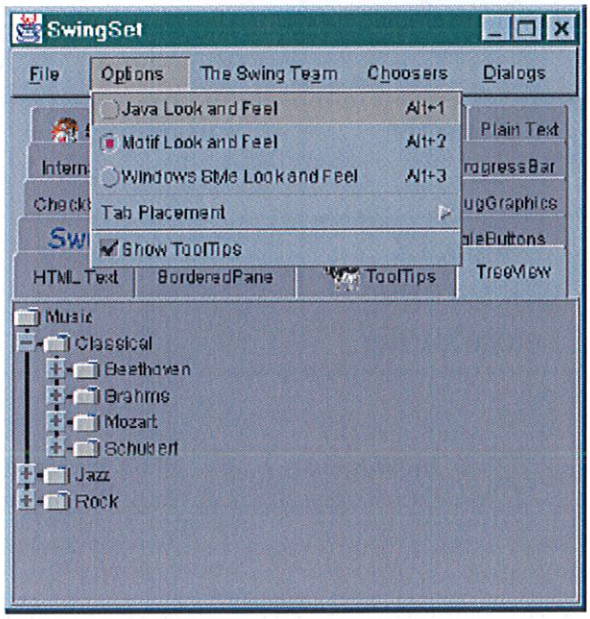
Da quanto si nota è presente un "navigator", un editor di testo, hierarchy, una console ed altre utilità.
Il navigator è il Package Explorer che tramite la sua visualizzazione ad albero permette di esplorare il package interessato.
L'editor di testo banalmente fa in modo di editare il sorgente e la finestra console stampa l'output dell'elaborazione. Hierarchy è una utility che permette di visualizzare le chiamate gerarchiche all'interno del file selezionato nel editor di testo.
Il workbench è unico, mentre gli editor e le viste sono comprese tra 0 e n.
La seguente figura rappresenta quali siano le maggiori componenti della piattaforma Eclipse



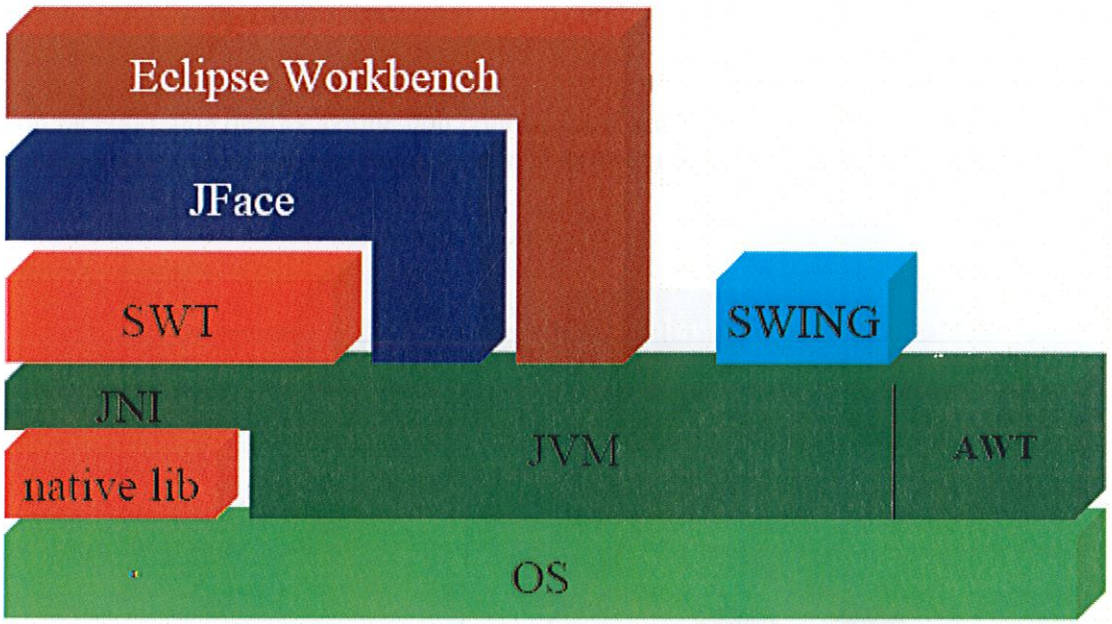
Adesso li andrò ad analizzare:

inizio con le SWT (Standard Widget Toolkit), perché il plugin sviluppato si basa soprattutto su esse, quindi per capire correttamente il loro funzionamento, bisogna confrontarle con le awt e le swing.





A sinistra è presente un esempio di awt, mentre a destra quello di una swing, allora:



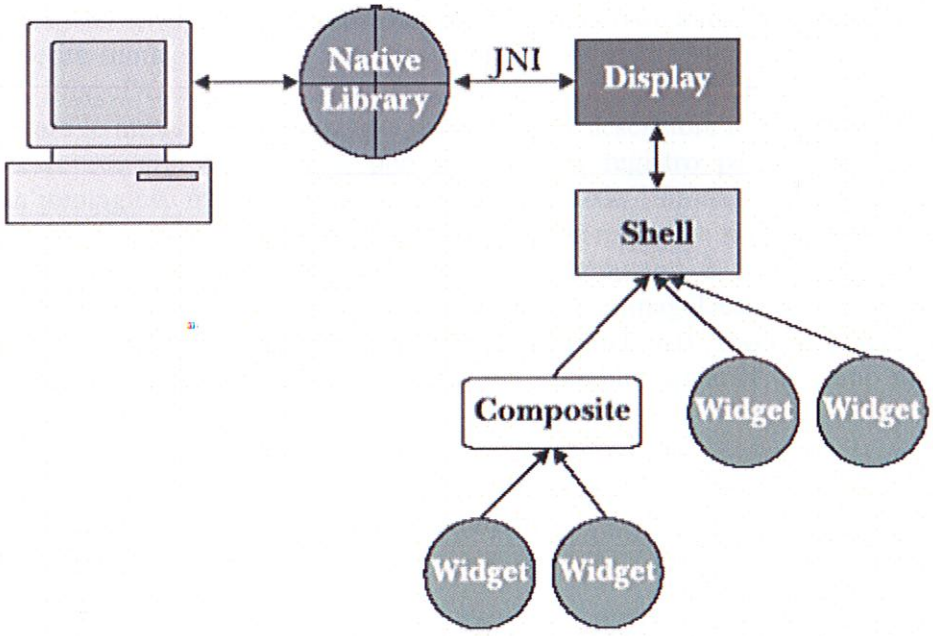
Quindi confrontando awt/swing/swt, si ottiene :

	AWT	SWING	SWT
Performance	+ +	-	+(++)
Versatilità	-	+	++
Gestione Memoria	Automatica	Automatica	Manuale
Look & Feel	Povero	Ricco	Ricco e conforme alla piattaforma

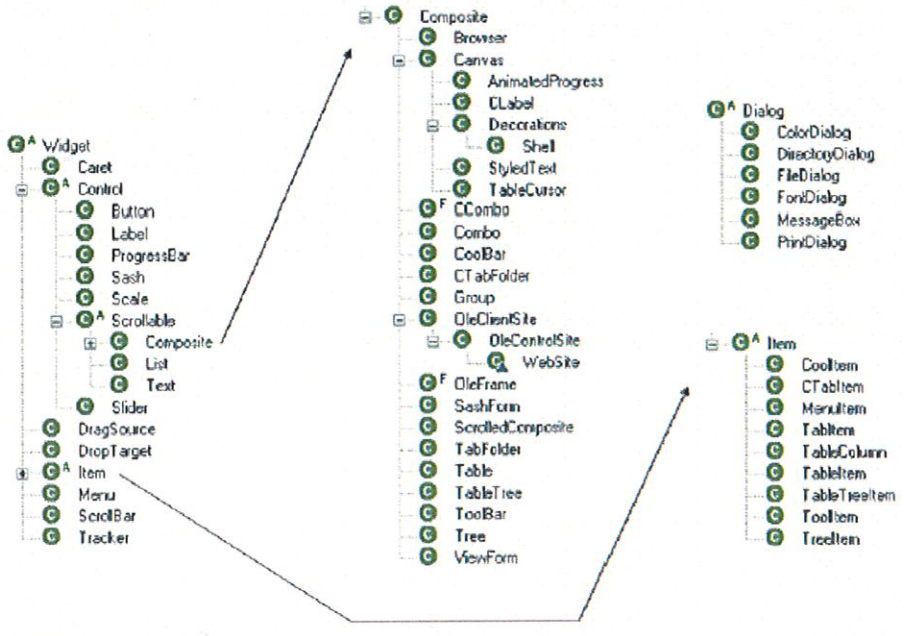
Vorrei sottolineare che nella gestione della memoria delle swt i componenti SWT allocano risorse al di fuori della JVM che vanno deallocate manualmente mediante il metodo dispose().
Un esempio può essere:
Color blue = new Color (display, 0, 0, 255);
blue.dispose()
oppure:

```
public void inputChanged(Viewer v, Object oldInput, Object newInput) {
                }
                public void dispose() {
}
```

Non facendo, una deallocazione non controllata si rischierebbero dei deadlock.
Concludo per quanto concerne le swt, con la struttura di comunicazione delle swt



e la gerarchia delle classi

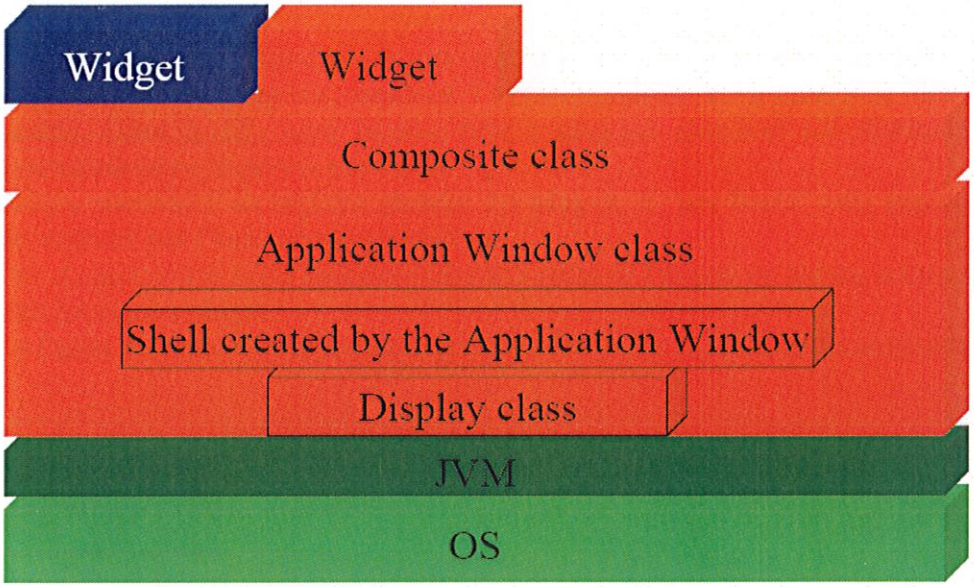


Un'altra componente importante è la libreria JFace che introduce il concetto di registro per le immagini (attraverso l'utilizzo di org.eclipse.jface.resource.ImageRegistry) e per effettuare le operazioni , è necessario ricorrere all'oggetto registro di immagini. L'oggetto ImageRegistry non ha particolari restrizioni, se non quella di richiedere un oggetto di tipo org.eclipse.swt.widgets.Display attivo. La classe ImageRegistry è essenzialmente una collezione nella quale inserire o dalla quale prelevare immagini. Inoltre essa automaticamente distrugge le immagini quando viene distrutto l'oggetto display naturalmente avviene quando l'ultima shell viene chiusa. Le immagini possono anche essere rimosse esplicitamente dal registro se lo si desidera, rilasciando le risorse occupate. Bisogna fare attenzione ad una caratteristica molto importante: non è possibile inserire immagini nel registro, ma solo descrittori di esse. Adesso cerco di chiarire cos'è un ImageDescriptor, e come crearlo. Un ImageDescriptor è un oggetto che sa come creare le immagini. Il vantaggio di inserire un descrittore nel registro anziché l'immagine stessa sta nel fatto che si può preparare il registro per il possibile inserimento di un'immagine, ma se l'applicazione non richiede l'immagine del registro, il descrittore non sarà mai utilizzato e non saranno sprecate risorse di sistema per caricare l'immagine in memoria. Per creare un descrittore di immagini è sufficiente utilizzare i metodi presenti nella classe org.eclipse.jface.resource.ImageDescriptor: createFromFile(Class location, String filename)) , createFromURL(URL url) , e getMissingImageDescriptor() . Il metodo createFromFile è l'unico che potrebbe apparire il meno chiaro. Lo si può interpretare in questo modo:

```
URL url = location.getResource(filename);
createFromURL(url);
```

In altre parole, va a cercare un'immagine relativa al file della classe inserita. Se "class location" è com.mycompany.MyClass, e "filename" è "/images/myImage.gif", si aspetterà di trovare l'immagine in questo percorso:
[APP ROOT]/com/mycompany/images/myImage.gif .
Riepiloghiamo adesso i passi che consentono di creare un ImageRegistry per la nostra applicazione:
- Creare un oggetto ImageRegistry (dopo aver creato un oggetto Display di SWT).
- Inserire tutte le immagini necessarie all'applicazione nell'ImageRegistry.
- Inserire il registro di immagini in una locazione comune in modo da poterlo utilizzare in qualsiasi porzione di codice che richieda le immagini.

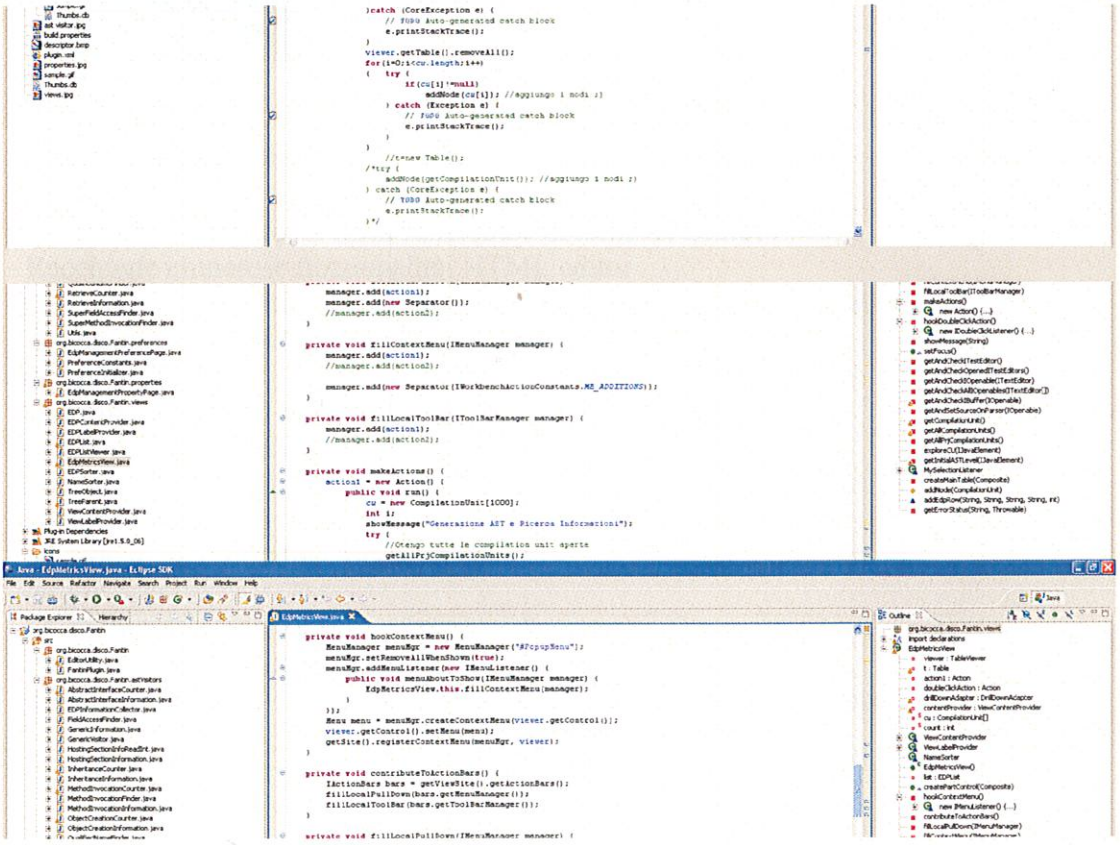
Quando necessario, richiamare le immagini dal registro per utilizzarle nei nostri componenti.
La libreria JFace ha una comunicazione strutturata di classi nel modo seguente:



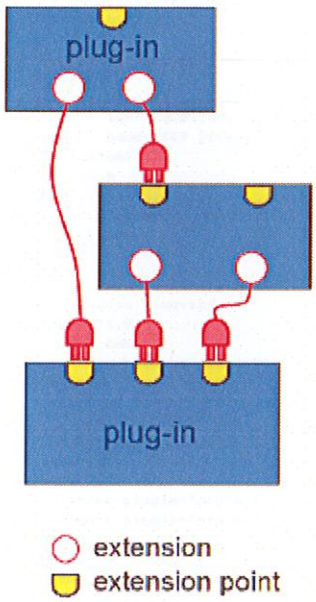
Per quanto riguarda la workbench di Eclipse:

- gestisce le finestre e le prospettive
- crea i menu e le toolbar
- crea le viste e gli editor

La seguente figura evince quello appena citato



Eclipse Platform Runtime è un micro-kernel, infatti tutte le funzionalità sono implementate in plug-ins, gestisce l'inizializzazione e la creazione dell'ambiente di lavoro. Oltre a ciò, esso ricerca tutti i plug-ins installati sul disco,unisce tutte le extensions con i rispettivi extension
points, costruisce un registro globale dei plug-in, e "per finire" crea una copia in cache del registro per l'esecuzione successiva.
Ora da descrivere delle maggiori caratteristiche di Eclipse ne manca solamente una, cioè l'architettura a plugin.
La figura sottostante è un esempio grafico di come in Eclipse venga architettato il plugin



Il plug-in è la più piccola unità funzionale in Eclipse, quindi :
- Racchiude numerose funzionalità: HTML editor
- Extension Point: entità nominale entro la quale collezionare "contributi"
- Extension: un contributo

Ogni plug-in:
- Contribuisce ad 1 o più extension point
- Opzionalmente dichiara 1 o più extesion point
- Dipende da un set di altri plug-in
- Contiene librerie di codice Java o altri file
- Risiede in una sottodirectory a lui dedicata

Plug-in Manifest:
- Dichiara tutti i "contributi"
- Implementa interfacce o fornisce API
- Plugin.xml: descrive le proprietà del plugin (vedi figura sottostante)


```
            >
      </category>
      <view
            name="Edp Detector"
            icon="sample.GIF"
            category="org.bicocca.disco.Fantin"
            class="org.bicocca.disco.Fantin.views.EdpMetricsView"
            id="org.bicocca.disco.Fantin">
      </view>
   </extension>
   <extension
         point="org.eclipse.ui.perspectiveExtensions">
      <perspectiveExtension
            targetID="org.eclipse.ui.resourcePerspective">
         <view
               ratio="0.5"
               relative="org.eclipse.ui.views.TaskList"
               relationship="right"
               id="org.bicocca.disco.Fantin.views.SampleView">
         </view>
      </perspectiveExtension>
   </extension>
   <extension
         point="org.eclipse.ui.preferencePages">
      <page
            class="org.bicocca.disco.Fantin.preferences.EdpManagementPreferencePage"
            id="org.bicocca.disco.Fantin.preferences.EdpManagementPreferencePage"
            name="EDP Detector Preferences"/>
   </extension>
   <extension
         point="org.eclipse.core.runtime.preferences">
      <initializer class="org.bicocca.disco.Fantin.preferences.PreferenceInitializer"/>
   </extension>
</plugin>
<plugin
   id="org.bicocca.disco.Fantin"
   name="Fantin Plug-in"
   version="1.0.0"
   provider-name=""
   class="org.bicocca.disco.Fantin.FantinPlugin">
   <runtime>
      <library name="Fantin.jar">
         <export name="*"/>
      </library>
   </runtime>
   <requires>
      <import plugin="org.eclipse.ui"/>
      <import plugin="org.eclipse.core.runtime"/>
      <import plugin="org.eclipse.core.resources"/>
      <import plugin="org.eclipse.jdt.core"/>
      <import plugin="org.eclipse.ui.workbench.texteditor"/>
      <import plugin="org.eclipse.jface.text"/>
   </requires>
   <extension
         point="org.eclipse.ui.propertyPages">
      <page
            class="org.bicocca.disco.Fantin.properties.EdpManagementPropertyPage"
            id="org.bicocca.disco.Fantin.properties.EdpManagementPropertyPage"
            name="Edp Management"
            nameFilter="*"
            objectClass="org.eclipse.jdt.core.IJavaProject">
      </page>
   </extension>
   <extension
         point="org.eclipse.ui.views">
      <category
            id="org.bicocca.disco.Fantin"
            name="EDP Detection"
```

Per quanto concerne l'attivazione dei Plugin ogni singolo Plugin
- ha il proprio Java class loader
- delega l'esecuzione al plug-in stesso
- restringe la visibilità alle API esportate
- i plug-in sono attivati solo al momento in cui sono necessari, cioè un esempio può essere quando il plug-in è attivato solamente quando l'utente seleziona una voce corrispondente in un menù
- utilizza una soluzione scalabile se si hanno molti plug-in installati (non è il mio caso)

Bisogna sottolineare il fatto che Eclipse incorpora Apache Ant, cioè è un tool che permette la configurazione, la compilazione ed il deployment di progetti Java e la sostituzione di makefiles con dei files XML. Eclipse permette anche di eseguire build files di Ant sia internamente che esternamente allo spazio di lavoro e PDE (Plug-in Development Environment) usa Ant per fare il deploy dei plug-in. Ho appena introdotto Ant, ma per capire cosa sia esattamente riporto le sue principali caratteristiche:
- Compilazione di sorgenti differenziata in base alle esigenze dell'utente
- Compilazione delle sole classi effettivamente modificate
- Creazione e cancellazione di directory di supporto
- Copia e cancellazione di file
- Creazione di file jar o war
- Installazione di componenti
- Generazione di javadoc

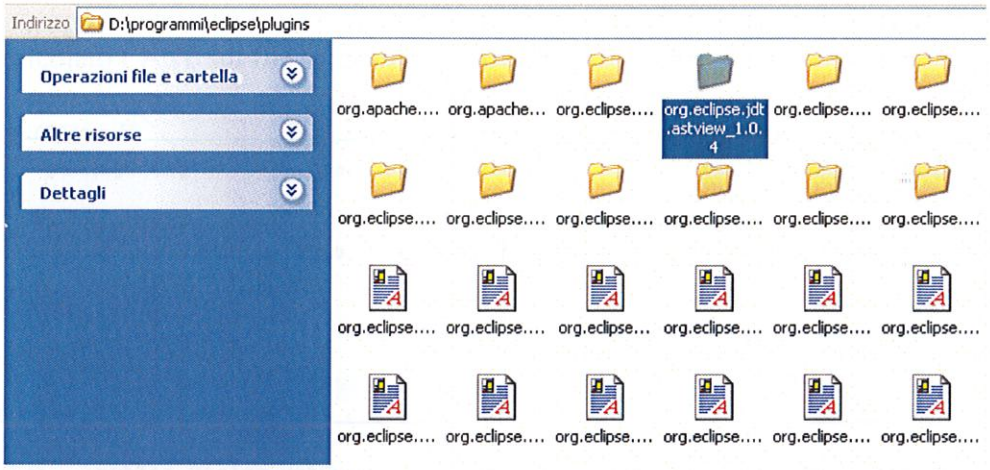
3.2 Architettura del plugin EDP Detector

In questo paragrafo verranno indicate tutte le funzionalità che sono presenti all'interno del plugin EDP DETECTION.

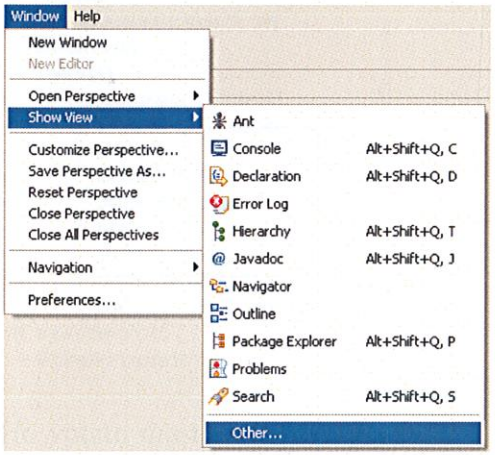
• Verificare la funzionalità del codice sulla versione 3.1.di Eclipse e se funziona questa sarà la versione di Eclipse di riferimento, in particolare, va verificato il funzionamento di:
1. ASTView
2. codice di esempio

1)

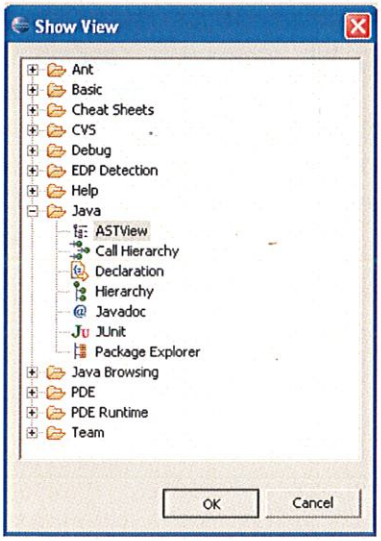
Inizialmente ho dovuto installare la piattaforma Eclipse e per visualizzare l'AST (Abstract Syntax Tree) ho utilizzato il plug-in org.eclipse.jdt.astview_1.0.4 la cui versione più recente è aggiornata al 30/03/2005.
Per verificare che tale plug-in è compatibile con la versione di Eclipse perché lo ho scompattato all'interno della directory plugins di Eclipse (come si vede in figura)



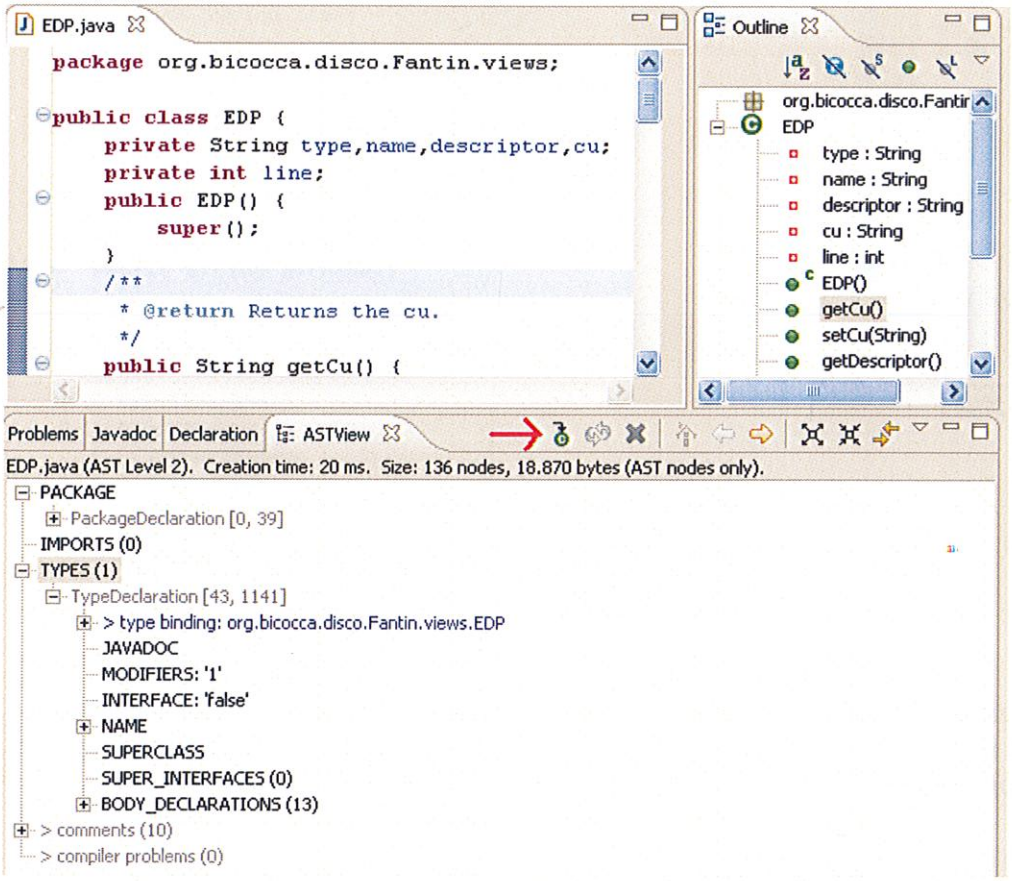
ho aperto Eclipse,lanciato una nuova istanza di Eclipse (con tutti i plugin presenti nell'ambiente di sviluppo) ,selezionato Window ,Show View, Other



quindi apro la cartella Java e seleziono AST View dando successivamente l'ok

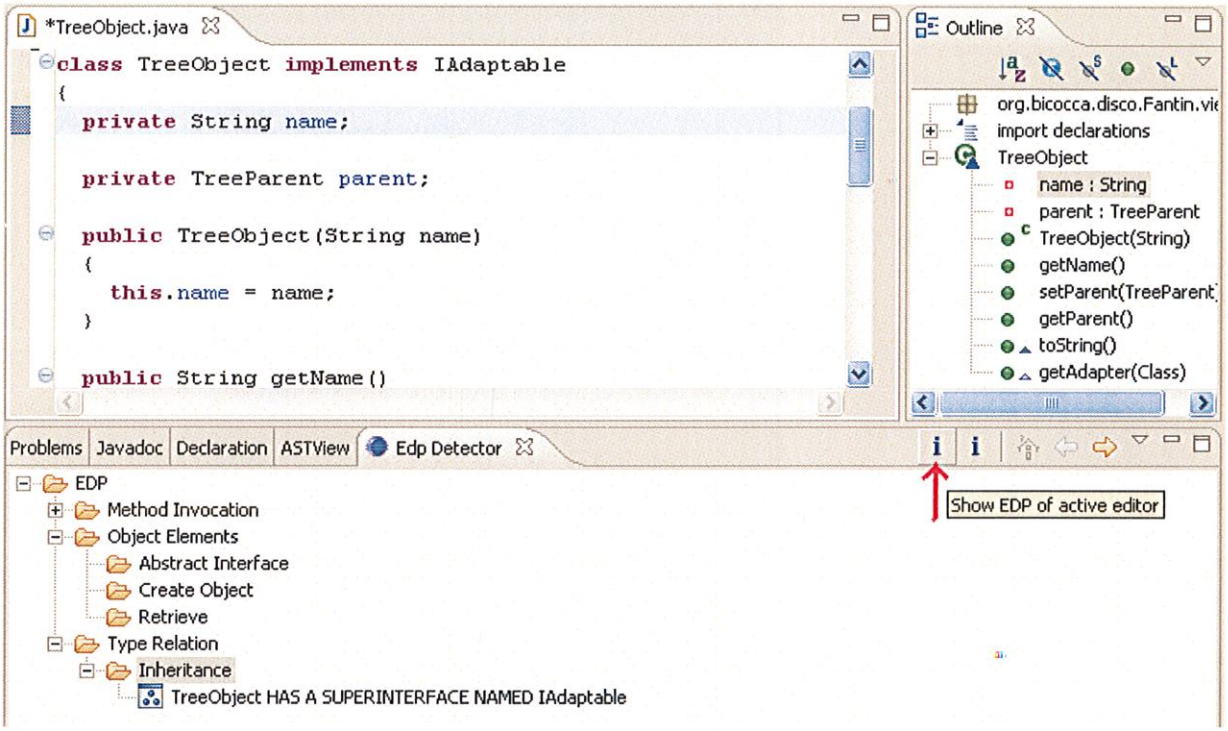


ottengo la tab ASTView e cliccando sul pulsante (evidenziato dalla freccia rossa) verrà calcolato l'AST sul file selezionato (EDP.java)



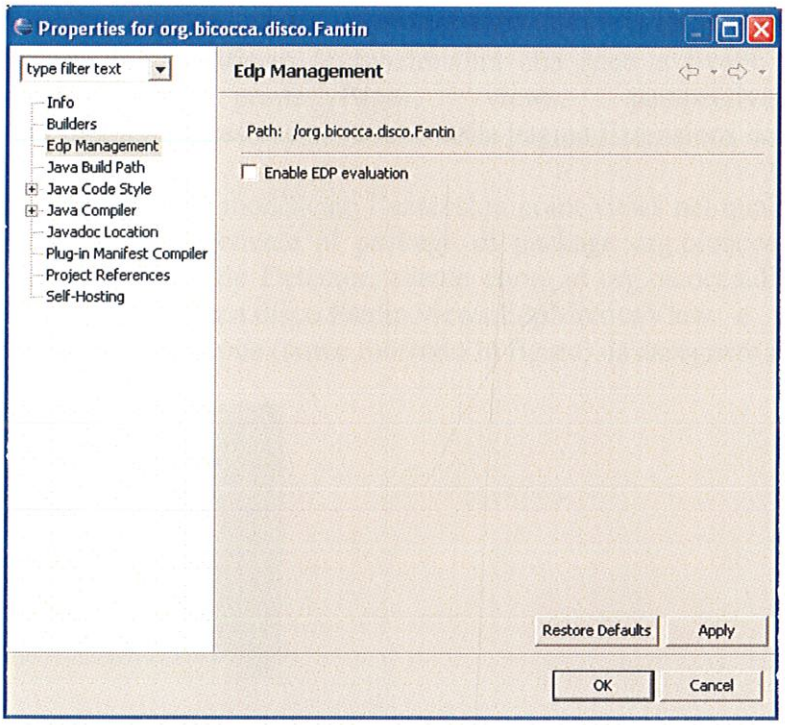
Ho voluto mostrare l'AST perché delle informazioni del mio plugin le vado a prendere proprio da esso.

2)
Il codice d'esempio del plugin è un progetto in fase di sviluppo ed è chiamato org.bicocca.disco.edpstarter , l'ho importato come progetto esistente di Eclipse, ho cercato di analizzare il sorgente e quindi l'ho fatto partire perché da questo sorgente assegnatomi dal mio correlatore che dovrò "fare apparire" il mio plugin funzionante.
Ora mostro, nella figura riportata sotto come è il funzionamento nella fase di esecuzione del sorgente assegnatomi:



se ne deduce che nella tab Edp Detector si hanno due pulsanti e facendo partire il primo, cioè quello indicato dalla freccia rossa, che svolge l'azione di cercare tutti gli EDP negli editor attivi si ottengono i valori in una visualizzazione ad albero. Mentre se faccio partire il secondo pulsante ottengo quello indicato dalla seguente figura.

quindi si deduce che questa misteriosa azione due sia stata eseguita…
Adesso, vado a controllare se è presente la property page e la preference page che sono una parte fondamentale ed integrante del plugin org.bicocca.disco.Fantin che verranno discusse successivamente. Il risultato è il seguente:
per quanto riguarda la property page trattata nella figura sottostante



si deduce che viene stampato a video il path ed è presente una check box per abilitare la valutazione degli EDP che a mio modesto parere è vero che li abilita, ma non li salva da nessuna parte…
Per quanto riguarda la preferences page non esiste assolutamente… lo stesso vale per JLS2 e JLS3…
Questo non significa che il progetto edpstarter dell'Università degli Studi di Milano Bicocca non funziona e che io devo farlo "girare" perché scriverei una cavolata … quello che voglio fare capire è che io sono partito da questo codice d'esempio che svolge in maniera sostanziale tutto quello che ho citato in precedenza e ho creato un plugin tutto mio (composto dai successivi punti). In particolare, Edp Starter funziona, ma il sorgente che mi è pervenuto riguardante appunto tale plugin è stato giustamente non completo,infatti sono stati cancellati dei pezzi di codice, infatti per questo lo si è inteso come codice d'esempio.

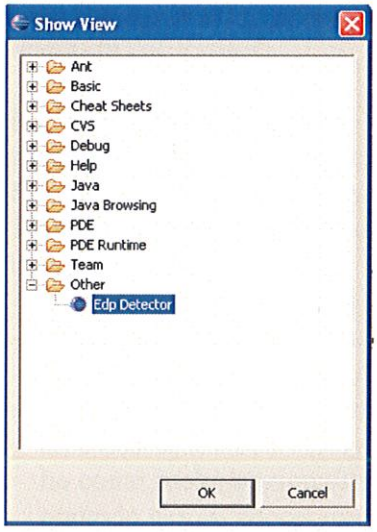
- Per visualizzare la vista Edp Detector, ora esiste un menu Other (che tra l'altro viene richiamato perché non c'è altro disponibile, con la segnalazione di un errore) aggiungere un Folder "EDP Detection" e far in modo che compaia lì l'EDP Detector

Inizialmente ho costruito un nuovo plug-in, con le stesse esatte funzionalità di quello esistente e l'ho chiamato org.bicocca.disco.edpFantin, dopodiché prima di avventurarmi nelle modifiche del sorgente, ho letto una prima volta in maniera superficiale, il materiale didattico che mi ha dato sia il mio relatore che il mio correlatore perchè ho iniziato questo argomento dello stage partendo da zero (poi successivamente me li sono letti una seconda volta in maniera più approfondita), i testi sono: Eclipse A Java Developer's Guide, Eclipse Cookbook (Jun.2004) , Contributing to Eclipse Principles, Patterns, and Plug-Ins ed infine eclipse-overview (un riassunto su Eclipse scritto dal mio correlatore) .

Avendo quindi acquisito le nozioni teoriche sia della reverse engineering che quelle su Eclipse ho modificato le funzionalità che sono a livello di punti di estensione più precisamente propertyPages, views, perspectiveExtensions,PreferencesPage, runtimePreferences che si vedono nella pagina Extensions del file plugin.xml (già trattato in precedenza).
In particolare ho modificato l'extension point views nel quale ho aggiunto una vista Edp Detection appartenente al package di package org.bicocca.disco.Fantin dopodiché ho creato la vista Edp Detector, avente come id org.bicocca.disco.Fantin,appartenente alla classe org.bicocca.disco.Fantin.views.EdpMetricsView e infine mi sono disegnato totalmente una icona (come mostrato in figura) da assegnare alla medesima vista.

Ho voluto apposta ingrandire tale icona per fare notare che rappresenta la scritta EDP (in giallo).
Tutto quello appena descritto (in maniera più concreta) è:

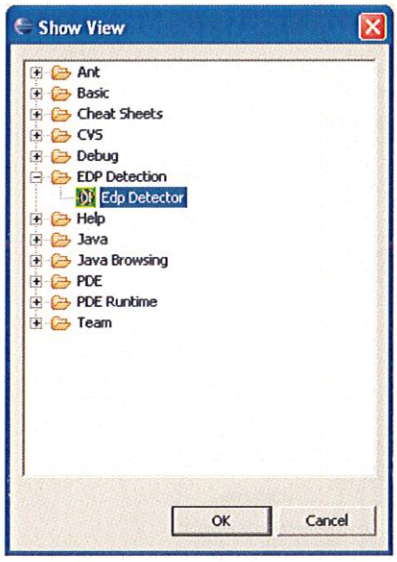
prima



Errore :
Category org.bicocca.disco.edpStarter not found for view org.bicocca.disco.edpStarter.views.EdpMetricsView. This view added to 'Other' category.

Nota: l'errore viene generato perché non veniva trovata la vista

Dopo:



Nota: non è presente nessun errore

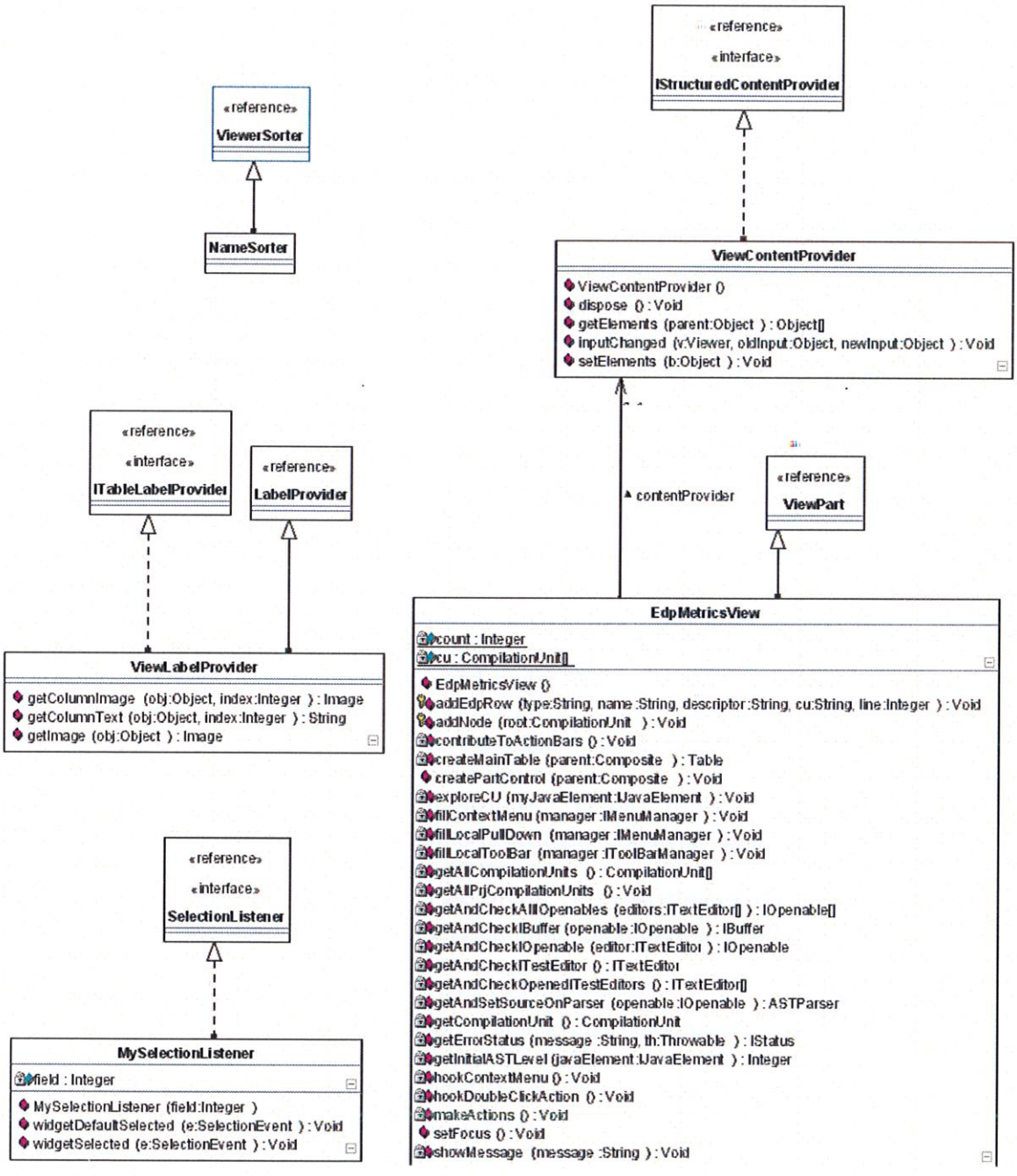
- Nella vista EDP Detector passare alla visualizzazione tabellare (al posto del tree), le cui colonne dovranno essere inizialmente EDP Type (varrà solo Object Element o Type Relation), EDP Name (valori possibili Abstract Interface, Create Object, Retrieve e Inheritance) e un generico Description (all'interno del quale andrà riportata la Stringa restituita dal Visitor).

Per avere una visualizzare tabellare ho cancellato buona parte del codice, per capirci meglio, quello appartenente alla visualizzazione ad albero presente nel file EdpMetricsView (appartenente al package Views) è ho creato una classe di tipo TableViewer. Ora cerco di descrivere le caratteristiche che mi hanno fatto propendere per la tale scelta:

- viene utilizzata jface (org.eclipse.jface.viewers.TableViewer)
- è una vista concreta basata su SWT Table control
- la classe non è intesa per essere sottoclasse fuori dal viewer framework
- è stata disegnata per essere instanziata con una pre-esistente tabella di controllo SWT ed è configurata con uno specifico dominio contenent provider, table label provider ed ha come scelte opzionali element filter e element sorter che quest'ultimo mi può essere comodo per un successivo punto (il riordinamento della tabella)
- TableViewer supporta SWT.Virtual flag, è importante notare che SWT.VIRTUAL usa il Widget basato sulle APIs che ritornerà null in entrambi i casi dove gli elementi non sono stati specificati o non stati creati

Prima di descrivere le modifiche di codice effettuate darei un idea di ciò che è stato fatto tramite un diagramma UML della classe EdpMetricsView generato in maniera automatica tramite il software gratuito sviluppato dall'università di Paterborn

(germania) chiamato Fujaba. Ho premesso che è svolto in maniera automatica, ma prima di riportarlo nella figura sottostante l'ho controllato perché è possibile che il programma generi un diagramma delle classi non corretto.



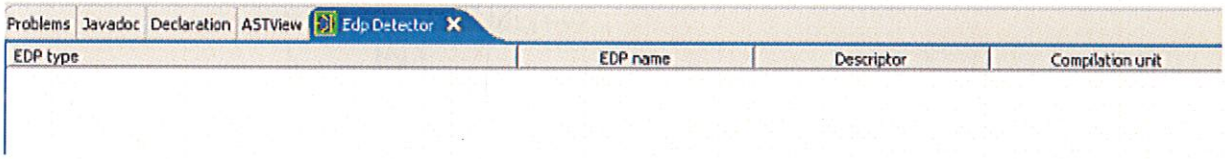
Mi scuso anticipatamente perché nel successivo discorso ho pensato che l'unico modo di fare realmente capire il lavoro svolto fosse riportare del codice sorgente e del commento su esso all'interno della stessa frase, ciò significa che quello scritto successivamente non è molto leggibile se non si conosce bene e a fondo il linguaggio di programmazione Java. Premesso ciò, a livello di sorgente sono andato a "toccare":

nella classe ViewContentProvider (che implementa IStructuredContentProvider) il metodo inputChanged a cui gli ho passato il Viewer e nel metodo createPartControl che ha come parametro Composite parent mi sono creato una nuova TableViewer parametrizzata da this.createMainTable(parent), mi sono creato new EDPList() da cui viewer.setContentProvider(new EDPContentProvider(list,viewer)) in quanto setContentProvider setta il contenet provider usato da questa vista e viewer.setLabelProvider(new EDPLabelProvider())quindi gli do un identificativo infine viewer.setInput(list) cioè l'implementazione del ContenentViewer di questo Viewer invoca inputChanged sul contenuto del provider, vorrei banalmente sottolineare che questo metodo fallisce se questa vista non ha un content provider.
Dopodiché, (sempre nella stessa classe) come appare evidente dal codice riportato sotto ho Menu menu = menuMgr.createContextMenu(viewer.getControl()) in particolare creo e ritorno un SWT context menu control per questo menu, mentre viewer.getControl().setMenu(menu) significa che mi ritorna il controllo SWT il quale vede il contenuto della vista, quindi getSite().registerContextMenu(menuMgr, viewer) ovviamente registra un pop-up menu con l'id default per estensione e questa default id è pop-up per una vista o editor tra un altro plugin. Nell'ordine di estensione la parte target deve pubblicare a ciascun menu la chiamata registerContextMenu. Una volta uscito dalla workbench si dovrà automaticamente inserire ciascuna action extension esistente.
viewer.getTable().removeAll() banalmente vuol dire che mi ritorna questa tabella in particolare il contro sulla visualizzazione tabellare.
Ora vado a trattare la creazione della tabella, che si trova nel metodo private Table createMainTable(Composite parent), quindi
Table table=newTable(parent,SWT.FULL_SELECTION) vale a dire creo una nuova tabella dando come parametri parent e il valore dello stile descritto dove per quest'ultimo intendo lo stile riguardante il comportamento del riempimento delle righe, passando poi a TableLayout layout = new TableLayout() cioè viene creato un nuovo stile di layout, ergo table.setLayout(layout) setto il layout che è associato con il receiver che deve essere passato per argomento che non deve mai essere nullo.
Passo ora a descrivere la visualizzazione dello header, pare ovvio table.setHeaderVisible(true), dopodiché nomino le stringhe che deve contenere lo header come viene dichiaratamente chiesto in questo punto
String[] header={"EDP type","EDP name","Descriptor","Compilation unit"}
poi creo una colonna parametrizzata da larghezza,minima larghezza in pixel e un boolean ridimensionamento cioè true se è possile ridimensionarla
layout.addColumnData(new ColumnWeightData(130,130,true))
per poi creare una nuova TableColumn rappresentante una colonna nella table Widget
TableColumn tc0 = new TableColumn(table,SWT.CENTER)
quindi setto il testo ricevuto, tc0.setText(header[0]) e poi faccio i seguenti due controlli
tc0.setAlignment(SWT.CENTER) e tc0.setResizable(true)
quindi svolgo un ciclio for per i 4 valori delle colonne iniziali cioè EDP type, EDP name, Descriptor ,Compilation unit, per fare in modo che ogni colonna abbia "tutto corretto", vale a dire:

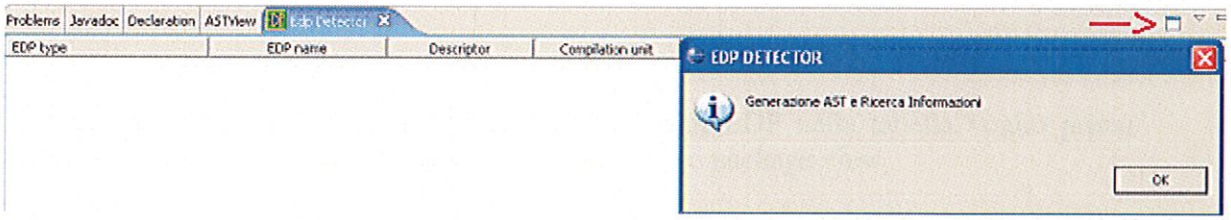
```
for(int i=1 ;i<4;i++)
{
        layout.addColumnData(new ColumnWeightData(60,60,true));
        TableColumn tc1 = new TableColumn(table,SWT.CENTER);
        tc1.setText(header[i]);
```

```
                tc1.setAlignment(SWT.CENTER);
                tc1.setResizable(true);
                //TODO aggiunta
                tc1.addSelectionListener(new MySelectionListener(i+1));
            }
            return table;
        }
```

per avere come risultato nell'Edp Detector la seguente visualizzazione tabellare



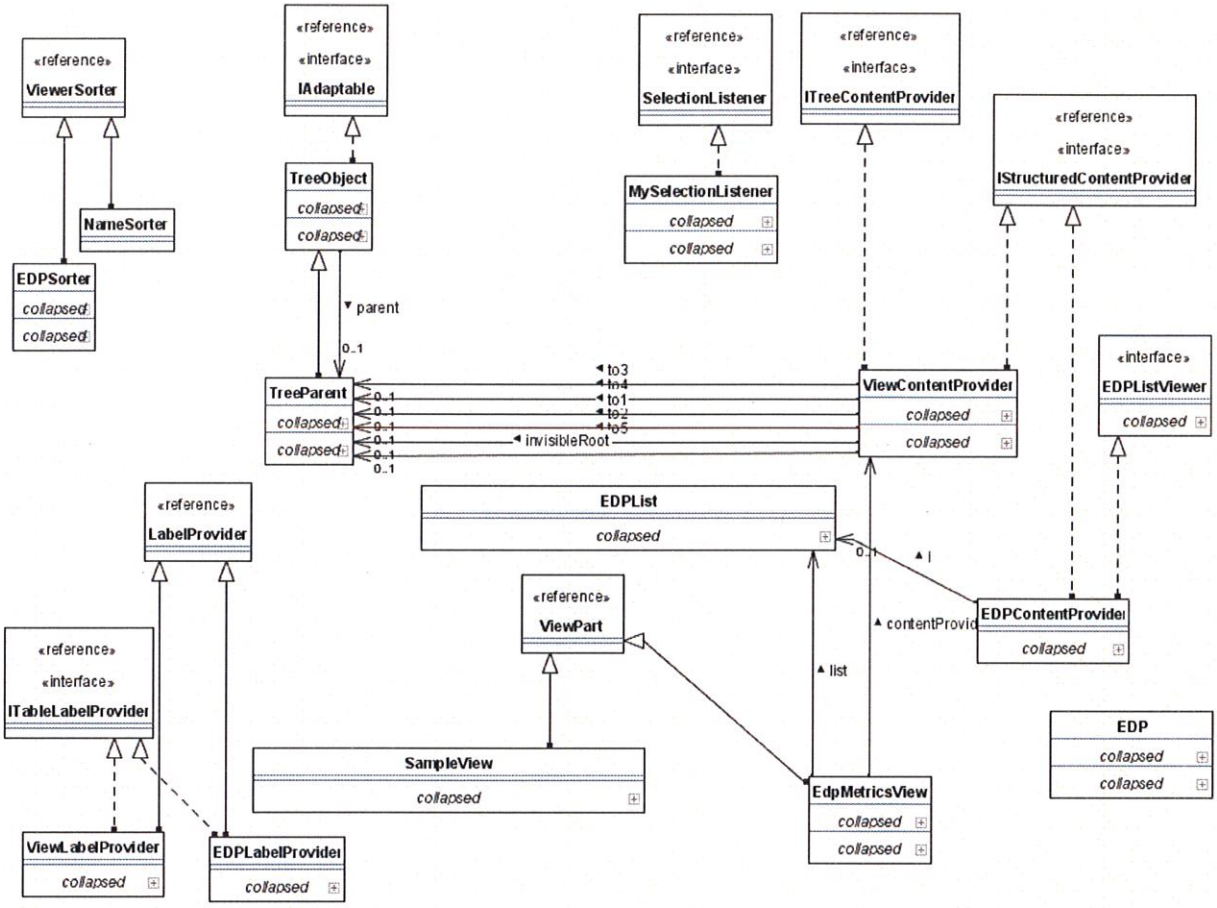
Penso che sia opportuno anche mostrare come sia possibile che quando si clicca il pulsante (evidenziato dalla freccia rossa) appaia la scritta EDP DETECTOR , per capirci meglio




```
private void hookDoubleClickAction() {
        viewer.addDoubleClickListener(new IDoubleClickListener() {
                public void doubleClick(DoubleClickEvent event) {
                        doubleClickAction.run();
                }
        });
    }

private void showMessage(String message) {
        MessageDialog.openInformation(
                viewer.getControl().getShell(),
                "EDP DETECTOR",
                message);
    }

private void makeActions() {
        action1 = new Action() {
                public void run() {
                        cu = new CompilationUnit[1000];
                        int i;
```

```
                showMessage("Generazione AST e Ricerca Informazioni");
                try {
                        //Otengo tutte le compilation unit aperte
                        getAllPrjCompilationUnits();
                }catch (CoreException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                viewer.getTable().removeAll();
                for(i=0;i<cu.length;i++)
                {       try {
                                if(cu[i]!=null)
                                        addNode(cu[i]); //aggiungo i nodi ;)
                        } catch (Exception e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }
        }
};
```

A riguardo di come vengono individuati e inseriti gli EDP nella tabella voglio prima mostrare un diagramma UML riguardante il complesso package view



quindi dare la seguente spiegazione a riguardo il recupero e l'inserimento degli EDP nella tabella:
gli EDP vengono recuperati nella classe EdpMetricsView, da cui parte la ricerca negli alberi sintattici usando vari visitor. Questi visitor riempiono degli ArrayList con degli oggetti che contengono i dati relativi agli EDP che individuano. Questi oggetti derivano da una classe base GenericInformation. Questi oggetti vengono poi estratti dagli ArrayList e convertiti in oggetti EDP, contenenti le stesse informazioni. Questi vengono poi inseriti in un contenitore, istanza della classe EDPList, che fornisce i metodi di accesso necessari per la visualizzazione degli EDP come tabella.
Una volta creati gli oggetti EDP, questi vengono inseriti in un contenitore, istanza della classe EDPList. Questa fornisce i comuni metodi di aggiunta, eliminazione e accesso, più la gestione di un insieme di EDPListViewer. EDPListViewer è un'interfaccia che definisce due metodi, uno di aggiunta e uno di rimozione. Ogni volta che nell'EDPList un elemento viene aggiunto o eliminato, viene chiamato il metodo corrispondente su tutti i viewer gestiti.
L'interfaccia EDPListViewer viene implementata dalla classe EDPContentProvider, che funge da ponte tra una EDPList e la tabella dove vengono riportati gli EDP individuati. Infatti questa classe implementa anche l'interfaccia IStructuredContentProvider, usata dalla table per ottenere il proprio contenuto. Quando l'EDPContentProvider viene notificato riguardo un'aggiunta o una rimozione, esegue l'operazione corrispondente sulla tabella.

La tabella, di per sé usa due oggetti: uno è EDPContentProvider, l'altro è EDPLabelProvider che fornisce le intestazioni delle colonne.

Adesso mostro tramite l'immagine successiva il risultato finale

EDP type	EDP name	Descriptor
Object Element	Create Object	QualifiedName
Object Element	Create Object	QualifiedName
Object Element	Create Object	QualifiedName
Object Element	Create Object	QualifiedName
Object Element	Create Object	QualifiedName
Object Element	Create Object	QualifiedName
Object Element	Create Object	QualifiedNameFinder
Type Relation	Inheritance	QualifiedNameFinder INHERIT GenericVisitor
Object Element	Retrieve	RADIO - FIELD
Object Element	Retrieve	RADIO - FIELD
Object Element	Retrieve	RADIO - FIELD
Object Element	Retrieve	READ_ONLY - FIELD
Object Element	Retrieve	READ_ONLY - FIELD
Object Element	Create Object	RetrieveCounter
Type Relation	Inheritance	RetrieveCounter INHERIT GenericVisitor
Object Element	Create Object	RetrieveInformation
Object Element	Create Object	RetrieveInformation
Object Element	Create Object	RetrieveInformation
Object Element	Create Object	RetrieveInformation
Object Element	Create Object	RetrieveInformation
Type Relation	Inheritance	RetrieveInformation INHERIT GenericInformation
Object Element	Retrieve	SUPPRESS_ARRAY_CREATION - FIELD
Object Element	Create Object	Separator
Object Element	Create Object	Separator
Object Element	Create Object	Status
Object Element	Create Object	String
Object Element	Create Object	SuperFieldAccessFinder
Type Relation	Inheritance	SuperFieldAccessFinder INHERIT GenericVisitor
Object Element	Create Object	SuperMethodInvocationFinder
Type Relation	Inheritance	SuperMethodInvocationFinder INHERIT GenericVisitor
Object Element	Create Object	Table
Object Element	Create Object	TableColumn
Object Element	Create Object	TableColumn
Object Element	Create Object	TableColumn
Object Element	Create Object	TableLayout
Object Element	Create Object	TableViewer
Object Element	Create Object	Text
Object Element	Create Object	Text

- Nella vista EDP Detector eliminare il secondo pulsante

Questo punto è semplice perché bisognava solamente capire nel sorgente quali erano le righe di codice riguardanti il secondo pulsante ed eliminarle. Il file che sono andato a modificare è EdpMetricsView.java

- Cambiare icona al primo pulsante (qualcosa sempre in blu con la scritta EDP)

Anche in questo punto si trattava solamente di capire dove veniva creato il pulsante e come veniva visualizzata l'icona per poi andarla a trattare. Il file che ho modificato è EdpMetricsView.

- Cambiare il tooltip del pulsante in "Show EDPs of active project"

Come nei precedenti due punti anche qui si trattava di capire dove si effettua il tooltip per poi andarlo a modificare.

- Per ora viene utilizzata solo una Compilation Unit, quella attiva, fare sì che il metodo di analisi riceva in Input un Vettore o un array di Compilation Unit e che effetti le sue analisi su tutte, a questo punto aggiungere anche una ulteriore colonna, all'interno della

tabella, Compilation Unit (dove ci va il nome della compilation Unit). Premendo il pulsante dovrà ovviamente avvenire l'analisi dell'intero progetto.

Questo punto non è assolutamente banale in più ci ho perso parecchio tempo (più di due settimane) in quanto ho avuto un incomprensione con il mio correlatore su come lo voleva fatto perché in un primo momento lo avevo svolto e funzionava solamente che pensavo al utilizzo dell'EdpDetector per ogni file aperto e se ne aprivo un altro doveva ricalcolare oltre al file precedente anche quello appena aperto e gli avevo dato come massimo mille files che si potevano aprire. Una volta finita la modifica ho scoperto che ciò non andava bene perché l'analisi deve essere fatta una volta sola analizzando tutti i files contenuti nella cartella.
Adesso mostro i metodi creati all'interno di EdpMetricsView adeguatamente commentati

```
private ITextEditor[] getAndCheckOpenedITestEditors() throws CoreException
{
      int i,count=0;
      ITextEditor[] returnValue =new ITextEditor[30];
         //Prendo tutti gli editor
      IEditorPart[] part = EditorUtility.getAllEditors();
      for(i=0;i<part.length;i++)
      {
         if (part[i] instanceof ITextEditor)
         {
          //Aggiorno array degli editor
          returnValue[count] = (ITextEditor) part[i];
             count++;
          }
      }
      //Restituisco gli editor o eccezione se non ce n'è
      if(count==0)
         throw new CoreException(getErrorStatus("Text editor not valid or null", null));
      else
         return returnValue;
}

private IOpenable[] getAndCheckAllIOpenables(ITextEditor[] editors) throws CoreException
    {
      IOpenable[] openables = new IOpenable[30];
      IOpenable openable;
      int i,count=0;
      for(i=0;i<editors.length;i++)
      {
                if(editors[i]!=null)
                {       //Restituisco il java element dell'editor -> perchè nell' ast c'è la
                            //classe java element
                        openable = EditorUtility.getJavaInput(editors[i]);
                   if (openable != null)
                   {
```

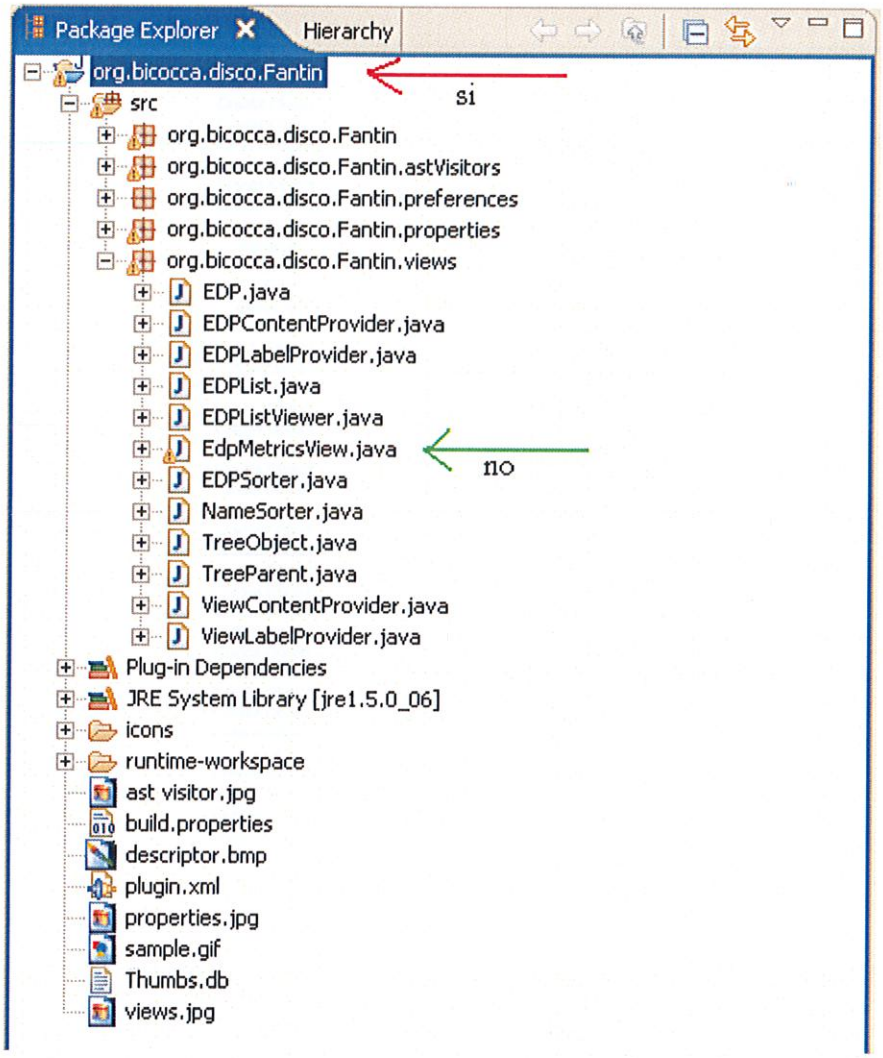
```
                        openables[count]=openable;
                        count++;
                    }
                }
            }
            if(count!=0)
                return openables;
            else
            throw new CoreException(getErrorStatus("Editor not showing a CU or classfile",null));
    }

private CompilationUnit[] getAllCompilationUnits() throws CoreException
        {
            CompilationUnit[] units = new CompilationUnit[30];
            int i;
                //Ottengo tutti gli editor di testo aperti
            ITextEditor[] editors = getAndCheckOpenedITestEditors();
            //Ottengo tutti gli oggetti apribili
            IOpenable[] openables = getAndCheckAllIOpenables(editors);
            for(i=0;i<openables.length;i++)
                if(openables[i]!=null)
                        //Carico le compilation unit non nulle

        units[i]=(CompilationUnit)getAndSetSourceOnParser(openables[i]).createAST(null);
//CREATE AST e restituisce la root dell'albero AST
            return units;
    }

private void getAllPrjCompilationUnits() throws CoreException
    {
                IJavaProject myJavaProject=null;
                /*
                 * recupero la directory di lavoro da Eclipse
                 *          */
                IWorkbench workbench = PlatformUI.getWorkbench();
                /*
                 * recupero le finestre attive dalla directory di lavoro  - package explorer
                 */
            IWorkbenchWindow[] workbenchWindows = workbench.getWorkbenchWindows();
            /*Finestra package explore /hierarchy -> Cerco tutte le finestre contenute in essa
             * Parto dalla finestra 0 -> guardo quale finestra è il package explorer -> Carico la
             * selezione verificando che sia un progetto */



            for (int j = 0; j < workbenchWindows.length; j++)
            {
              /*
```

```
 * recupero la pagina per la finestra corrente
 */
IWorkbenchPage page = workbenchWindows[j].getPages()[0];
/*
 * recupero il riferimento alla vista attiva della finestra
 */
IViewReference[] ref = page.getViewReferences();
/*
 * cerco dal package di explorer la vista
 *                  */
for(int k = 0; k < ref.length; k++)
{
 /*
  * trovo il package di explorer
  */
 if(ref[k].getId().equals("org.eclipse.jdt.ui.PackageExplorer"))
 {
  /*
   * recupero l'attuale vista attiva in Eclipse
   */
IWorkbenchPart part = page.getActivePart();
/*
 * "Scrivo" sulla pagina di explorer
 */
  page.activate(ref[k].getPart(false));
  /*
   * If  (struttura di selezione nel package di explorer)
   */
  if ( ref[k].getPage().getSelection() instanceof
       org.eclipse.jface.viewers.IStructuredSelection)
  { /*
    * If (seleziona la risorsa java da recuperare nel progetto)
    */
   if(((IStructuredSelection)ref[k].getPage().getSelection()).getFirstElement()
        instanceof IJavaElement)
   {
         myJavaProject = ((IJavaElement)((IStructuredSelection)ref[k].getPage().
           getSelection()).getFirstElement()).getJavaProject();
   }
   /*
    * selezione non valida
    */
   else return;
  }


  // "scrive" la finestra attiva in Eclipse
  page.activate(part);
```

```
            }
          }
        }
        if(myJavaProject!=null)
        {
          count=0;
          //Ricerco le compilation unit del progetto (files .java)
          exploreCU(myJavaProject);
        }
        else
        {
          //Errore
          return;
        }
}

    //Metodo ricorsivo per analizzare tutti i JavaElement del progetto

    /*
     * NB: I files java possono essere solo delle CompilationUnit, quindi
     * se trovo qualunque altro tipo di JavaElement ne analizzo i figli
     * alla ricerca di file java
     */
    private void exploreCU(IJavaElement myJavaElement)
    {
        //CompilationUnit[] cu;
        IJavaElement[] children;
        int i;
        //int count=0;
        try
        {
         //Switch per decidere il tipo di elemento
            switch(myJavaElement.getElementType())
         {
            //Compilation unit
            case(IJavaElement.COMPILATION_UNIT):
                //Prendo i figli
                children=((ICompilationUnit)myJavaElement).getChildren();
                //System.out.println(myJavaElement.getElementName());
                //Se è un file java sono su una cu di interesse

              if(myJavaElement.getElementName().toLowerCase().endsWith(".java"))
                {       //Aggiungo la cu tra quelle da analizzare per gli EDP

cu[count]=(CompilationUnit)getAndSetSourceOnParser(((IOpenable)myJavaElement)).createAST(null);
                    count++;
                }
```

```
                else
                        //Analizzo i figli per vedere se la cu contiene files .java
                        if(children!= null)
                                for(i=0;i<children.length;i++)
                                        exploreCU(children[i]);
                return;
        //Progetto
        case(IJavaElement.JAVA_PROJECT):
                //Analizzo i figli
                children=((IJavaProject)myJavaElement).getChildren();
                if(children!= null)
                        for(i=0;i<children.length;i++)
                                exploreCU(children[i]);
                return;
        //Package
        case(IJavaElement.PACKAGE_FRAGMENT):
        //      Analizzo i figli
                children=((IPackageFragment)myJavaElement).getChildren();
                if(children!= null)
                        for(i=0;i<children.length;i++)
                                exploreCU(children[i]);
                return;
        //Modello
        case(IJavaElement.JAVA_MODEL):
        //      Analizzo i figli
                children=((IJavaModel)myJavaElement).getChildren();
                if(children!= null)
                        for(i=0;i<children.length;i++)
                                exploreCU(children[i]);
                return;
        //Frammeto di package
        case(IJavaElement.PACKAGE_FRAGMENT_ROOT):
                //      Analizzo i figli
                children=((IPackageFragmentRoot)myJavaElement).getChildren();
                if(children!= null)
                        for(i=0;i<children.length;i++)
                                exploreCU(children[i]);
                return;
        default:
                return;
        }
      }catch(Exception e){/*e.printStackTrace();*/return;}
}
```

per fare capire meglio il lavoro svolto ho deciso di mostrarlo con la seguente figura con la quale si capisce che deve essere svolta l'analisi dell'intero progetto e non l'analisi dei singoli file che vengono aperti



Infine, aggiunto una nuova colonna compilation unit la quale deve contenere il nome della compilation unit.

EDP type	EDP name	Descriptor	Compilation unit
Object Element	Create Object	QualifiedName	EdpManagementPropertyPage.java
Object Element	Create Object	QualifiedName	EdpManagementPropertyPage.java
Object Element	Create Object	QualifiedName	EdpManagementPropertyPage.java
Object Element	Create Object	QualifiedName	EdpManagementPropertyPage.java
Object Element	Create Object	QualifiedName	EdpManagementPropertyPage.java
Object Element	Create Object	QualifiedName	EdpMetricsView.java
Object Element	Create Object	QualifiedNameFinder	RetrieveCounter.java
Type Relation	Inheritance	QualfiedNameFinder INHERIT GenericVisitor	QualfiedNameFinder.java
Object Element	Retrieve	RADIO - FIELD	EdpManagementPropertyPage.java
Object Element	Retrieve	RADIO - FIELD	EdpManagementPropertyPage.java
Object Element	Retrieve	RADIO - FIELD	EdpManagementPropertyPage.java
Object Element	Retrieve	READ_ONLY - FIELD	EdpManagementPropertyPage.java
Object Element	Retrieve	READ_ONLY - FIELD	EdpManagementPropertyPage.java
Object Element	Create Object	RetrieveCounter	EDPInformationCollector.java
Type Relation	Inheritance	RetrieveCounter INHERIT GenericVisitor	RetrieveCounter.java
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java
Type Relation	Inheritance	RetrieveInformation INHERIT GenericInformation	RetrieveInformation.java
Object Element	Retrieve	SUPPRESS_ARRAY_CREATION - FIELD	ObjectCreationCounter.java
Object Element	Create Object	Separator	EdpMetricsView.java
Object Element	Create Object	Separator	EdpMetricsView.java
Object Element	Create Object	Status	EdpMetricsView.java
Object Element	Create Object	String	EdpMetricsView.java
Object Element	Create Object	SuperField4ccessFinder	RetrieveCounter.java
Type Relation	Inheritance	SuperFieldAccessFinder INHERIT GenericVisitor	SuperFieldAccessFinder.java
Object Element	Create Object	SuperMethodInvocationFinder	RetrieveCounter.java
Type Relation	Inheritance	SuperMethodInvocationFinder INHERIT GenericVisitor	SuperMethodInvocationFinder.java
Object Element	Create Object	Table	EdpMetricsView.java
Object Element	Create Object	TableColumn	EdpMetricsView.java
Object Element	Create Object	TableColumn	EdpMetricsView.java
Object Element	Create Object	TableColumn	EdpMetricsView.java
Object Element	Create Object	TableLayout	EdpMetricsView.java
Object Element	Create Object	TableViewer	EdpMetricsView.java
Object Element	Create Object	Text	EdpManagementPropertyPage.java
Object Element	Create Object	Text	EdpManagementPropertyPage.java

I punti precedenti sono stati scritti apposta entrando nel dettaglio per fare capire che non sono stati banalmente "copiati da dei sorgenti disponibili su Internet" ma si è cercata di capire la teoria che ci stava alla base (se non si conosce non si domina…) per poi cercare di creare un dignitoso "adattamento" del sorgente modificato.
Siccome i successivi punti sono molto complessi e questa è una tesi di laurea di primo livello in informatica e non un articolo scritto da un ricercatore, ho cercato di non entrare troppo nel dettaglio, ma di realmente di fare capire a chi osa leggere questa tesi, in "maniera umana" il lavoro svolto dallo studente Stefano Fantin.

• Permettere di gestire le due versioni di Java 1.4 e 1.5 (JLS2 e JLS3): aggiungendo questa funzionalità sulla pagina di properties (ad esempio con un radio button o una check box). Meglio ancora recuperare dalle caratteristiche del progetto il tipo di codice contenuto e consigliare l'utente in tal senso.

Prima di spiegare quello che ho fatto voglio fare capire cosa si intende per JLS2 e JLS3.
JLS2 è una costante che sta per Java Language Specification Second edition, i linguaggi compatibili sono quelli fino alla versione J2SE 1.4.
JLS3 è una costante che sta per Java Language Specification Third Edition, serve in pratica a gestire J2SE 1.5.

Quindi per quanto riguarda l'analisi degli EDP si avrà che:

Create Object
Coincide sostanzialmente con la creazione di un oggetto tramite l'operatore new.

AST
La creazione delle classi è presente nei nodi di tipo ClassInstanceCreation in JLS2 la grammatica è la seguente:

```
ClassInstanceCreation:
        [ Expression . ] new Name
            ( [ Expression { , Expression } ] )
            [ AnonymousClassDeclaration ]
```

in JLS3 invece:

```
ClassInstanceCreation:
        [ Expression . ]
            new [ < Type { , Type } > ]
            Type ( [ Expression { , Expression } ] )
            [ AnonymousClassDeclaration ]
```

La creazione degli Array è presente nei nodi di tipo ArrayCreation

In JSL2:

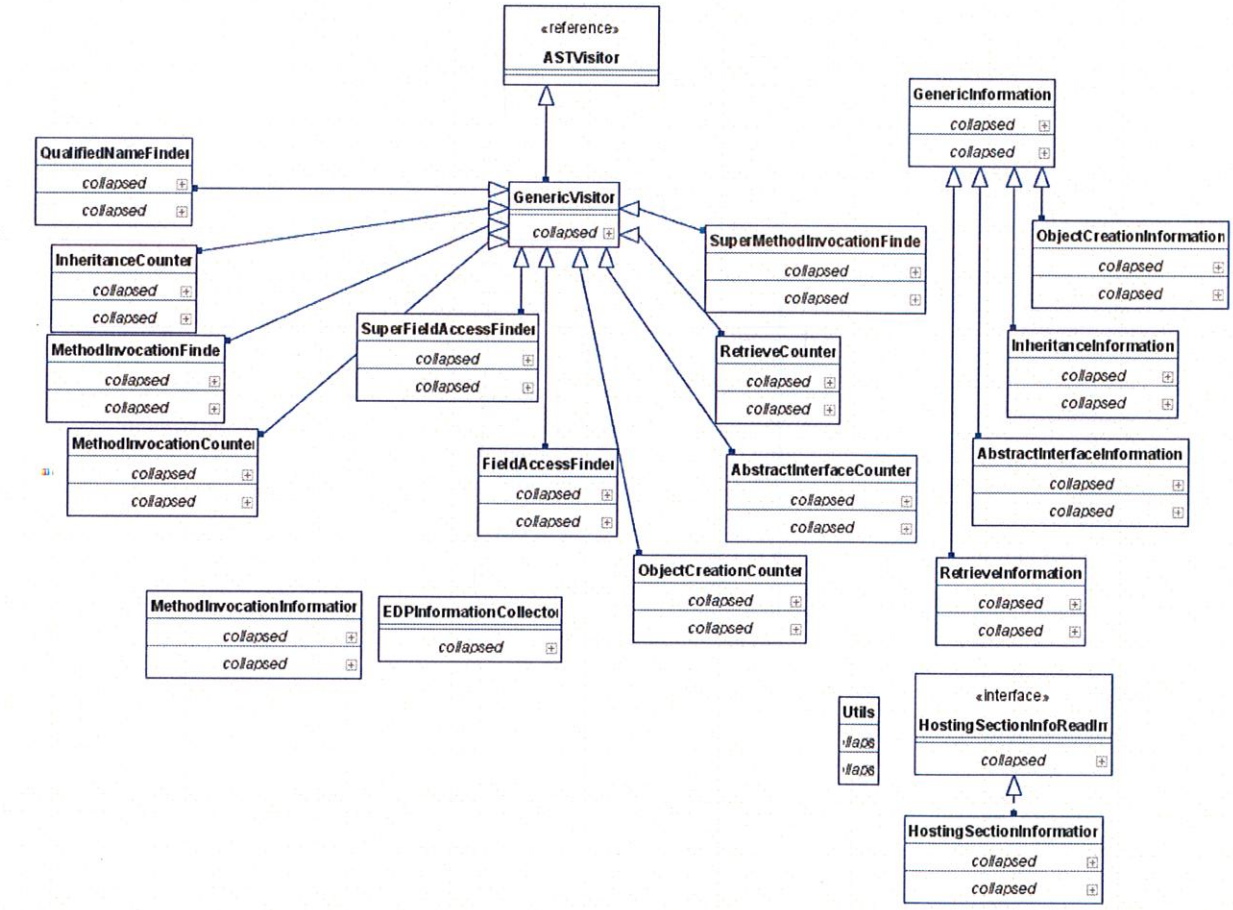
```
ArrayCreation:
    new PrimitiveType [ Expression ] { [ Expression ] } { [ ] }
    new TypeName [ Expression ] { [ Expression ] } { [ ] }
    new PrimitiveType [ ] { [ ] } ArrayInitializer
    new TypeName [ ] { [ ] } ArrayInitializer
```

In JSL3:

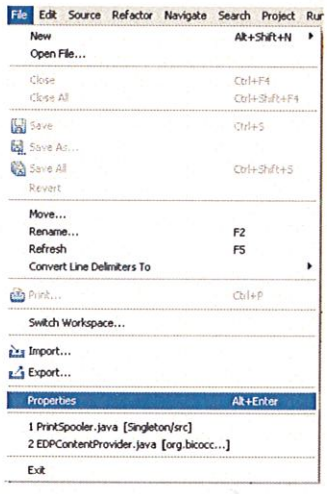
```
ArrayCreation:
    new PrimitiveType [ Expression ] { [ Expression ] } { [ ] }
    new TypeName [ < Type { , Type } > ]
        [ Expression ] { [ Expression ] } { [ ] }
    new PrimitiveType [ ] { [ ] } ArrayInitializer
    new TypeName [ < Type { , Type } > ]
        [ ] { [ ] } ArrayInitializer
```

La gestione dei due livelli di JLS (2 e 3) avviene in due momenti: nella creazione dell'albero sintattico e nella visita.
La creazione dell'AST comporta la scelta di quale livello di JLS deve usare il parser; questo viene determinato basandosi sul valore delle opzioni settate per il progetto (vedi nelle immagini sottostanti). Se l'opzione JLS_LEVEL è settata a un valore specifico (JLS2 o JLS3) viene usato quel livello, mentre se non è settata o è settata sul valore che indica di usare l'opzione del compilatore, viene applicata quest'ultima.

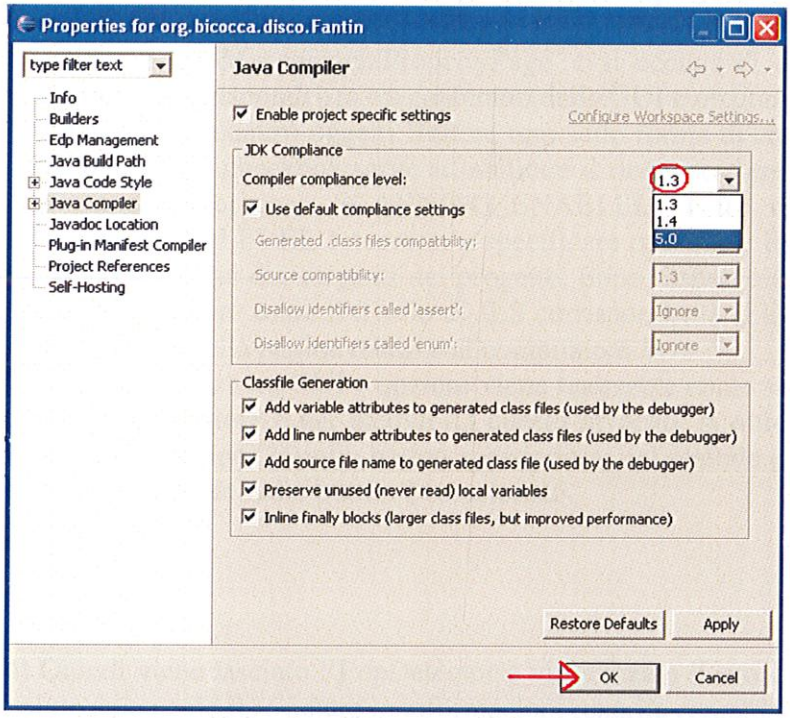
Siccome poi il parser produce alberi diversi a seconda di quale livello è in uso, durante la visita è necessario adattare il codice a gestire entrambi i livelli, a seconda del caso. È stato necessario modificare solo due visitor, ObjectCreationCounter e InheritanceCounter che sono visibili nella successiva figura raffigurante il package astVisitor



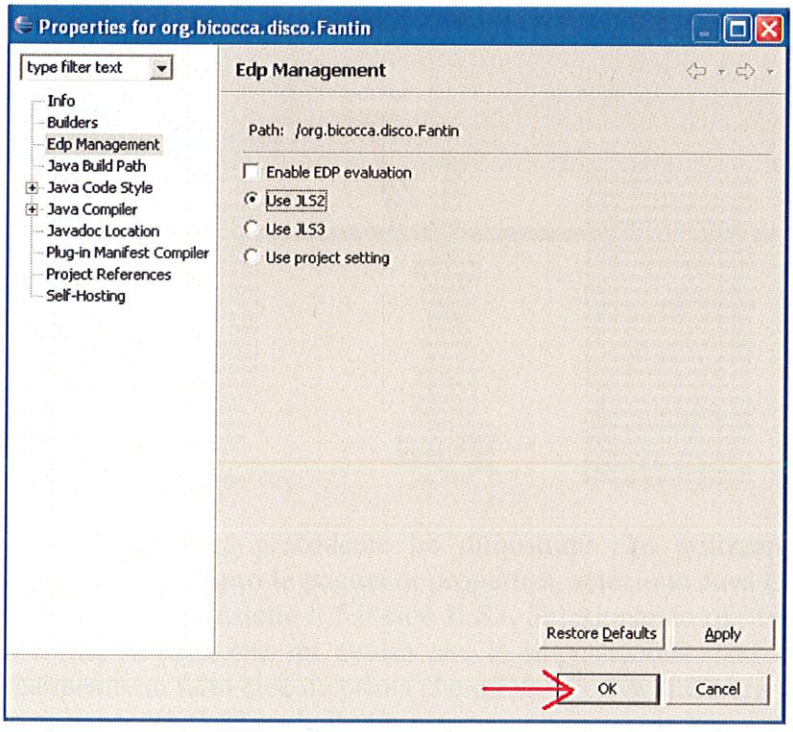
Adesso cerco di spiegare come viene utilizzato JLS2 e JLS3 in fase di esecuzione:
1) apro la pagina di properties riguardante l'intero progetto



2) seleziono Java Compiler e gli imposto che livello utilizzare, quindi 1.3 e 1.4 sono JLS2 mentre 5.0 è JLS3. Inizialmente ho selezionato 1.3

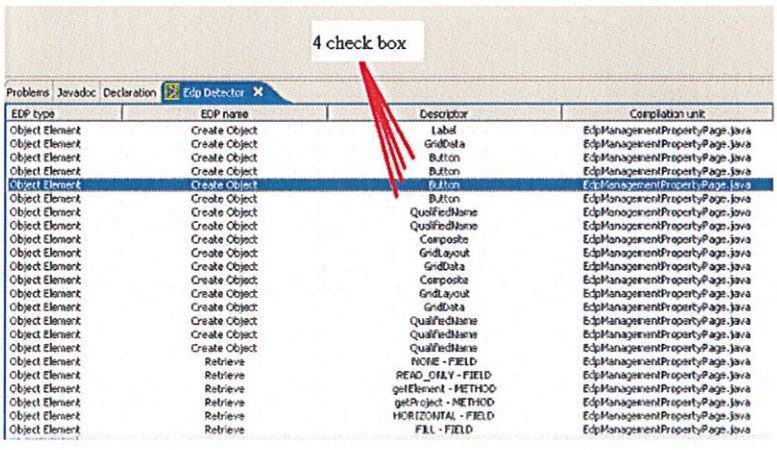


3) quindi seleziono l'Edp Managment e seleziono JLS2

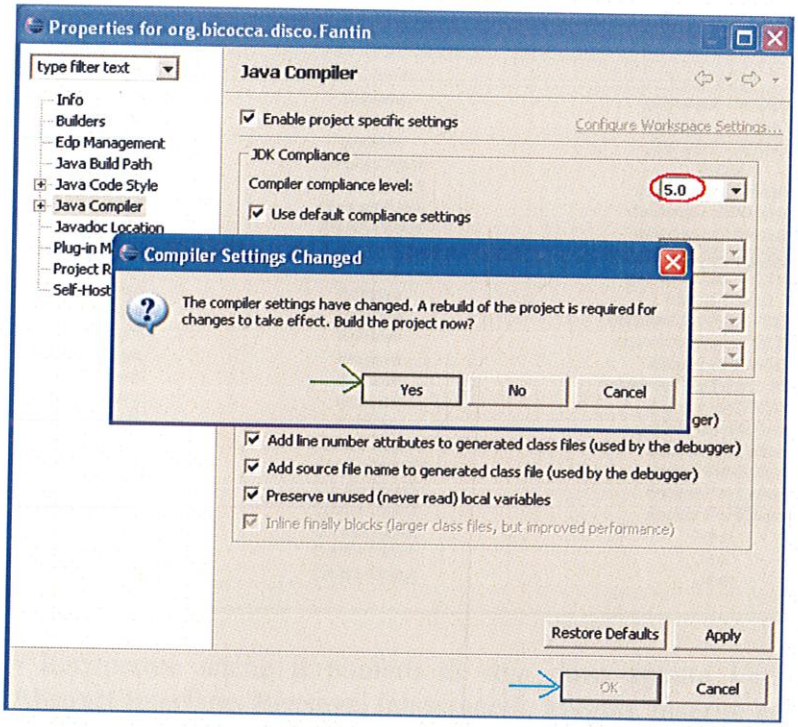


Mi vorrei soffermare su questa figura in quanto vorrei dare la nozione teorica che permette appunto la visualizzazione di tale pagina di properties.
La property page, disponibile per gli oggetti di tipo progetto, è stata realizzata tramite l'estensione org.eclipse.ui.propertyPages, estendendo la classe PropertyPage. La classe in questione è EdpManagementPropertyPage, e si occupa di creare l'interfaccia grafica e realizzare la corrispondenza tra contenuto della GUI e opzioni salvate per il progetto.
Vengono creati quattro oggetti Button, uno sotto forma di checkbox e tre sotto forma di radio button. La checkbox serve ad abilitare il riconoscimento degli EDP nel progetto, e corrisponde all'opzione SCANNING_ENABLED. I tre radio button corrispondono all'opzione JLS_LEVEL e servono a specificare con quale livello di JLS verranno creati gli alberi sintattici dei sorgenti del progetto. Sono disponibili tre possibilità: usare JLS2, usare JLS3 oppure usare il livello di JLS corrispondente al livello di sorgente specificato come opzione nella pagina relativa al compilatore Java.
La corrispondenza tra GUI e opzioni viene realizzata come segue: quando viene creata la GUI, nel costruttore e nei metodi da questo invocati, si ottiene l'opzione di progetto, se esiste, e si seleziona il radio button appropriato; nel metodo performOK si setta l'opzione a seconda di quale radio button è selezionato.

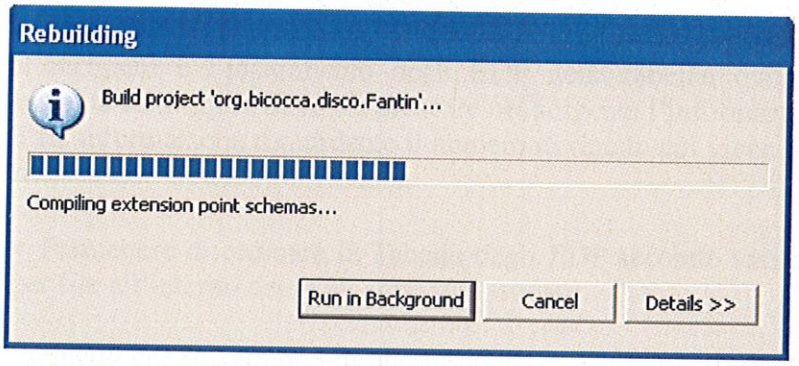
4) Quindi viene lasciato l'EdpDetector e visualizzato il suo contenuto. In questo caso ho voluto solamente fare notare le informazioni reperite riguardanti i quattro bottoni presenti nella figura precedente, cioè che appartengono alla compilation unit EdpManagmentPropertyPage e riguardano la creazione degli oggetti quindi sono di tipo Object Element



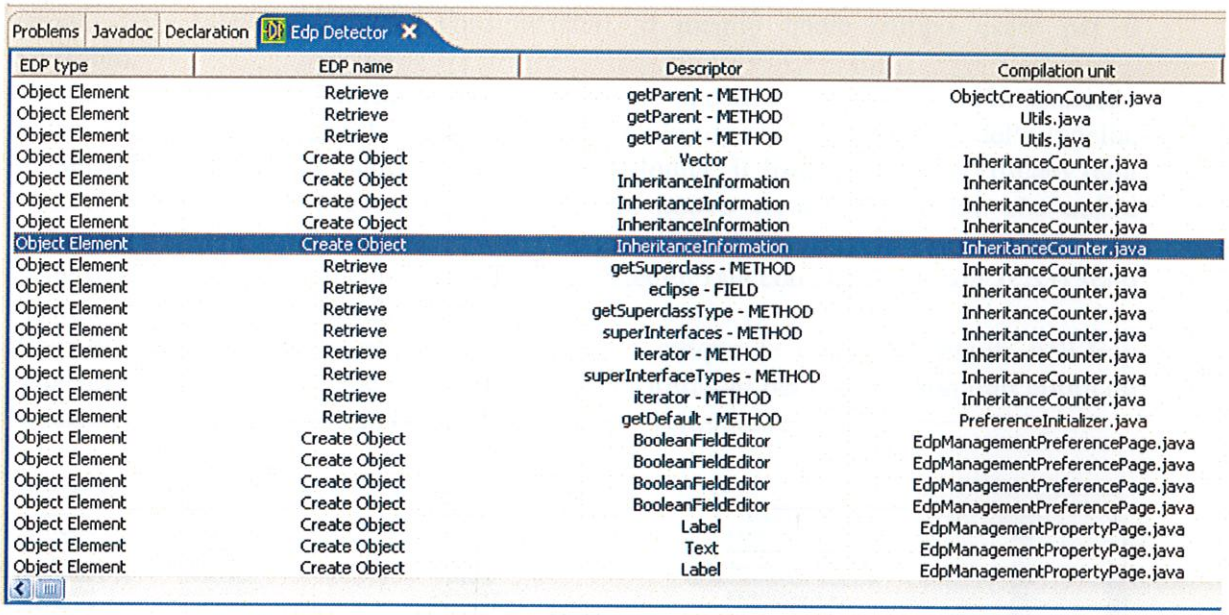
5) Con la figura precedente ho dimostrato che utilizzando la JLS2 l'EdpDetector funziona. Ora riapro la pagina di properties, seleziono Java Compiler e gli assegno come livello di compilazione il 5.0 cioè JLS3. Selezionando ok (freccia azzura) mi appare una seconda finestra che mi avvisa che le impostazioni sono cambiate, che è quello che realmente ho fatto cioè da prima che mi analizzava JLS2 ora gli dico di svolgere lo stesso progetto sotto JLS3, quindi clicco su yes.



6) Quindi mi vengono ricalcolate le stesse informazioni dello stesso progetto in JLS3



7) Quindi seleziono il progetto, clicco sul pulsante che mi genera l'AST e ricerca le informazioni, viene svolta l'analisi dell'intero progetto in JLS3 e produce il corretto risultato come evidente dalla successiva figura



Problems | Javadoc | Declaration | Edp Detector

EDP type	EDP name	Descriptor	Compilation unit
Object Element	Retrieve	getParent - METHOD	ObjectCreationCounter.java
Object Element	Retrieve	getParent - METHOD	Utils.java
Object Element	Retrieve	getParent - METHOD	Utils.java
Object Element	Create Object	Vector	InheritanceCounter.java
Object Element	Create Object	InheritanceInformation	InheritanceCounter.java
Object Element	Create Object	InheritanceInformation	InheritanceCounter.java
Object Element	Create Object	InheritanceInformation	InheritanceCounter.java
Object Element	Create Object	InheritanceInformation	InheritanceCounter.java
Object Element	Retrieve	getSuperclass - METHOD	InheritanceCounter.java
Object Element	Retrieve	eclipse - FIELD	InheritanceCounter.java
Object Element	Retrieve	getSuperclassType - METHOD	InheritanceCounter.java
Object Element	Retrieve	superInterfaces - METHOD	InheritanceCounter.java
Object Element	Retrieve	iterator - METHOD	InheritanceCounter.java
Object Element	Retrieve	superInterfaceTypes - METHOD	InheritanceCounter.java
Object Element	Retrieve	iterator - METHOD	InheritanceCounter.java
Object Element	Retrieve	getDefault - METHOD	PreferenceInitializer.java
Object Element	Create Object	BooleanFieldEditor	EdpManagementPreferencePage.java
Object Element	Create Object	BooleanFieldEditor	EdpManagementPreferencePage.java
Object Element	Create Object	BooleanFieldEditor	EdpManagementPreferencePage.java
Object Element	Create Object	BooleanFieldEditor	EdpManagementPreferencePage.java
Object Element	Create Object	Label	EdpManagementPropertyPage.java
Object Element	Create Object	Text	EdpManagementPropertyPage.java
Object Element	Create Object	Label	EdpManagementPropertyPage.java

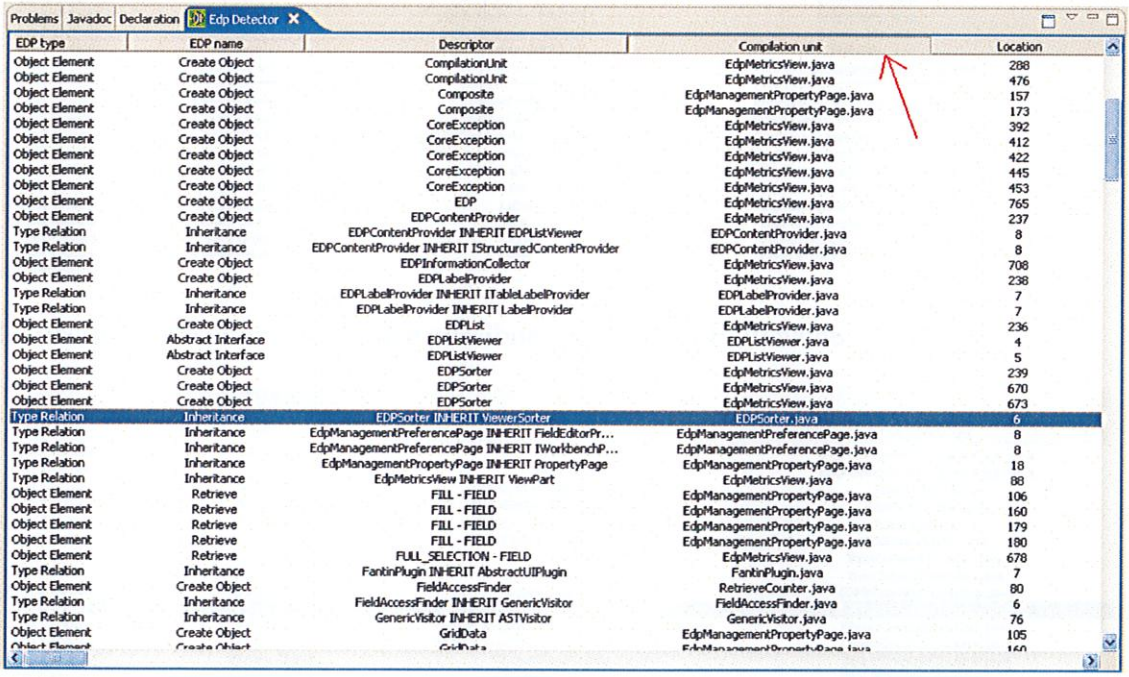
- Recuperare anche il numero di riga dove appare l'edp ricercato (Create Object, Abstract Interface, Retrieve, Inheritance). Meglio. Inserire nel codice sorgente perlomeno un commento che indichi il tipo di EDP trovato e possibilmente rendere agevole lo spostamento tra la tabella degli EDP e il codice

Nel punto riguardante la visualizzazione tabellare ho cercato di analizzare come avviene il recupero e l'inserimento degli EDP nella tabella, quindi dominando tali concetti bisogna solo capire dove sia presente nel sorgente l'informazione sulla riga.
Tale informazione riguardante il numero di riga in cui viene individuato l'EDP è presente nella classe GenericInformation.

- Permettere di ordinare la Tabella degli EDP secondo vari criteri: es. per tipo di EDP, per file all'interno dei quali sono stati trovati

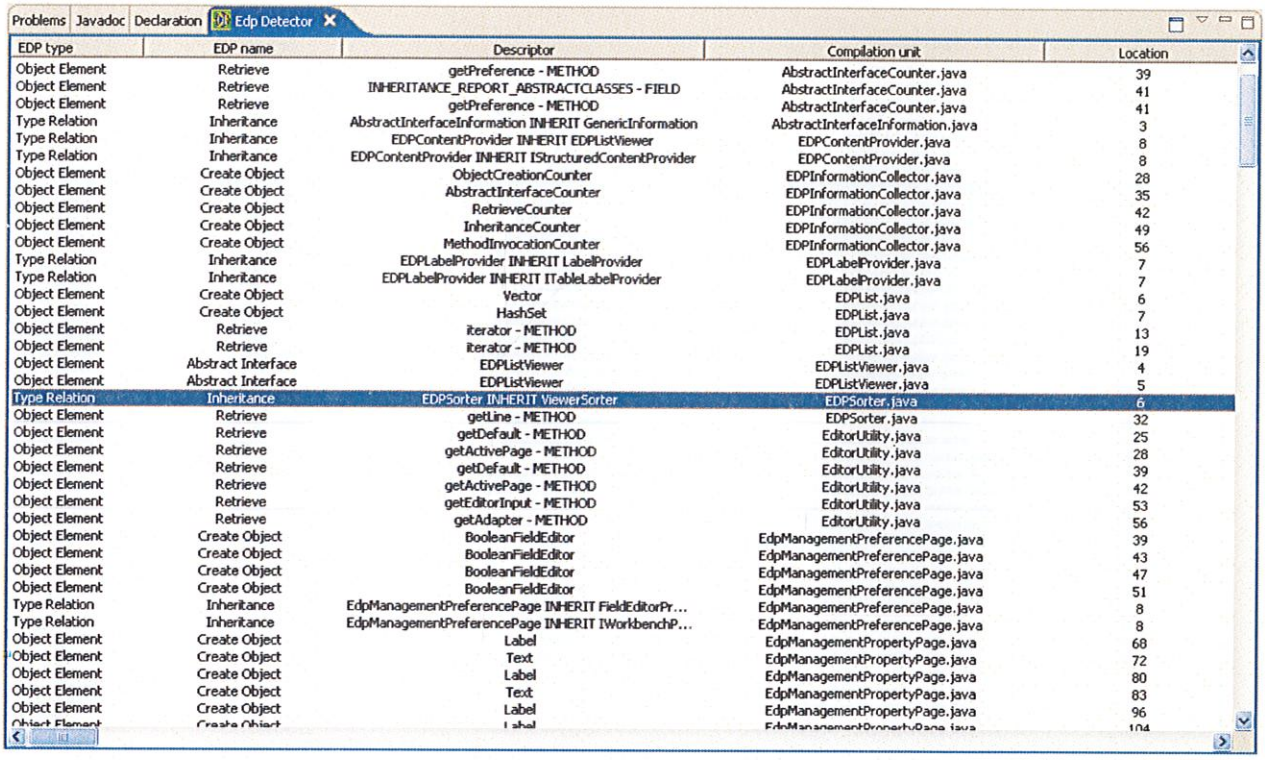
Premetto che lo sviluppo di questo punto non è proprio banale, infatti per conoscere tutta la teoria che ci stava dietro ho impiegato parecchio tempo, premetto anche che l'idea di svolgere questo punto fosse di costruire tre bottoni i quali dovevano ordinare EDP name, EDP type e Compilation unit. Riflettendo, ho deciso di non farlo così perché secondo me non erano comodi i bottoni, anche se effettivamente costruire un bottone e fargli fare un'azione è una cosa semplicissima, quindi volendomi complicare la vita ho deciso di svolgerlo in modo che cliccando sull'intestazione della colonna venisse il riordinamento della tabella. Mi è venuta questa idea perché è un modo sicuramente più raffinato che i bottoni e poi perché in Eclipse quando si deve riordinare qualcosa non esiste un bottone che lo faccia ma bisogna appunto cliccare sull'intestazione della colonna. Adesso analizzo come è stato possibile svolgere questo punto.
L'ordinamento della tabella viene realizzato tramite la classe EDPSorter, che estende la classe ViewerSorter. Responsabilità di questa classe è definire la relazione di ordine usata dall'algoritmo di ordinamento della tabella, implementato nel framework di Eclipse. Quando viene creato un EDPSorter, si deve specificare un parametro che indica su quale

campo della tabella effettuare l'ordinamento, o meglio quale campo usare per il confronto.
È possibile ordinare la tabella secondo un campo specifico cliccando sull'intestazione della colonna relativa. Questo è ottenuto registrando, durante la creazione della tabella, un SelectionListener per ogni intestazione di colonna; il listener viene notificato ogni volta che si clicca sulla colonna, e assegna un nuovo sorter alla tabella, che ordini secondo il campo specificato.
Nelle figure successive facendo partire l'Edp Detector (sia con JLS2 che con JLS3) sono mostrati degli esempi di riordinamento.
Premetto che è possibile riordinare per tutte e cinque le colonne presenti nell'Edp Detector; siccome volevo che i risultati venissero riordinati per compilation unit allora ho cliccato col mouse appunto su compilation unit (freccia rossa)



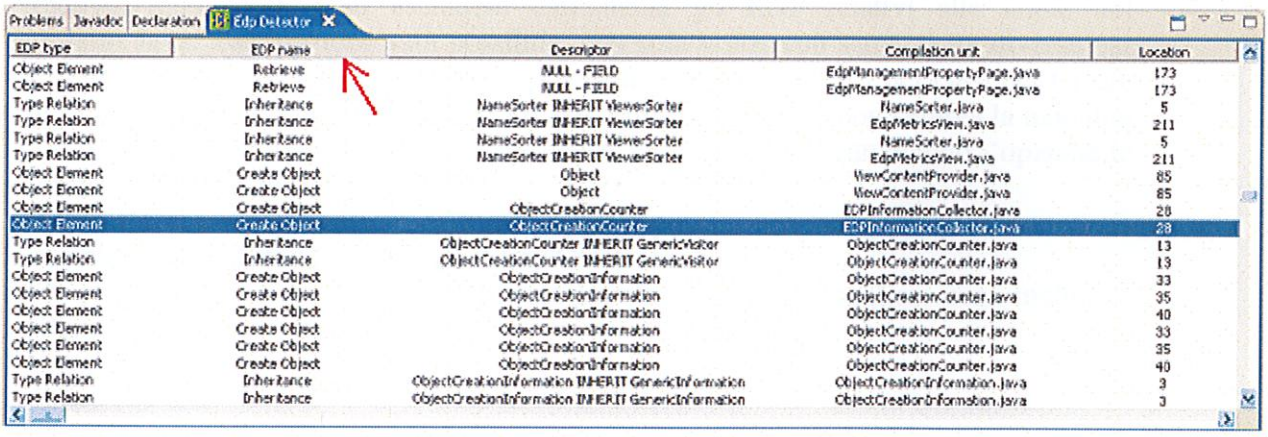
EDP type	EDP name	Descriptor	Compilation unit	Location
Object Element	Create Object	CompilationUnit	EdpMetricsView.java	288
Object Element	Create Object	CompilationUnit	EdpMetricsView.java	476
Object Element	Create Object	Composite	EdpManagementPropertyPage.java	157
Object Element	Create Object	Composite	EdpManagementPropertyPage.java	173
Object Element	Create Object	CoreException	EdpMetricsView.java	392
Object Element	Create Object	CoreException	EdpMetricsView.java	412
Object Element	Create Object	CoreException	EdpMetricsView.java	422
Object Element	Create Object	CoreException	EdpMetricsView.java	445
Object Element	Create Object	CoreException	EdpMetricsView.java	453
Object Element	Create Object	EDP	EdpMetricsView.java	765
Object Element	Create Object	EDPContentProvider	EdpMetricsView.java	237
Type Relation	Inheritance	EDPContentProvider INHERIT EDPListViewer	EDPContentProvider.java	8
Type Relation	Inheritance	EDPContentProvider INHERIT IStructuredContentProvider	EDPContentProvider.java	8
Object Element	Create Object	EDPInformationCollector	EdpMetricsView.java	708
Object Element	Create Object	EDPLabelProvider	EdpMetricsView.java	238
Type Relation	Inheritance	EDPLabelProvider INHERIT ITableLabelProvider	EDPLabelProvider.java	7
Type Relation	Inheritance	EDPLabelProvider INHERIT LabelProvider	EDPLabelProvider.java	7
Object Element	Create Object	EDPList	EdpMetricsView.java	236
Object Element	Abstract Interface	EDPListViewer	EDPListViewer.java	4
Object Element	Abstract Interface	EDPListViewer	EDPListViewer.java	5
Object Element	Create Object	EDPSorter	EdpMetricsView.java	239
Object Element	Create Object	EDPSorter	EdpMetricsView.java	670
Object Element	Create Object	EDPSorter	EdpMetricsView.java	673
Type Relation	Inheritance	EDPSorter INHERIT ViewerSorter	EDPSorter.java	6
Type Relation	Inheritance	EdpManagementPreferencePage INHERIT FieldEditorPr...	EdpManagementPreferencePage.java	8
Type Relation	Inheritance	EdpManagementPreferencePage INHERIT IWorkbenchP...	EdpManagementPreferencePage.java	8
Type Relation	Inheritance	EdpManagementPropertyPage INHERIT PropertyPage	EdpManagementPropertyPage.java	18
Type Relation	Inheritance	EdpMetricsView INHERIT ViewPart	EdpMetricsView.java	88
Object Element	Retrieve	FILL - FIELD	EdpManagementPropertyPage.java	106
Object Element	Retrieve	FILL - FIELD	EdpManagementPropertyPage.java	160
Object Element	Retrieve	FILL - FIELD	EdpManagementPropertyPage.java	179
Object Element	Retrieve	FILL - FIELD	EdpManagementPropertyPage.java	180
Object Element	Retrieve	FULL_SELECTION - FIELD	EdpMetricsView.java	678
Type Relation	Inheritance	FantinPlugin INHERIT AbstractUIPlugin	FantinPlugin.java	7
Object Element	Create Object	FieldAccessFinder	RetrieveCounter.java	80
Type Relation	Inheritance	FieldAccessFinder INHERIT GenericVisitor	FieldAccessFinder.java	6
Type Relation	Inheritance	GenericVisitor INHERIT ASTVisitor	GenericVisitor.java	76
Object Element	Create Object	GridData	EdpManagementPropertyPage.java	105

Dopodiché, come è possibile notare dalla successiva figura l'Edp Detector ha correttamente ordinato per compilation unit.



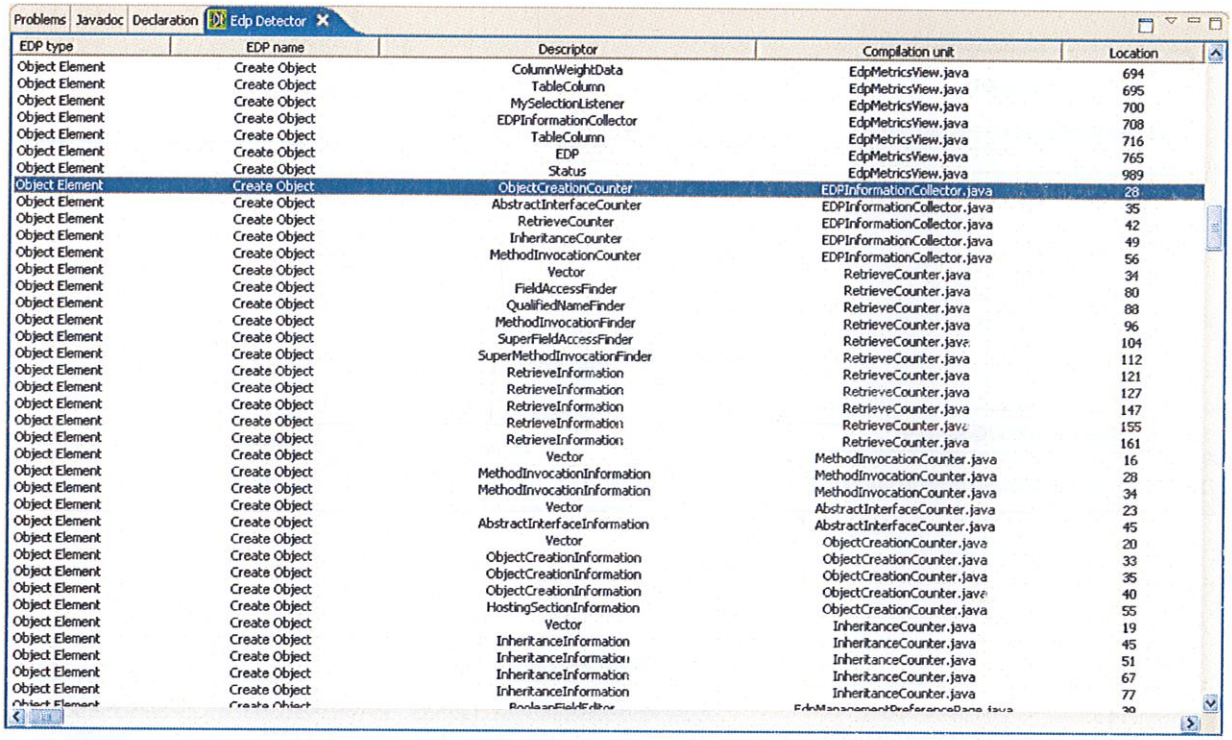
EDP type	EDP name	Descriptor	Compilation unit	Location
Object Element	Retrieve	getPreference - METHOD	AbstractInterfaceCounter.java	39
Object Element	Retrieve	INHERITANCE_REPORT_ABSTRACTCLASSES - FIELD	AbstractInterfaceCounter.java	41
Object Element	Retrieve	getPreference - METHOD	AbstractInterfaceCounter.java	41
Type Relation	Inheritance	AbstractInterfaceInformation INHERIT GenericInformation	AbstractInterfaceInformation.java	3
Type Relation	Inheritance	EDPContentProvider INHERIT EDPListViewer	EDPContentProvider.java	8
Type Relation	Inheritance	EDPContentProvider INHERIT IStructuredContentProvider	EDPContentProvider.java	8
Object Element	Create Object	ObjectCreationCounter	EDPInformationCollector.java	28
Object Element	Create Object	AbstractInterfaceCounter	EDPInformationCollector.java	35
Object Element	Create Object	RetrieveCounter	EDPInformationCollector.java	42
Object Element	Create Object	InheritanceCounter	EDPInformationCollector.java	49
Object Element	Create Object	MethodInvocationCounter	EDPInformationCollector.java	56
Type Relation	Inheritance	EDPLabelProvider INHERIT LabelProvider	EDPLabelProvider.java	7
Type Relation	Inheritance	EDPLabelProvider INHERIT ITableLabelProvider	EDPLabelProvider.java	7
Object Element	Create Object	Vector	EDPList.java	6
Object Element	Create Object	HashSet	EDPList.java	7
Object Element	Retrieve	iterator - METHOD	EDPList.java	13
Object Element	Retrieve	iterator - METHOD	EDPList.java	19
Object Element	Abstract Interface	EDPListViewer	EDPListViewer.java	4
Object Element	Abstract Interface	EDPListViewer	EDPListViewer.java	5
Type Relation	Inheritance	EDPSorter INHERIT ViewerSorter	EDPSorter.java	6
Object Element	Retrieve	getLine - METHOD	EDPSorter.java	32
Object Element	Retrieve	getDefault - METHOD	EditorUtility.java	25
Object Element	Retrieve	getActivePage - METHOD	EditorUtility.java	28
Object Element	Retrieve	getDefault - METHOD	EditorUtility.java	39
Object Element	Retrieve	getActivePage - METHOD	EditorUtility.java	42
Object Element	Retrieve	getEditorInput - METHOD	EditorUtility.java	53
Object Element	Retrieve	getAdapter - METHOD	EditorUtility.java	56
Object Element	Create Object	BooleanFieldEditor	EdpManagementPreferencePage.java	39
Object Element	Create Object	BooleanFieldEditor	EdpManagementPreferencePage.java	43
Object Element	Create Object	BooleanFieldEditor	EdpManagementPreferencePage.java	47
Object Element	Create Object	BooleanFieldEditor	EdpManagementPreferencePage.java	51
Type Relation	Inheritance	EdpManagementPreferencePage INHERIT FieldEditorPr...	EdpManagementPreferencePage.java	8
Type Relation	Inheritance	EdpManagementPreferencePage INHERIT IWorkbenchP...	EdpManagementPreferencePage.java	8
Object Element	Create Object	Label	EdpManagementPropertyPage.java	68
Object Element	Create Object	Text	EdpManagementPropertyPage.java	72
Object Element	Create Object	Label	EdpManagementPropertyPage.java	80
Object Element	Create Object	Text	EdpManagementPropertyPage.java	83
Object Element	Create Object	Label	EdpManagementPropertyPage.java	96

ho deciso anche di mostrare che è possibile ordinare per EDP name



EDP type	EDP name	Descriptor	Compilation unit	Location
Object Element	Retrieve	NULL - FIELD	EdpManagementPropertyPage.java	173
Object Element	Retrieve	NULL - FIELD	EdpManagementPropertyPage.java	173
Type Relation	Inheritance	NameSorter INHERIT ViewerSorter	NameSorter.java	5
Type Relation	Inheritance	NameSorter INHERIT ViewerSorter	EdpMetricsView.java	211
Type Relation	Inheritance	NameSorter INHERIT ViewerSorter	NameSorter.java	5
Type Relation	Inheritance	NameSorter INHERIT ViewerSorter	EdpMetricsView.java	211
Object Element	Create Object	Object	ViewContentProvider.java	85
Object Element	Create Object	Object	ViewContentProvider.java	85
Object Element	Create Object	ObjectCreationCounter	EDPInformationCollector.java	28
Object Element	Create Object	ObjectCreationCounter	EDPInformationCollector.java	28
Type Relation	Inheritance	ObjectCreationCounter INHERIT GenericVisitor	ObjectCreationCounter.java	13
Type Relation	Inheritance	ObjectCreationCounter INHERIT GenericVisitor	ObjectCreationCounter.java	13
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	33
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	35
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	40
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	33
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	35
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	40
Type Relation	Inheritance	ObjectCreationInformation INHERIT GenericInformation	ObjectCreationInformation.java	3
Type Relation	Inheritance	ObjectCreationInformation INHERIT GenericInformation	ObjectCreationInformation.java	3

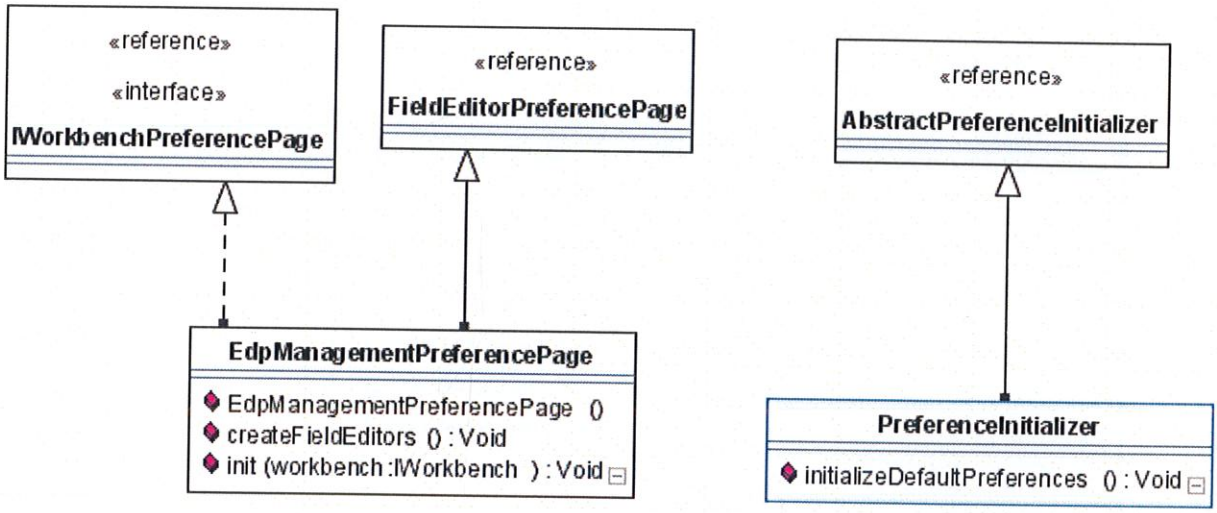
e il corretto ordinamento viene mostrato nella seguente figura

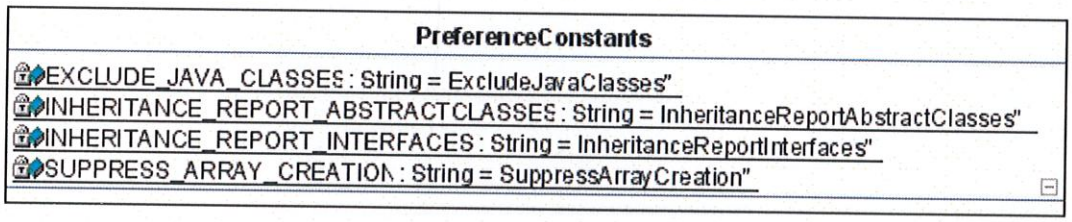


EDP type	EDP name	Descriptor	Compilation unit	Location
Object Element	Create Object	ColumnWeightData	EdpMetricsView.java	694
Object Element	Create Object	TableColumn	EdpMetricsView.java	695
Object Element	Create Object	MySelectionListener	EdpMetricsView.java	700
Object Element	Create Object	EDPInformationCollector	EdpMetricsView.java	708
Object Element	Create Object	TableColumn	EdpMetricsView.java	716
Object Element	Create Object	EDP	EdpMetricsView.java	765
Object Element	Create Object	Status	EdpMetricsView.java	989
Object Element	Create Object	ObjectCreationCounter	EDPInformationCollector.java	28
Object Element	Create Object	AbstractInterfaceCounter	EDPInformationCollector.java	35
Object Element	Create Object	RetrieveCounter	EDPInformationCollector.java	42
Object Element	Create Object	InheritanceCounter	EDPInformationCollector.java	49
Object Element	Create Object	MethodInvocationCounter	EDPInformationCollector.java	56
Object Element	Create Object	Vector	RetrieveCounter.java	34
Object Element	Create Object	FieldAccessFinder	RetrieveCounter.java	80
Object Element	Create Object	QualifiedNameFinder	RetrieveCounter.java	88
Object Element	Create Object	MethodInvocationFinder	RetrieveCounter.java	96
Object Element	Create Object	SuperFieldAccessFinder	RetrieveCounter.java	104
Object Element	Create Object	SuperMethodInvocationFinder	RetrieveCounter.java	112
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java	121
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java	127
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java	147
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java	155
Object Element	Create Object	RetrieveInformation	RetrieveCounter.java	161
Object Element	Create Object	Vector	MethodInvocationCounter.java	16
Object Element	Create Object	MethodInvocationInformation	MethodInvocationCounter.java	28
Object Element	Create Object	MethodInvocationInformation	MethodInvocationCounter.java	34
Object Element	Create Object	Vector	AbstractInterfaceCounter.java	23
Object Element	Create Object	AbstractInterfaceInformation	AbstractInterfaceCounter.java	45
Object Element	Create Object	Vector	ObjectCreationCounter.java	20
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	33
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	35
Object Element	Create Object	ObjectCreationInformation	ObjectCreationCounter.java	40
Object Element	Create Object	HostingSectionInformation	ObjectCreationCounter.java	55
Object Element	Create Object	Vector	InheritanceCounter.java	19
Object Element	Create Object	InheritanceInformation	InheritanceCounter.java	45
Object Element	Create Object	InheritanceInformation	InheritanceCounter.java	51
Object Element	Create Object	InheritanceInformation	InheritanceCounter.java	67
Object Element	Create Object	InheritanceInformation	InheritanceCounter.java	77

• Permettere su una pagina di tipo Preferences (raggiungibile tramite Windows|Preferences|…) di escludere dall'analisi gli EDP relativi alle classi del pacchetto Java. Tale opzione sarà possibile, una sola volta, con semantiche diverse per tutti i 4 EDP. Ad esempio se escludo il processing delle class Java nel caso del Create Object non registrerò la new di un Vettore, per Abstract Interface non cambia la modalità di processing, mentre per Retrieve e Inheritance, sì. Ovviamente cambiando l'opzione le informazioni verranno ricalcolate.
Sulle Preferences: relativamente a Create Object si potrà scegliere se visualizzare o meno la creazione degli Array (nodi di tipo Array Creation)
Sulle preferences, relativamente a Abstract Interface, si potrà scegliere cosa includere nell'analisi:
1. Interfacce
2. Classi Astratte

La pagina preference è stata creata tramite le estensioni org.eclipse.ui.proferencePages e org.eclipse.core.runtime.preferences. Tre classi concorrono a fornire le opzioni:
- PreferenceConstants, che fornisce alcune costanti usate per identificare le opzioni;
- PreferenceInitializer, che viene usato per generare i valori di default delle opzioni (estensione preferences);
- EdpManagementPreferencePage, che realizza la pagina di preference vera e propria, creando i campi usati per modificare le opzioni (estensione preferencePages).
Detto ciò, ora mostro nella figura seguente il diagramma delle classi appartenenti al package preferences

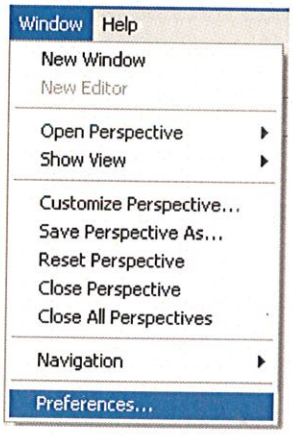




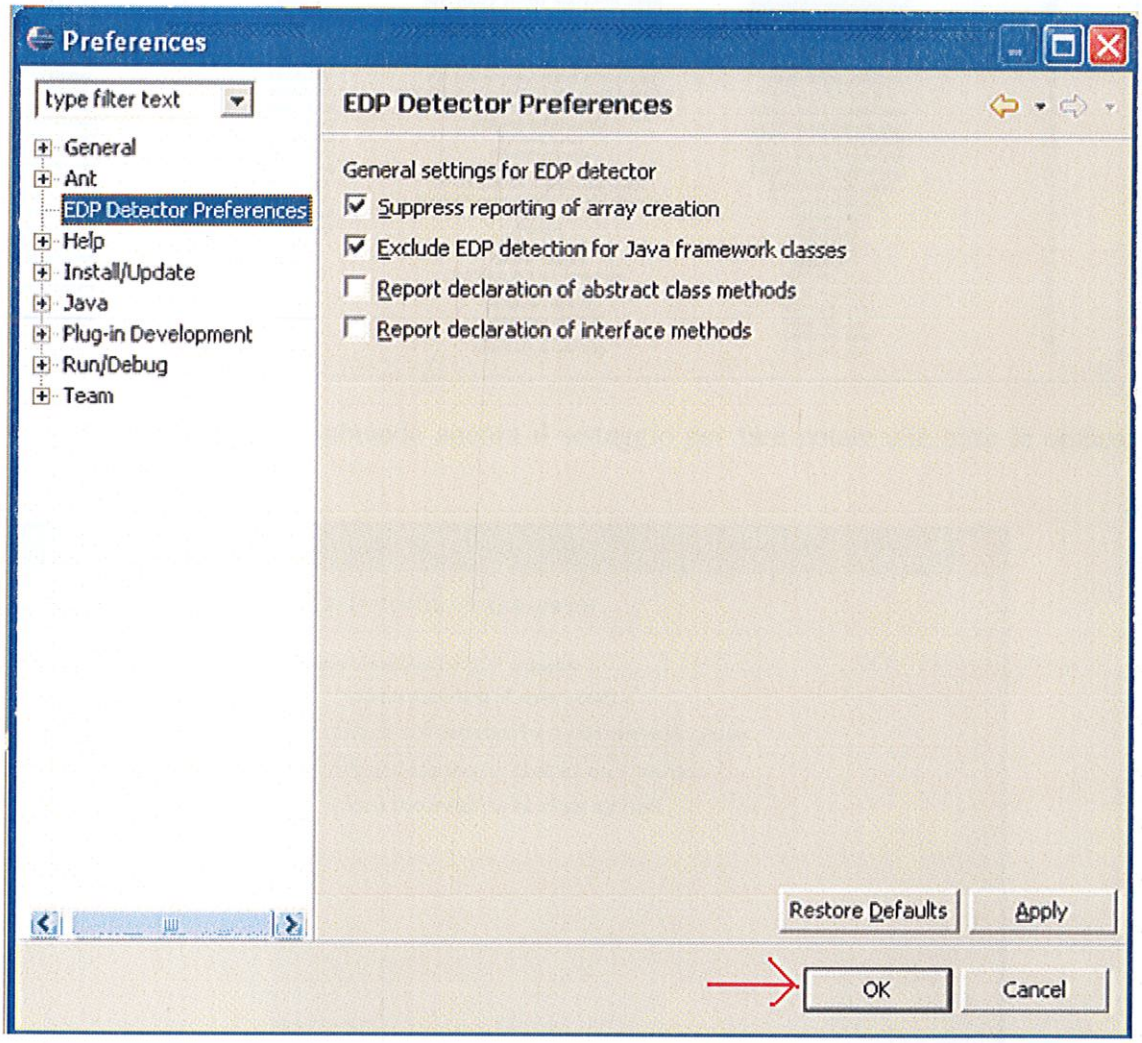
Per quanto riguarda l'uso delle property e delle preference nel codice l'unica property JLS_LEVEL viene usata per determinare a quale livello di JLS il parser generare l'albero sintattico.
L'opzione di soppressione degli EDP riguardanti la creazione degli array viene usata nel visitor ObjectCreationCounter, responsabile di segnalare gli EDP relativi a creazione di oggetti. Le opzioni di selezione degli EDP riguardanti la dichiarazione di metodi astratti viene usata nel visitor AbstractInterfaceCounter.

Adesso vado a trattare cosa succede in fase di esecuzione.

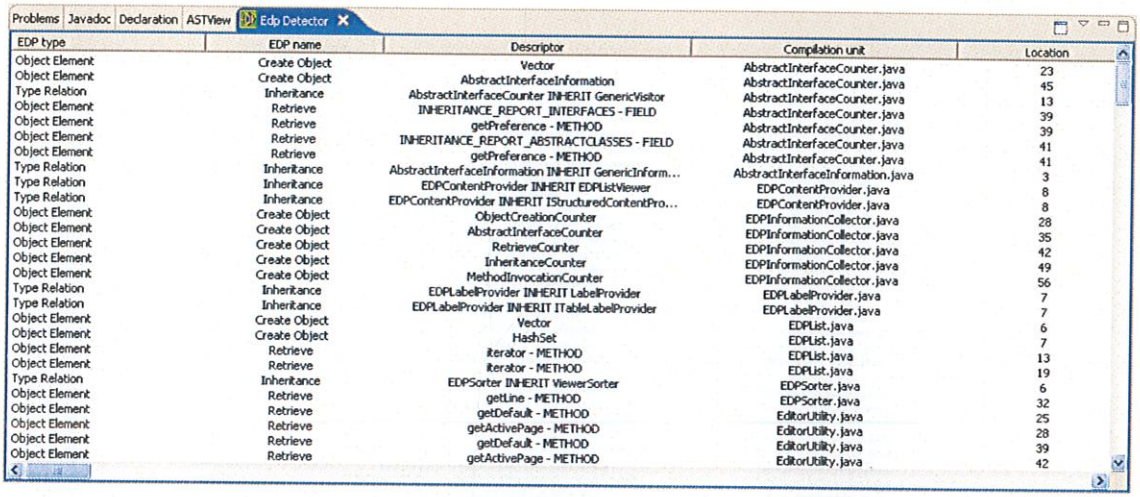
1)Clicco su Windows e poi su Preferences



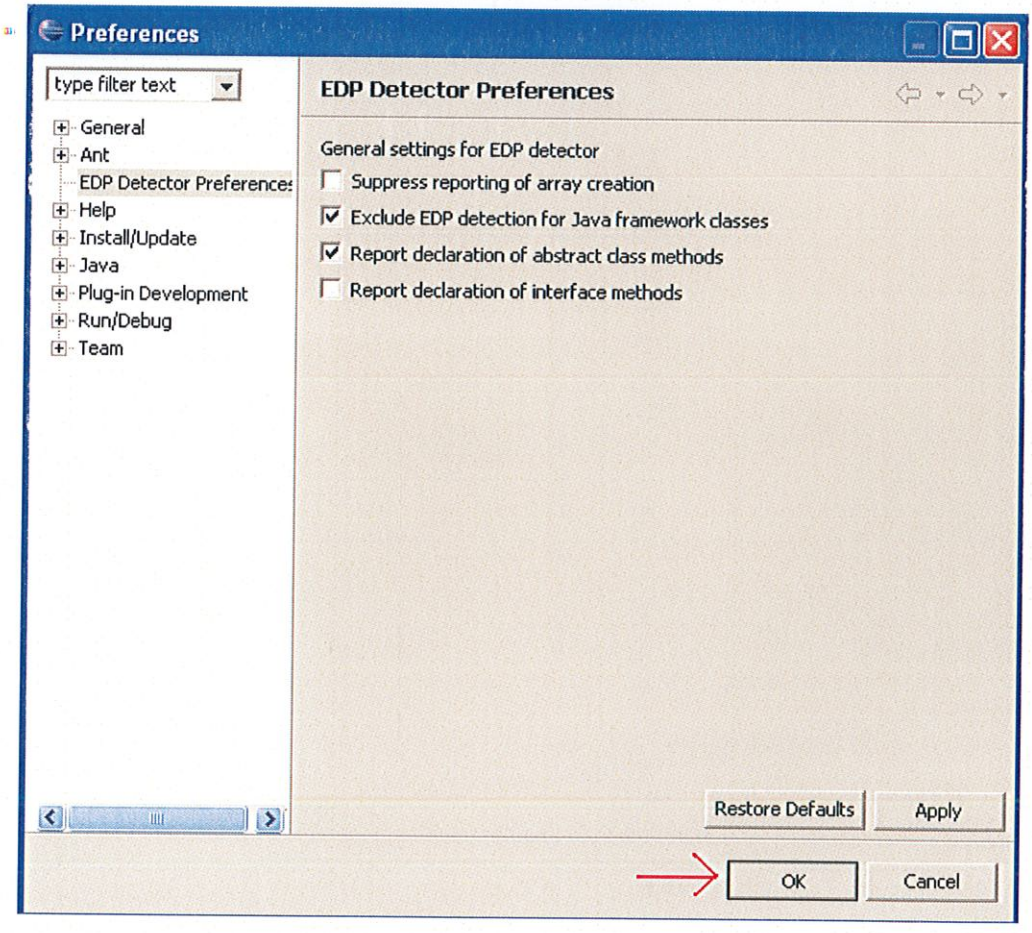
2)ottengo la pagina di preferences, seleziono le mie due preferenze e clicco su ok



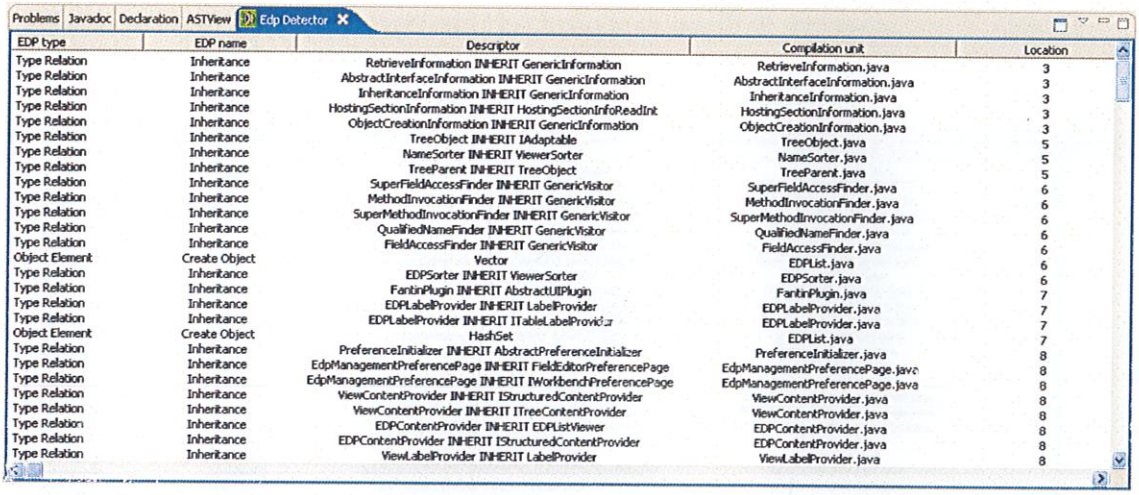
3) faccio partire l'Edp Detector e controllo il risultato il quale dovrà essere identico a quello ottenuto con le preferences di default, con la differenza che i pattern di creazione di array non saranno più riportati.



4)ripeto l'operazione cambiando ancora il setteggio per fare notare che tutte le opzioni funzionano

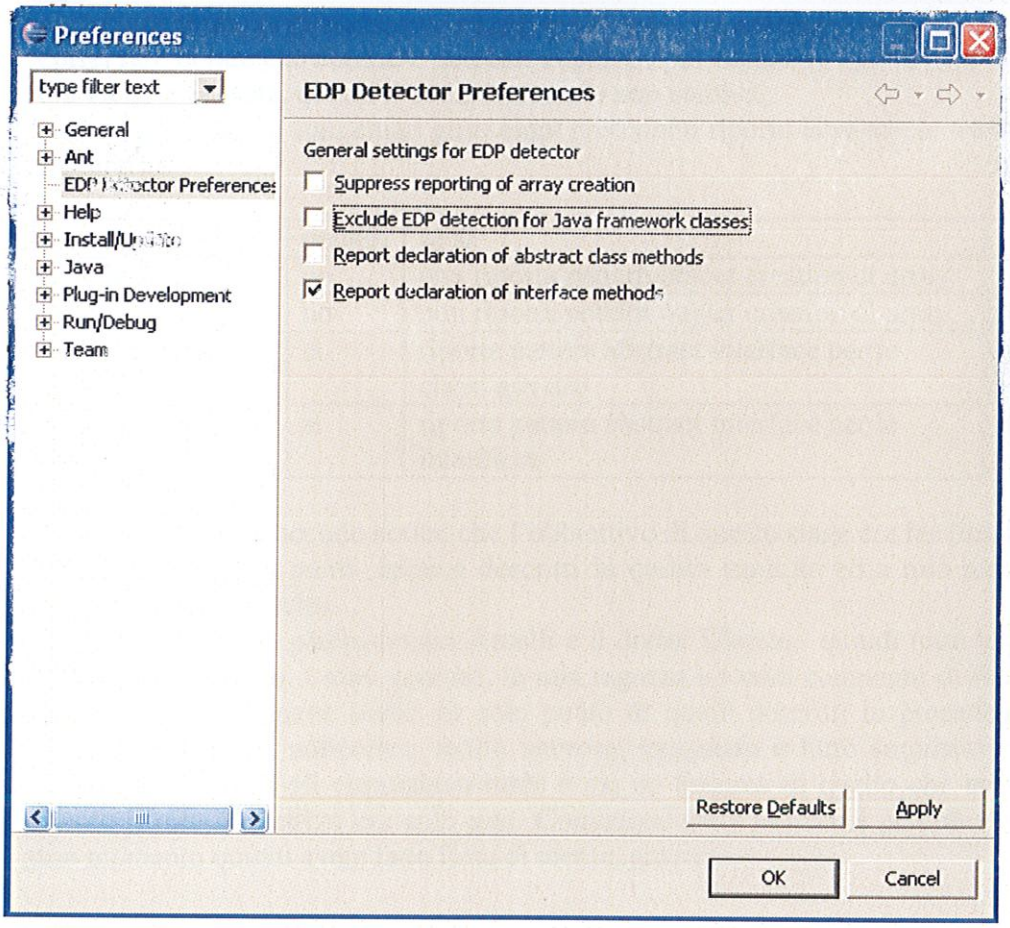


5)faccio partire l'Edp Detector e controllo se è tutto giusto, , in quanto mi aspetto che vengano esclusi i pattern relativi alle classi standard di Java, e che per quanto riguarda i pattern di dichiarazione di metodo astratto vengano riportati quelli dichiarati nelle classi astratte.

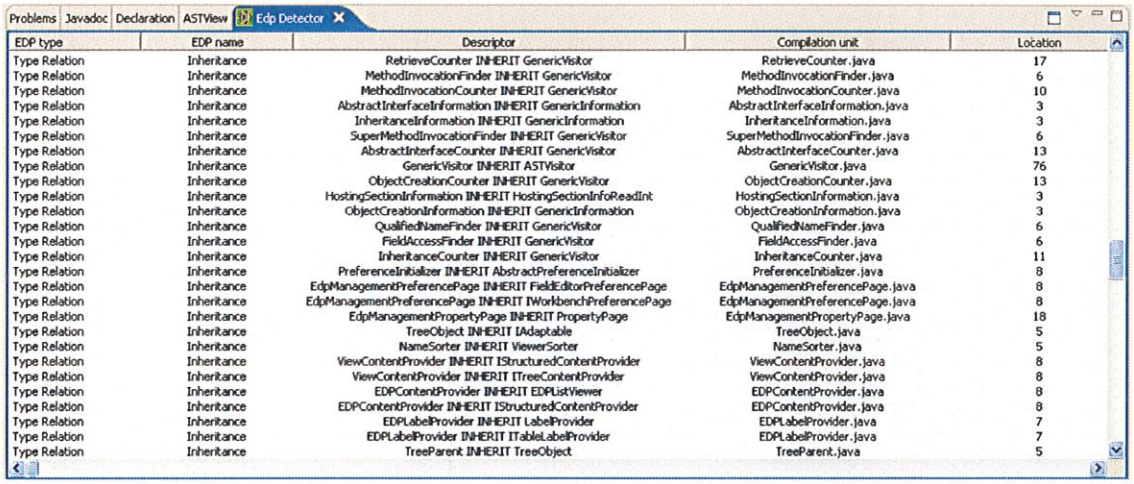


EDP type	EDP name	Descriptor	Compilation unit	Location
Type Relation	Inheritance	RetrieveInformation INHERIT GenericInformation	RetrieveInformation.java	3
Type Relation	Inheritance	AbstractInterfaceInformation INHERIT GenericInformation	AbstractInterfaceInformation.java	3
Type Relation	Inheritance	InheritanceInformation INHERIT GenericInformation	InheritanceInformation.java	3
Type Relation	Inheritance	HostingSectionInformation INHERIT HostingSectionInfoReadInt	HostingSectionInformation.java	3
Type Relation	Inheritance	ObjectCreationInformation INHERIT GenericInformation	ObjectCreationInformation.java	3
Type Relation	Inheritance	TreeObject INHERIT IAdaptable	TreeObject.java	5
Type Relation	Inheritance	NameSorter INHERIT ViewerSorter	NameSorter.java	5
Type Relation	Inheritance	TreeParent INHERIT TreeObject	TreeParent.java	5
Type Relation	Inheritance	SuperFieldAccessFinder INHERIT GenericVisitor	SuperFieldAccessFinder.java	6
Type Relation	Inheritance	MethodInvocationFinder INHERIT GenericVisitor	MethodInvocationFinder.java	6
Type Relation	Inheritance	SuperMethodInvocationFinder INHERIT GenericVisitor	SuperMethodInvocationFinder.java	6
Type Relation	Inheritance	QualifiedNameFinder INHERIT GenericVisitor	QualifiedNameFinder.java	6
Type Relation	Inheritance	FieldAccessFinder INHERIT GenericVisitor	FieldAccessFinder.java	6
Object Element	Create Object	Vector	EDPList.java	6
Type Relation	Inheritance	EDPSorter INHERIT ViewerSorter	EDPSorter.java	6
Type Relation	Inheritance	FantinPlugin INHERIT AbstractUIPlugin	FantinPlugin.java	7
Type Relation	Inheritance	EDPLabelProvider INHERIT LabelProvider	EDPLabelProvider.java	7
Type Relation	Inheritance	EDPLabelProvider INHERIT ITableLabelProvider	EDPLabelProvider.java	7
Object Element	Create Object	HashSet	EDPList.java	7
Type Relation	Inheritance	PreferenceInitializer INHERIT AbstractPreferenceInitializer	PreferenceInitializer.java	8
Type Relation	Inheritance	EdpManagementPreferencePage INHERIT FieldEditorPreferencePage	EdpManagementPreferencePage.java	8
Type Relation	Inheritance	EdpManagementPreferencePage INHERIT IWorkbenchPreferencePage	EdpManagementPreferencePage.java	8
Type Relation	Inheritance	ViewContentProvider INHERIT IStructuredContentProvider	ViewContentProvider.java	8
Type Relation	Inheritance	ViewContentProvider INHERIT ITreeContentProvider	ViewContentProvider.java	8
Type Relation	Inheritance	EDPContentProvider INHERIT EDPListViewer	EDPContentProvider.java	8
Type Relation	Inheritance	EDPContentProvider INHERIT IStructuredContentProvider	EDPContentProvider.java	8
Type Relation	Inheritance	ViewLabelProvider INHERIT LabelProvider	ViewLabelProvider.java	8

6) cambio ancora le preferenze



7) faccio ricalcolare ancora le operazioni all'Edp Detector e constato che vengono visualizzati tutti i pattern trovati con l'esclusione di quelli di dichiarazione di metodo astratto in una classe astratta come mi aspettavo.



Problems | Javadoc | Declaration | ASTView | Edp Detector

EDP type	EDP name	Descriptor	Compilation unit	Location
Type Relation	Inheritance	RetrieveCounter INHERIT GenericVisitor	RetrieveCounter.java	17
Type Relation	Inheritance	MethodInvocationFinder INHERIT GenericVisitor	MethodInvocationFinder.java	6
Type Relation	Inheritance	MethodInvocationCounter INHERIT GenericVisitor	MethodInvocationCounter.java	10
Type Relation	Inheritance	AbstractInterfaceInformation INHERIT GenericInformation	AbstractInterfaceInformation.java	3
Type Relation	Inheritance	InheritanceInformation INHERIT GenericInformation	InheritanceInformation.java	3
Type Relation	Inheritance	SuperMethodInvocationFinder INHERIT GenericVisitor	SuperMethodInvocationFinder.java	6
Type Relation	Inheritance	AbstractInterfaceCounter INHERIT GenericVisitor	AbstractInterfaceCounter.java	13
Type Relation	Inheritance	GenericVisitor INHERIT ASTVisitor	GenericVisitor.java	76
Type Relation	Inheritance	ObjectCreationCounter INHERIT GenericVisitor	ObjectCreationCounter.java	13
Type Relation	Inheritance	HostingSectionInformation INHERIT HostingSectionInfoReadInt	HostingSectionInformation.java	3
Type Relation	Inheritance	ObjectCreationInformation INHERIT GenericInformation	ObjectCreationInformation.java	3
Type Relation	Inheritance	QualifiedNameFinder INHERIT GenericVisitor	QualifiedNameFinder.java	6
Type Relation	Inheritance	FieldAccessFinder INHERIT GenericVisitor	FieldAccessFinder.java	6
Type Relation	Inheritance	InheritanceCounter INHERIT GenericVisitor	InheritanceCounter.java	11
Type Relation	Inheritance	PreferenceInitializer INHERIT AbstractPreferenceInitializer	PreferenceInitializer.java	8
Type Relation	Inheritance	EdpManagementPreferencePage INHERIT FieldEditorPreferencePage	EdpManagementPreferencePage.java	8
Type Relation	Inheritance	EdpManagementPreferencePage INHERIT IWorkbenchPreferencePage	EdpManagementPreferencePage.java	8
Type Relation	Inheritance	EdpManagementPropertyPage INHERIT PropertyPage	EdpManagementPropertyPage.java	18
Type Relation	Inheritance	TreeObject INHERIT IAdaptable	TreeObject.java	5
Type Relation	Inheritance	NameSorter INHERIT ViewerSorter	NameSorter.java	5
Type Relation	Inheritance	ViewContentProvider INHERIT IStructuredContentProvider	ViewContentProvider.java	8
Type Relation	Inheritance	ViewContentProvider INHERIT ITreeContentProvider	ViewContentProvider.java	8
Type Relation	Inheritance	EDPContentProvider INHERIT EDPListViewer	EDPContentProvider.java	8
Type Relation	Inheritance	EDPContentProvider INHERIT IStructuredContentProvider	EDPContentProvider.java	8
Type Relation	Inheritance	EDPLabelProvider INHERIT LabelProvider	EDPLabelProvider.java	7
Type Relation	Inheritance	EDPLabelProvider INHERIT ITableLabelProvider	EDPLabelProvider.java	7
Type Relation	Inheritance	TreeParent INHERIT TreeObject	TreeParent.java	5

Vorrei fare notare che nell'EDP detector l'opzione per escludere i pattern delle classi java non viene mai usata, quindi il comportamento non cambia.
Qualora non fossero stati chiari gli esempi precedenti riporto la seguente tabella riassuntiva:

opzione	default	se sì	se no
exclude array	no	non riporta pattern object creation di array	li riporta
exclude java	no	non riporta pattern object creation di array	li riporta
report abstract class	sì	riporta pattern abstract interface per le classi astratte	non li riporta
report interface	sì	riporta pattern abstract interface per le interfacce	non li riporta

Vorrei concludere facendo notare che l'obbiettivo di questo stage era far funzionare l'Edp Detector con tutti i punti appena descritti in questo capitolo ed a mio modesto parere penso di esserci riuscito.
Ringrazio ancora la professoressa Arcelli e il dottor Ubezio, quindi tutte le persone che mi sono state vicino, i miei genitori, la mia ragazza e i miei compagni di facoltà perché quando non funzionava anche un solo punto di quelli descritti in precedenza, ero una persona scorbutica, antipatica, molto nervosa, irrequieta e tutto sommato ora mi sono accorto che magari gli rispondevo male e me ne fregavo di quello che mi dicevano in quanto il mio obbiettivo era solo uno. Comunque siete fantastici perché avete capito il mio momento quindi avete fatto finta di niente, grazie.

Stefano Fantin Barbaiana di Lainate 16/1/2006