ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

Факултет по Компютърни Системи и Управление
Катедра “Компютърни системи”

Д И П Л О М Н А Р А Б О Т А

Тема: “Програма за автоматичен анализ на софтуерни метрики”

Дипломант :
Богдан Иванов Вътков
Фак. No : РК 028414

Научен ръководител:
Доц. д-р Стоян Бонев

София 2005

Съдържание

Увод

Качеството на програмния код съставящ една програмна система е основен фактор за оптималното протичане на процесите на разработка, тестване и поддръжка. Наличието на лесен способ за измерване и оценка на качеството на програмния код би спестило значителни усилия на цели екипи от разработчици, техници по тестуването и техници по поддръжката на софтуерни системи. Съществуват редица системи от формализми дефиниращи методи за измерване и оценка качеството на програмния код (софтуерни метрики). Софтуерните метрики адресират различни показатели на качеството на програмния код: обем на кода, сложност на алгоритмите, степен на поддържаемост, качество на Обектно-Ориентиран Дизайн, и т.н.

Достъпът до показателите за качество във всеки един етап от разработката на програмните системи може да се окаже критично важен, тъй-като показателите за качество са основният източник на информация за пресмятане на риска за определена програмна единица: проект, пакет, клас дори методи. Метриките за качество са показател, който може да даде отношението на цената на поддръжка и цената за пренаписване на даден програмен код.

Този дипломен проект има за цел да изгради програмна система за автоматичен анализ на софтуерни метрики. Анализът се осъществява над първичен или обектен програмен код а резултатите от анализа – софтуерните метрики – се съхраняват в популярен формат, което трябва да способства за интегрирането на текущата система с други системи, които могат да оползотворят извлечените метрики. Основната идея при реализацията на проекта е да се ползват максимален брой готови за употреба библиотеки със отворен код. Това повишава всеобщото качеството на разработката, тъй-като активните проекти с отворен код най-често имат високи показатели за стабилност и бързодействие. Поради сходни разсъждения проектът на дипломната работа е публикуван като проект с отворен код, спазвайки Lesser General Public License (LGPL) лиценз.

1 Функционално описание на програмата. Пакети от софтуерни метрики.

1.1 Цел. Изисквания.

Основната цел на проекта за анализ на софтуерни метрики е да се създаде единна разширяема система за автоматичен анализ на програмен код и извеждане на широк спектър от софтуерни метрики. В понятието *програмен код* се включва както първичен програмен код, така и двоичен (обектен) програмен код.

Основните изисквания към програмата са формулирани както следва:

1. Необходимост и достатъчност:

Програмният код е единствения източник на данни за процеса на анализ на софтуерни метрики. Програмата трябва да прави разграничение от програмен код подлежащ на анализ на софтуерни метрики и такъв – не подлежащ на анализ на софтуерни метрики, но необходим за правилното протичане на анализи предхождащи анализа на софтуерни метрики; например: семантичен анализ установяващ типовете на аргументи на повикани методи. Двата типа програмен код за краткост ще наричаме "ресурси за анализ" и "допълнителни ресурси".

Програмата трябва да дава възможност на потребителя да избира множества от програмен код с точност до файл от локална файлова система.

2. Информативност:

Резултатът от анализа на програмния код трябва да бъде достатъчно информативен за да може потребителя да вземе решение за нуждата от преработване на програмния код. За всяка метрика трябва да бъдат предоставени сравнителни стойности, чрез които да бъде извършвана автоматична класификация на анализирания програмен код. Пример: минимално допустими стойности, максимално допустими стойности, препоръчителни стойности, и др.

3. Удобство на употреба:

Програмата за анализ трябва да бъде лесна и удобна за употреба от страна на потребители със минимална компютърна грамотност. Програмата трябва да предоставя индикация на прогреса на анализа в реално време.

4. Отчетност:

Програмата трябва да предоставя резултата от анализ под формата на файл с популярен формат (XML, CSV, HTML) и минимален обем.

5. Разширяемост:

Различаваме два основни аспекта на разширяемостта.

5.1.Технически аспект:

Реализацията на програмата трябва да използва стандартни подходи за разширение на функционалността. Разширяемост се изисква в две основни направления: добавяне на поддръжка за програмни езици, чрез които е създаден програмния код, който се подлага на анализ и добавяне на пакети от софтуерни метрики.

5.2.Социален аспект:

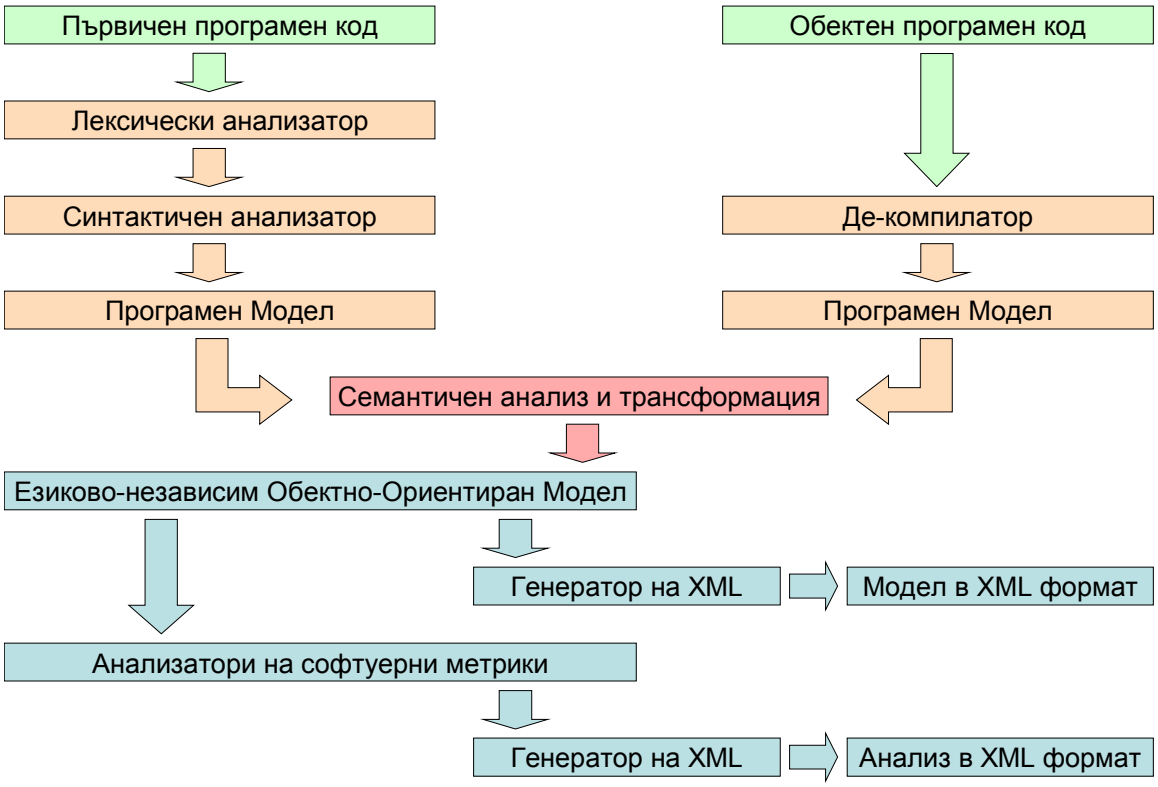
Програмата трябва да бъде публикувана с безплатен достъп за разработчици, които имат желание и възможности да разширяват функционалността.

1.2 Понятието "софтуерна метрика".

Понятието "софтуерна метрика" покрива обширен спектър от метрики свързани с планирането, проектирането, производството, тестуването и поддръжката на софтуер. В текущото изложение под "софтуерна метрика" или само "метрика" ще се разбира такава метрика, която може да бъде автоматично ("от автомат") изчислена след прочитане на даден програмен код. Съществува огромен набор изучени, формализирани и систематизирани софтуерни метрики. В текущата разработка са включени повечето от най-широко разпространените софтуерни метрики [Halstead 77], [McCabe 94], [Welker 95], [Martin 94], [Chidamber 91], [Henderson 96], [Briand 98].

1.3 Обобщена схема на функционалността.

Обобщената схема на функционалността на програмата (фиг.1.3) описва в груб план архитектурата на приложението и основните процеси необходими за изпълнението на задачата.



фиг. 1.3

1.4 Пакети от софтуерни метрики.

Всяка система от софтуерни метрики е обособена като "пакет" от софтуерни метрики, като най-често името на пакета съвпада с името на автора на формализма на съответната метрика.

1.4.1 Метрики за сложност на Halstead.

Метриките за сложност на Halstead са разработени с цел да се пресметне сложността на програмен модул директно от използвайки първичният програмен код. Фокусът тук е върху изислителната сложност на програмата. Метриките са разработени от Maurice Halstead като средство за определяне на количествен показател на сложността като функция на употребените оператори и операнди в програмата [Halstead 77]. Сред най-ранните софтуерни метрики, тези метрики са силни индикатори за сложността на кода. Тъй като тези метрики се прилагат върху

съществуващ код, те най често се използват като метрики за поддръжка. Има различни мнения относно ефективността на метриките на Halstead, започвайки от "сложни и ненадеждни" [Jones 94] до "сред най-добрите метрики за поддържаемост на кода" [Oman 91]. Едно е сигурно. Метриките на Halstead носят повече информация от колкото най-простата метрика: брой редове.

Метриките на Halstead се базират на четири скаларни величини извлечени директно първичния програмен код:

n1 = броят на уникалните оператори

n2 = броят на уникалните операнди

N1 = общият брой на операторите

N2 = общият брой на операндите

1.4.1.1 Размер на речника.

Размер на Речника (Program Vocabulary)се бележи с n и се изразява чрез формулата:

$$\mathbf{n = n1 + n2}$$

1.4.1.2 Размер на реализацията.

Размерът на реализацията (Implementation Length) се бележи с N и се пресмята с формулата:

$$\mathbf{N = N1 + N2}$$

1.4.1.3 Израз на дължината.

Изразът на дължината (Length Equation) се бележи с N' и се изразява с формулата:

$$\mathbf{N' = n1*log2n1 + n2*log2n2}$$

Експериментално е установено че изразът на дължината често доближава размерът на реализацията. Изразът на дължината дава връзка между размера на речника и размера на имплементацията.

1.4.1.4 Обем на програмата.

Обемът на Програмата (Program Volume) се бележи с V и се изразява посредством формулата:

$$\mathbf{V = N*log2n}$$

1.4.1.5 Време за съставяне.

Времето за съставяне (Time Equation) се бележи с T, измерва се в

секунди и се изразява с формулата:

$$T = (n1*N2*(n1*\log2n1 + n2*\log2n2)*\log2n) / 2*n2*S$$

Психологът John Stroud в "The Psychological Moment in Perception" дефинира понятието "психологически момент" като интервала от време, необходим на човешкия мозък да извърши най-примитивното (атомарно) умозаключение. Тук константата S е броят на моменти на Stroud за секунда, като 5 <= S <= 20. В текущата реализация е използвана константа 10.

1.4.1.6 Сложност на програмата.

Сложността на програмата (Program Difficulty) се бележи с D и се изразява чрез формулата:

$$D = (n1/2)*(N2/n2)$$

1.4.1.7 Усилие за съставяне.

Усилието за съставяне (Program Effort) се бележи с E и се изразява чрез формулата:

$$E = D*V$$

1.4.1.8 Ниво на програмата.

Нивото на програмата (Program Level) се бележи с L и се пресмята с:

$$L = 1/D$$

1.4.1.9 Интелектно съдържание.

Интелектното съдържание (Intelligence Content) се бележи с I и се пресмята чрез:

$$I = (2*n2 / n1*N2) * (N1 + N2)*\log2(n1 + n2)$$

1. Предимства на подхода на Halstead :

- Не изисква анализ в дълбочина на структурите на програмите.
- Предсказва вероятността от възникване на грешки.
- Предсказва усилията необходими за поддръжка.
- Полезен при планиране и отчитане прогреса на проекти.
- Измерва общото качество на програмите.
- Изисква прости изчисления.
- Може да бъде използван за всеки програмен език.
- Множество индустриално значими анализатори поддържат употребата Halstead в предсказването на усилията необходими за съставяне на програми и посредствен брой на грешките при програмиране.

2. Недостатъци на метода на Halstead:

Изисква завършен (компилируем) код.

1.4.2 Метрики на McCabe.

Този пакет от метрики съответства на системата от метрики на McCabe. Включват се метриките: сложност на преходите (), плътност на преходите

Сложността на преходите (Cyclomatic Complexity) на McCabe, най-интересната метрика предложена от McCabe, е най-широко използваната софтуерна метрика. Предложена е от Thomas McCabe през 1976. Тя измерва броя на линейно-независимите пътища в програмата. Метриката сложност на преходите често е наричана просто "сложност на програмата" или "сложност на McCabe". Често се използва в комбинация с други софтуерни метрики. Сложността на преходите е метрика инвариантна по отношение на програмния език [McCabe 94].

1.4.2.1 Сложност на преходите.

Концепцията на метриката сложност на преходите е подобна на метриката за сложност на текст измерена, чрез теста Flesch-Kincaid Readability Test. Тестът е функция на средния брой срички и средната дължина на изреченията.

Сложността на преходите се пресмята чрез граф изразяващ преходите в една програма. Възлите на графа съответстват на командите в програмата. Насочена дъга свързва два възела ако командата съответстваща на втория възел може да бъде изпълнена веднага след командата съответстваща на първия възел. Бележи се с CC и се изразява с формулата:

$$\mathbf{CC = E - N + P}$$

E – броя на дъгите в графа

N - броя на възлите в графа

P – броя на съединените възли

Сложността на преходите може просто да бъде дефинирана като броя на разклоненията в процедура, метод, програма или въобще система. Правилото на McCabe гласи, че стойности за сложността на преходите по-големи от 10 сочат предразположен към грешки код. Switch/Case структури са изключение тъй като те могат лесно да изразяват повече от 10 разклонения без това да се отразява на вероятността за увеличаване на грешките.

За да може една програма да бъде тествана напълно всички пътища на изпълнение трябва да бъдат проиграни. Това обяснява защо програма с голяма сложност на преходите изисква по големи усилия за тестване от колкото програма с по малка сложност на преходите, тъй като сложността на преходите индикира пътищата в изпълнението на кода. Големите стойности на сложността на преходите също е признак за четливостта на кода, тъй като програмиста трябва да разбере

различните пътища на изпълнение и резултатите от тези изпълнения .

Все пак съществуват типове програми, от които нормално се очаква да притежават големи стойности на сложността на преходите. Примери: различни валидации, машинни автомати, и др.

1.4.2.2 Плътност на преходите.

Плътността на преходите (Decision Density), се бележи с DD и се изразява с формулата:

$$\mathbf{DD = CC / LOC}$$

LOC (Lines Of Code) – брой редове на кода.

1. Предимства на сложността на преходите на McCabe:

- Може да се използва като добавка на метрики за поддържаемостта.
- Използван като метрика за качеството показва сравнителни сложности за различни архитектури на програмите.
- Може да бъде пресметната в по ранни фази от жизнения цикъл от колкото метриките на Halstead.
- Измерва областите изискващи минимални и максимални усилия за тестване.
- Направлява тестването, чрез ограничения на програмната логика по време на разработка.
- Лесна за изчисление.

2. Недостатъци на сложността на преходите на McCabe:

- Сложността на преходите е метрика на алгоритмите но не и на данните използвани в тях.
- С еднакво тегло се броят вложените и не-вложените условни преходи. Дълбоко вложените условни структури са по трудни за разбиране от не-вложените такива.
- Може да доведе до заблуждаващи стойности в следствие на множество прости сравнения и машини на състоянието.

1.4.3 Обобщаващи метрики.

В този пакет са включени метрики, които не са причислени към една конкретна система от софтуерни метрики. Включва брой редове, брой класове и индекс на поддържаемостта.

1.4.3.1 Индекс на поддържаемостта.

Дефиниция за поддържаемост:

Лекотата, с която една програмна система може да бъде модифицирана с цел корекция на грешки, подобряване на производителността или други параметри, или адаптирана към промени в обкръжението [IEEE 90].

Индексът на поддържаемостта (Maintainability Index) като техника за измерване на поддържаемостта на програма [Welker 95] е резултат от многогодишен съвместен труд на различни организации като: Software Engineering Test Laboratory of the University of Idaho, the Idaho National Engineering Laboratory, Hewlett-Packard, и др.

Усилията свързани с измерването на поддържаемостта се оправдават от възможността да бъде намалена вероятността от нарастване на ентропията на кода или понижаване на цялостта на програмата. Индексът на поддържаемост служи за индикация на това дали би било по-евтино (или с по малък риск) пренаписването на кода от колкото промяната с цел корекция или адаптация.

Индексът на поддържаемост за една програма се бележи с IM и се изразява с формулата:

MI = 171 - 5.2 * ln(aV) - 0.23 * aV(g') - 16.2 * ln (aLOC) + 50 * sin (sqrt(2.4 * pCM))

aV – средна стойност на Обема на програмата на Halstead за програмата

aV(g') – среда стойност на Сложността на преходите на McCabe за програмата

aLOC – средна стойност на Брой на редовете за програмата

pCM – среден стойност на процента на редовете коментари за програмата. Това е опционален компонент на полинома.

Коефициентите в уравнението са получени след калибриране на базата на анализи на редица програмни системи поддържани от Hewlett-Packard. Детайлно описание на калибровката на уравнението може да се намери в [Coleman 94, Coleman, 95, Pearse 95, Welker 95]. Авторите твърдят, че последвали опити показват че посочената форма на уравнението е приложима и при други индустриални програмни системи. [Oman 94 and Welker 95]. Препоръчва се коефициентите да бъдат тествани за всяка по-голяма програмна система, на която се прилага индексът на поддържаемост.

Въпреки известната доза субективност при калибрация и дефиниране на гранични и препоръчителни стойности, си позволих да поставя такива в следствие подлагането на проекта на измерване с индекс на поддържаемостта.

За стойности на индекса по-малки от 50 се извежда предупреждение за относително ниска степен на поддържаемост.

За стойности на индекса по-малки от 30 се извежда съобщение за критично ниска степен на поддържаемост на програмата.

1.4.3.2 Брой на редове.

Броят на редове (Lines Of Code) е най-примитивната софтуерна метрика.

1.4.3.3 Брой Класове.

Броят класове е (Class Count) е броят на класовете в пакет (или пространство от имена). Броят на класове най-често е равен на броя на файлове но не винаги, тъй като съществуват програмни езици, при които в един клас може да има други "вътрешни" класове.

1.4.4 Метрики за качеството на Обектно-Ориентирания Дизайн. Анализ на зависимостите.

Този пакет от софтуерни метрики съответства на системата от метрики на Robert Martin, наречена "Software Package Metrics" [Martin 02]. Включват се метриките: брой конкретни класове, брой абстрактни класове (интерфейси), брой зависими елементи, брой влияещи елементи, абстрактност, нестабилност, разстояние от основната последователност. Това е система от обектно-ориентирани софтуерни метрики.

1.4.4.1 Брой конкретни класове и брой абстрактни класове

Броят на конкретните и броят на абстрактните класове (интерфейси) в един програмен пакет са индикатор за разширяемостта на пакета. Бележат се съответно с Acc и Ccc. Общият брой на класовете в пакета се бележи с Tcc.

1.4.4.2 Брой зависими пакети.

Броят зависими от даден програмен пакет други пакети (Afferent Couplings) е показател за отговорността на пакета. Бележи се с Ca.

1.4.4.3 Брой влияещи пакети.

Броят влияещи на даден програмен пакет други пакети (Efferent Couplings) е показател за независимостта на пакета. Бележи се с Ce.

1.4.4.4 Абстрактност.

Абстрактността на даден пакет е отношението на абстрактните класове към общия брой на класовете в пакета.

$$\mathbf{A = Acc/Tcc}$$

Допустими стойности са от 0 до 1, като A=0 означава напълно конкретен пакет, а A=1 означава напълно абстрактен пакет.

1.4.4.5 Нестабилност

Нестабилността (Instability) се определя като отношението на броя на

зависимите пакети на даден пакет от общия брой на зависимостите на пакета.

$$\mathbf{I = Ce / (Ce + Ca)}$$

Тази метрика е показател за гъвкавостта при промени на пакета. Диапазонът от стойности логично е от 0 до 1, като I=0 показва напълно стабилен пакет, а I=1 показва напълно нестабилен пакет.

1.4.4.6 Разстояние от основната последователност. Нормализирано разстояние.

Когато говорим за отношението абстрактност – стабилност две "идеални" категории могат да бъдат посочени: максимално абстрактна и стабилна, и максимално конкретна и нестабилна. Първата категория се определя като "идеална" поради факта, че разпространява пълна стабилност при наследяване, а такова трябва да има, тъй като класовете при тази категория са напълно абстрактни; логично е че те ще бъдат наследявани. Втората категория е "идеална" поради факта, че нестабилността, която притежава пакета няма да бъде разпространена, тъй като класовете в пакета са напълно конкретни.

Ако дефинираме двумерно пространство с измерения абстрактност и нестабилност и свържем двете най-благоприятни точки [I=0, A=1] и [I=1, A=0] ще получим линия на основната последователност (Main Sequence) (фиг. 1.4.4.6). Всички пакети лежащи на линията притежават баланс между абстрактност и нестабилност. Или казано по друг начин, пакети лежащи на линията на основната последователност не са прекалено абстрактни за стабилността си нито са прекалено нестабилни за абстрактността си.

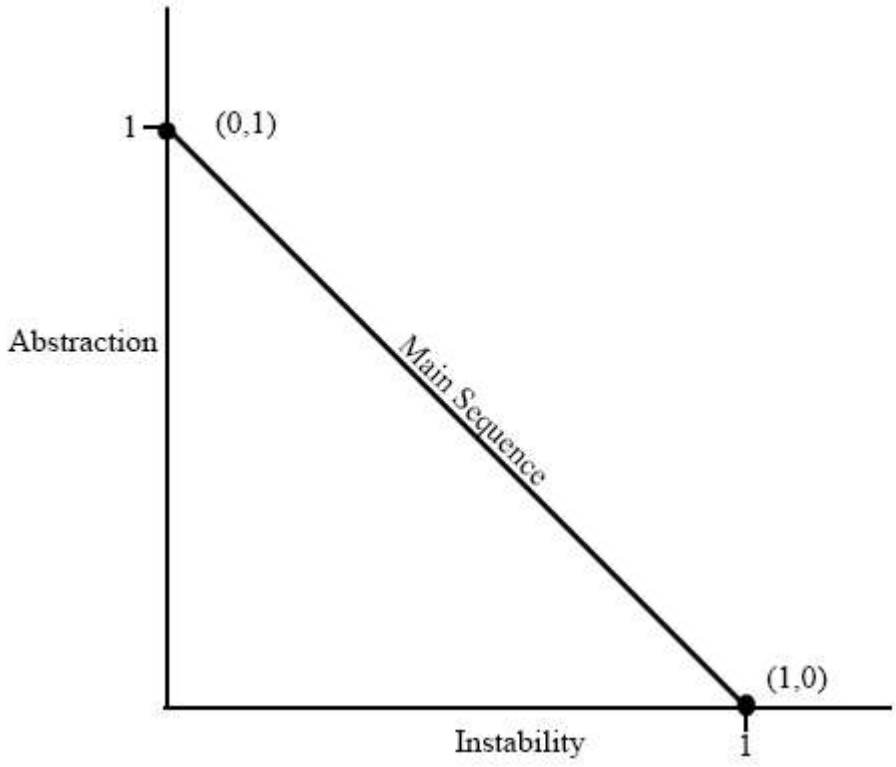
Разстоянието от основната последователност е перпендикулярното разстояние от идеализираната линия A + I = 1.

$$\mathbf{D = | (A + I – 1)/2 |}$$

Диапазонът от стойности е от 0 до ~0.707. Martin препоръчва и нормализация за улеснение на интерпретацията:

$$\mathbf{Dn = | A + I – 1 |}$$

По този начин се постига диапазон от 0 до 1, като при Dn=0 пакета има лежи на линията на основната последователност.



Фиг. 1.4.4.6

1.4.5 Метрики на Chidamber и Kemerer.

1.4.5.1 Сложност на методите на класа.

Сложността на методите на класа (Weighted Methods Per Class) е показател за количеството време и усилия необходими за създаването и поддръжката на класовете. Големият брой методи в класа води до потенциалната възможност за нежелано въздействие над класовете наследници. Класовете с голям брой методи най-вероятно са зависими от конкретно приложение на класа и по този начин намаляват степента на повторно използване (Reusability). Установено е също че броят на грешките е функция на броя методи в клас.

Класове с над 20 метода се считат за излишно усложнени, а такива с над 35 метода се считат за критично тежки и се препоръчва разделяне на класа.

1.4.5.2 Дълбочина на дървото на наследяване.

Дълбочината на дървото на наследяване (Depth of Inheritance Tree) е право пропорционално на сложността на системата. Колкото по дълбоко е един

клас в йерархията толкова повече методи се очаква да наследи, от където и повишаване на сложността. Дълбоките дървета са показател и за сложността на дизайна.

Това обаче не означава, че класовете с най-голяма дълбочина в йерархията имат най-голям риск от грешки. Glasberg и други [Glasberg 00] са установили, че най-податливите на грешки класове се намират по средата на дървото. Според тях коренните и класовете с най дълбока йерархия се ползват често от други класове и поради фамилиарността на разработчиците с тези класове – при тях има по-ниска вероятност за допускане на грешки.

Дълбочина на дървото на наследяване по голяма от 5 води до повишаване сложността на разработката.

1.4.5.3 Брой директни наследници.

Броят на директните наследници (Number of Children) е правопропорционален на степента на повторна употреба. Големите стойности на наследниците пък може да значи и неправилна абстракция на родителския клас. Ако един клас има голям брой наследници може да означава и некачествена схема на наследяване в йерархията на класовете. Разсъжденията тук са аналогични на тези за дълбочината на наследяване, колкото повече класове наследяват даден клас, толкова повече нараства опасността от "замърсяване" на наследниците с ненужни наследени методи.

Броят наследници измерва ширината на дървото на наследяване, докато дълбочината на йерархията – дълбочината на дървото. Препоръчва се с нарастването на дълбочината на дървото на наследяване броят на директни наследници на се намалява.

1.4.5.4 Свързаност на класовете.

Свързаността на класовете (Coupling between Object Classes) е метрика показваща степента на зависимост на класовете един от друг. Два класа са свързани когато методи декларирани в единия използват методи или член променливи декларирани в другия клас. Отношението може да се разглежда и от двете страни – зависим и влияещ, но при всички случаи се брои само едната посока.

Многократните позовавания към един и същ клас се считат за едно позоваване. Само извикване на методи и достъп до променливи се броят. Други типове позоваване като употреба на константи, обработка на събития, употреба на потребителски типове, и инстанциирания на обекти се изключват.

Големи стойности на свързаност не са за предпочитане. Прекомерната свързаност между класовете е вредна за модуларността на дизайна, и възпрепятства

повторната употреба. Колкото по-независим е един клас, толкова по лесна е неговата употреба в друго приложение. С цел повишаване модуларността и индуциране на енкапсулация свързаностите между класове трябва да бъде свеждана до минимум. С нарастването на свързаността на един клас нараства и чувствителността от промени в свързаните с него класове и по този начин нараства и цента за поддръжка на кода. Установено е, че високата степен на свързаност води до предразположен към грешки код, следователно се налага и задълбочаване на тестването.

Степента на споделяне на ресурси между обектите може да бъде интересен показател за "обектната-ориентираност" на системата. Например система, в която един клас има изключително голяма степен на свързаност със останалите класове, а всички останало помежду си нямат никаква свързаност, се оказва всъщност "структурно-ориентирана", а не "обектно-ориентирана" система.

1.4.5.5 Отговорност на клас.

Отговорността на клас (Response for a Class) е размерът на множеството от методи, за които има потенциална възможност да бъдат повикани от методите на класа.

Стойността на метриката се дава, чрез формулата:

RFC = M + R (първо измерване)

RFC' = M + R' (пълно измерване)

M – броят на методите на класа

R – броят на методите принадлежащи на други класове, повикани от методите на класа

R' = броят на методите принадлежащи на други класове, повикани рекурсивно в цялото дърво на извикванията.

Изчислява се броят се уникалните методи.

Тъй-като метриката включва методи извикани извън анализирания клас, тя може да се счита и за метрика измерваща потенциалната комуникация между класа и други класове.

Установено е, че големите стойности на отговорността на класа сочат код с повече грешки. Класовете с голяма отговорност са по-трудно разбираеми, тестването и отстраняването на грешки се усложнява.

RFC е оригиналната дефиниция на метриката. Тя преброява извикванията на външни методи само на първо ниво. С цел подобряване информативността на метриката в последствие се дефинира RFC', с която се измерва пълната отговорност включвайки рекурсивно извикванията на чужди за класа методи. Ако даден метод е полиморфичен, всички възможни методи, които

могат да бъдат извикани се включват в R и R'.

Употребата на RFC' е за препоръчване. RFC отначало е била дефинирана в този си вид, тъй като не е било практично да се анализира пълното дърво на извикванията на ръка.

1.4.5.6 Липса на кохезия между методите.

Метриката "липсата на кохезия между методите" (Lack of Cohesion of Methods), като броят на двойките методи в класа, които не реферират една и съща член-променлива, за първи път е дефинирана от Chidamber и Kemerer през 1991 [Chidamber 91]. В литературата се нарича още LCOM1. Същите автори предлагат и втора дефиниция LCOM2 през 1993 [Chidamber 94].

Дефиницията на LCOM2 се базира на два брояча: P и Q. Ако два метода на един клас използват не пресичащи се множества от член-променливи на същия клас, то стойността на P нараства с единица. Ако разглежданите методи споделят поне една член-променлива, стойността на Q нараства с единица.

Ако P > Q

LCOM2 = P - Q

иначе

LCOM2 = 0

Ниските стойности на LCOM2 сочат голяма степен на свързаност между методите, което е знак за добър дизайн на класа.

Така дефинирана метриката липса на кохезия между методите намира солидна доза закономерни критики. Най-значителния недостатък на този метод е, че LCOM2 има стойност 0 за много коренно различни класове.

Съществува голяма поредица от предложения за подобряване на метриката. Накратко по-важните предложения: LCOM3 [Li 93], LCOM4 и Co (Connectivity)[Hitz 95], TCC (Tight Class Cohesion) и LCC (Loose Class Cohesion) [Bieman 95], DC [Bardi 95], LCOM5 [Henderson-Sellers 96], Coh [Briand 98].

Ще разгледаме дефинициите на LCOM5 и Coh (модификация на LCOM5):

Дефинициите на двата подобрени метода се базират на:

m – броят на методи в класа

a – броят на член-променливи на класа

mA – броят методи, в които се извършва позоваване на дадена член-променлива

sum(mA) – сумата от mA за всички член-променливи на класа.

Coh се дефинира като процента на методите, които не извършват позоваване на дадена член-променлива спрямо всички член-променливи на класа.

Coh = 1 - sum(mA)/(m*a)

Ако броят на методите или член-променливите е равен на 0, то Coh е неопределен и се представя като 0.

LCOM5, познат още като LCOM* се дефинира като:

LCOM5 = (m - sum(mA)/a) / (m-1)

Диапазонът на LCOM5 е от 0 до 2, като LCOM5>1 трябва да се счита като предупреждение за ниска кохезия между методите. При класове с големите стойности на LCOM5 се препоръчва разпределяне на методите на класа в два или повече класа.

Ако класът има само един метод или нито една член-променлива LCOM5 е неопределен и се представя като 0.

2 Проектиране на графичен потребителски интерфейс към основната програма.

За целта на създаването на графичен потребителски интерфейс се спрях на платформата Eclipse (The Eclipse Project). "The Eclipse Project" – това е проект с отворен код, който има за цел осигуряването на стабилна, богата на функционалност, качествена, индустриална платформа за разработка на силно интегриран инструментариум от приложения за развойна дейност. [eclipse.org]

Предлагам кратка история на eclipse.org:

Лидери в ИТ индустрията като IBM, Borland, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft и Webgain създават организациата eclipse.org през ноември, 2001.

През 2002 се присъединяват Serena, Sybase and Fujitsu, Hitachi, Instantiations, MontaVista, Scapa Technologies and Telelogic, ETRI, HP, MKS and SlickEdit, Oracle, Catalyst Systems, Flashline, Parasoft, SAP, teamstudio, TimeSys и Object Management Group (OMG).

През 2003 съответно Fraunhofer Institute, Ericsson, LogicLibrary, M7 Corporation, QA Systems, SilverMark, Inc., Advanced Systems Concepts, Genuitec, INNOOPRACT Informationssysteme GmbH, CanyonBlue, Ensemble Systems, Intel, Micro Focus, Tensilica и Wasabi стават част от eclipse.org.

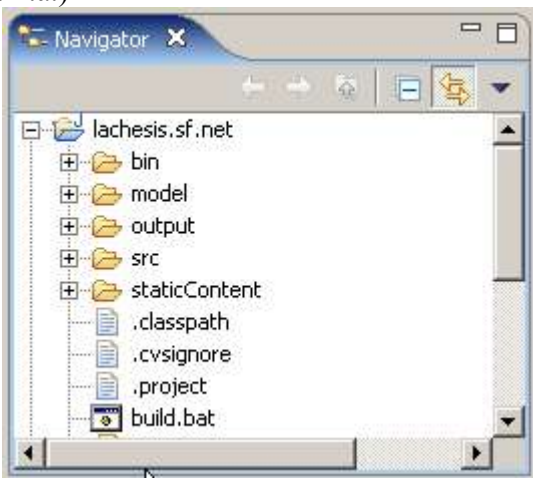
През 2004 проектът става проект с отворен код; Като стратегически разработчици са посочени Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP, Serena Software и Actuate.

Тук ще дефинирам няколко основни за разбирането на Eclipse понятия:

- Изглед (View) – Работен прозорец, който може да съдържа информация от

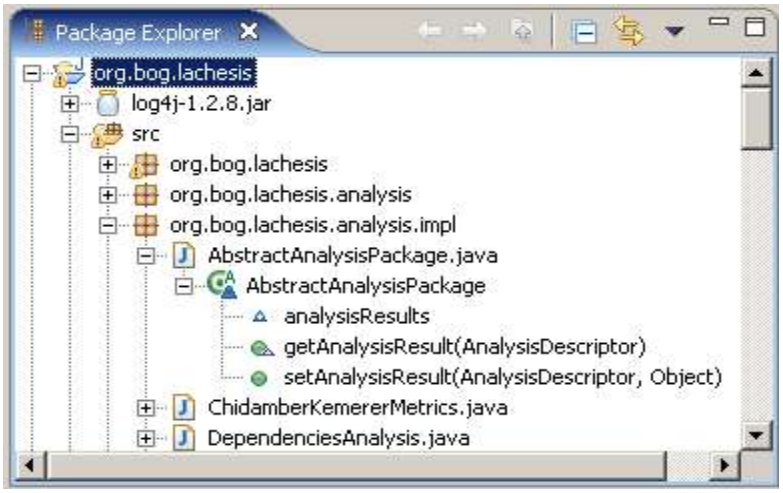
произволен характер и в произволна форма. Позволява се само една инстанция от даден тип изглед. Може да бъде преместван на произволни места в рамките на работната среда. Действията извършени в изгледа се извършват атомарно и със автоматично съхранение, поради тази причина изгледа не предоставя индикатор за променено и не съхранено състояние на данните. Най-често изгледите се ползват за различни форми на навигация сред комплекси от данни, с дървовидна или списъчна структура. Ето и някои от изгледите включени в средата по подразбиране:

- Ресурсен навигатор (Navigator) – служи за навигация сред локална файлова система , предоставя възможност за основни манипулации на файлове и директории (фиг. 2.а.)



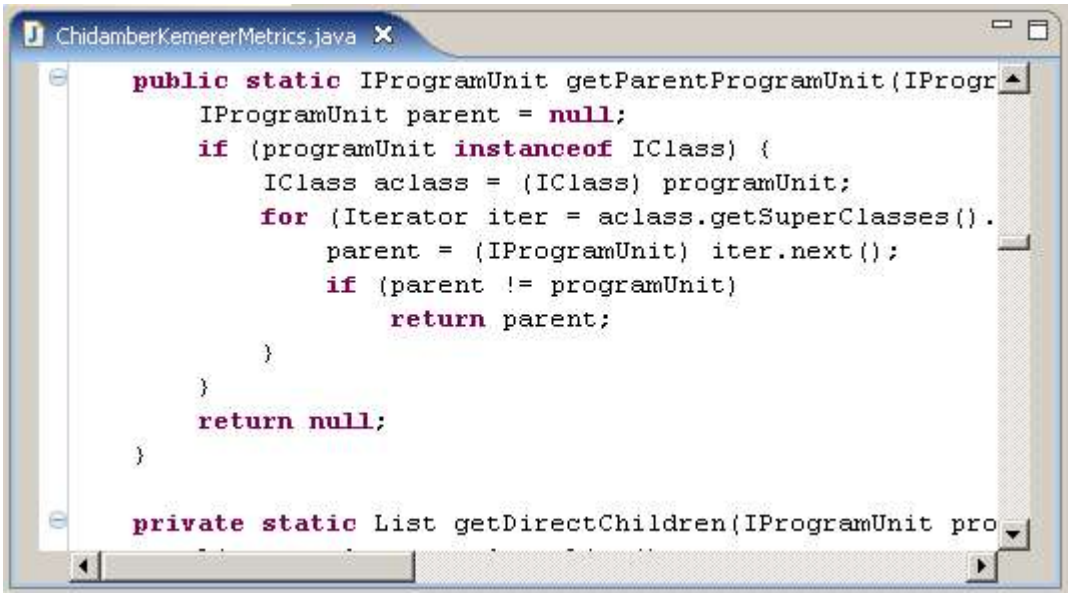
фиг. 2.а

- Навигатор на Java ресурси (Package Explorer) – служи за навигация и редица манипулации над Java ресурси: проекти, пакети, класове (интерфейси), вътрешни класове (интерфейси), член-променливи и методи (фиг. 2.б.).



фиг. 2.б.

- Редактор (Editor) - Работен прозорец, който може да съдържа информация от произволен характер и в произволна форма. Позволяват се множество инстанции от даден тип редактор. Не може да бъде преместван на произволно място в рамките на работната среда. Промените на съдържанието, извършени в редактора, не се съхраняват автоматично, а само след изрично извикване на действие "съхрани" (Save). Съответно редактора предоставя индикатор за променено и не съхранено състояние на данните. Най-често редакторите се използват редактиране данни в разнообразни формати: ascii/unicode текст, първичен програмен код на различни програмни езици, XML, HTML и т.н. Един от най-известните редактори, включен по подразбиране в средата, е редактора за Java първичен код (Java Editor) (фиг.2.в).



фиг. 2.в.

Перспектива (Perspective) – набор от изгледи и редактори предназначени за сходен род дейности; Примери: Работа с ресурси (Resource) - работа с файлови ресурси от локалната файлова система (включва изгледът Navigator); Работа с Java ресурси (Java) – (включва изгледът Package Explorer) Debug DebugCVS CVS

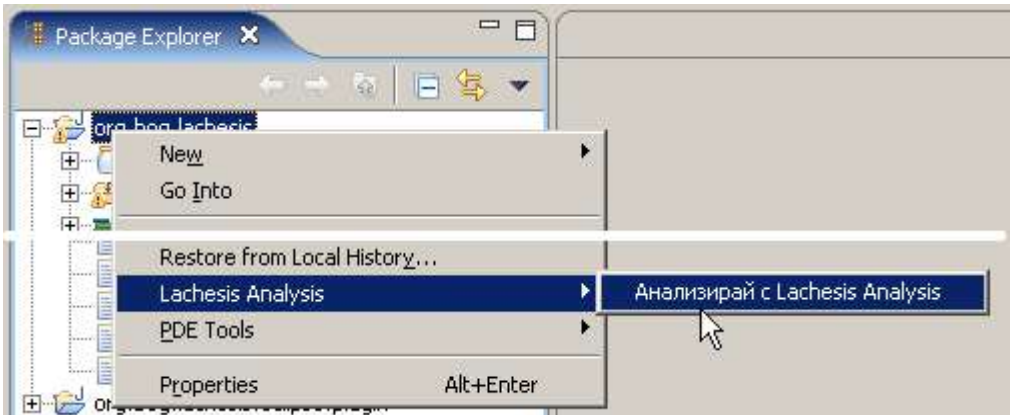
2.1 Контекстно меню.

Стартирането на процеса на анализ се осъществява след първоначален избор на проект, чиито ресурси ще бъдат анализирани. Селекцията на проекти е функционалност предоставена от самата среда, в случая говорим за селектиране, което се осъществява в изглед Package Explorer. Това което трябва да направим в допълнение е да регистрираме разширение на точката за разширение за опции в контекстно меню, като регистрираме опцията само за обекти от тип Java проект.

Извадка от plugin.xml:

```
   <extension
         point="org.eclipse.ui.popupMenus">
      <objectContribution
            objectClass="org.eclipse.jdt.core.IJavaProject"
            id="org.bog.lachesis.eclipse.plugin.contribution1">
         <menu
               label="%lachesis"
               path="additions"
               id="org.bog.lachesis.eclipse.plugin.menu1">
            <separator
                  name="group1">
            </separator>
         </menu>
         <action
               label="%analyze.with.lachesis"
               class="org.bog.lachesis.eclipse.plugin.popup.actions.Acti
onNewAnalysis"
               menubarPath="org.bog.lachesis.eclipse.plugin.menu1/group1"
               enablesFor="1"
               id="org.bog.lachesis.eclipse.plugin.ActionNewAnalysis">
         </action>
      </objectContribution>
   </extension>
```

Като резултат от регистрацията на разширение в контекстното меню на Java проект се появява нова опция (фиг. 2.1.a).

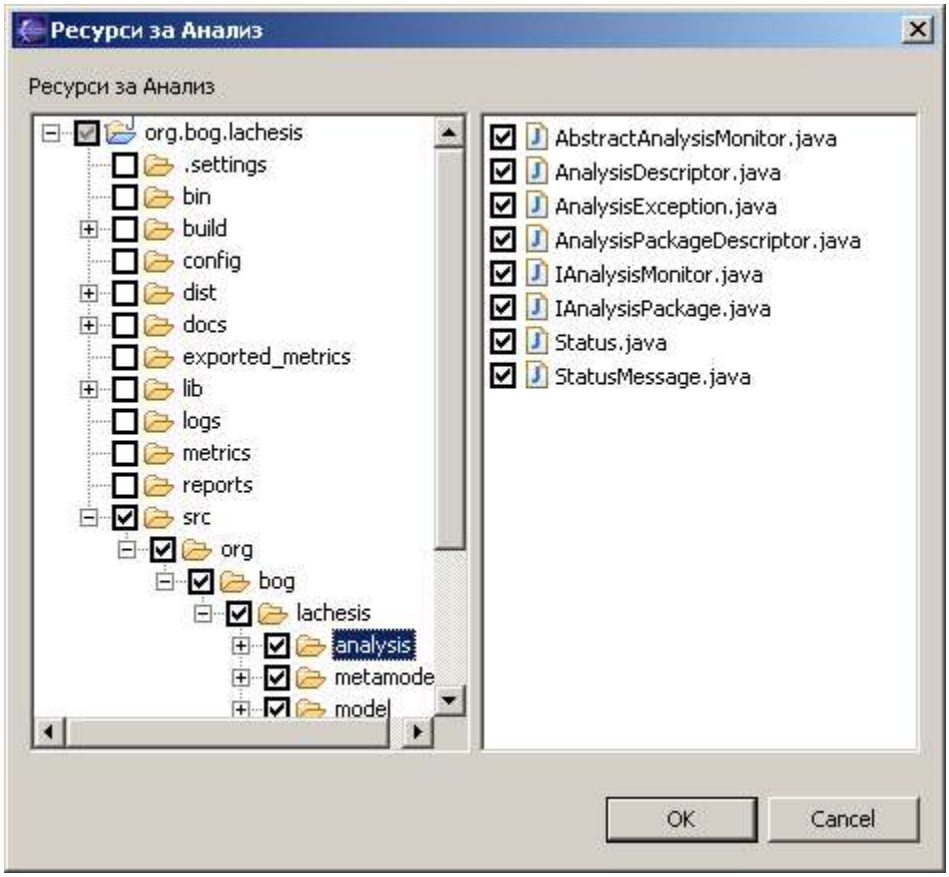


фиг. 2.1.a.

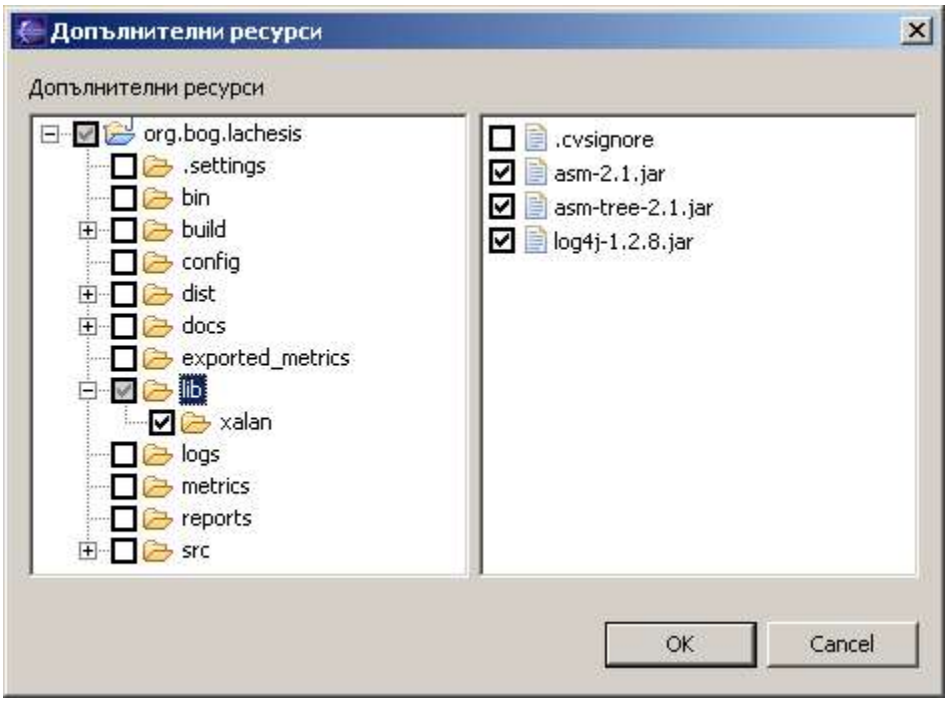
2.2 Диалози за избор на ресурси.

За да бъдат покрити изискванията за "1. Необходимост и достатъчност" потребителският интерфейс трябва да предостави възможност на потребителя да избира ресурси от даден Java проект с точност до файл. За целта се ползва диалог за избор на ресурси **org.eclipse.ui.dialogs.ResourceSelectionDialog**. Това е стандартен диалог за избор на ресурси. Компонентът предоставя възможност за първоначално установяване на избраните ресурси. От тази функционалност се възползва графичният интерфейс с цел улесняване на потребителя: рекурсивно се

обхождат всички поддиректории на директориите отбелязани като контейнери на първичен програмен код и всеки срещнат файл с разширение java се маркира като избран. Потребителя има възможността да модифицира първоначалната селекция на ресурси по свой избор. Последователно се извеждат два диалога: един за избор на ресурси за анализ (фиг. 2.2.а) и един за избор на допълнителни ресурси (фиг. 2.2.б)



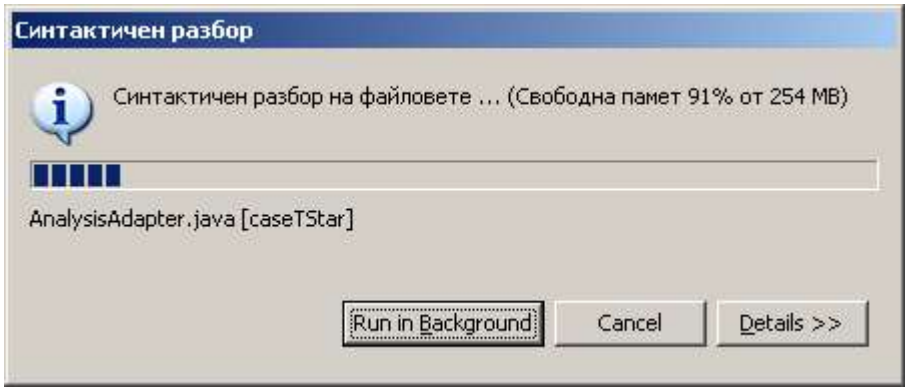
фиг 2.2.а



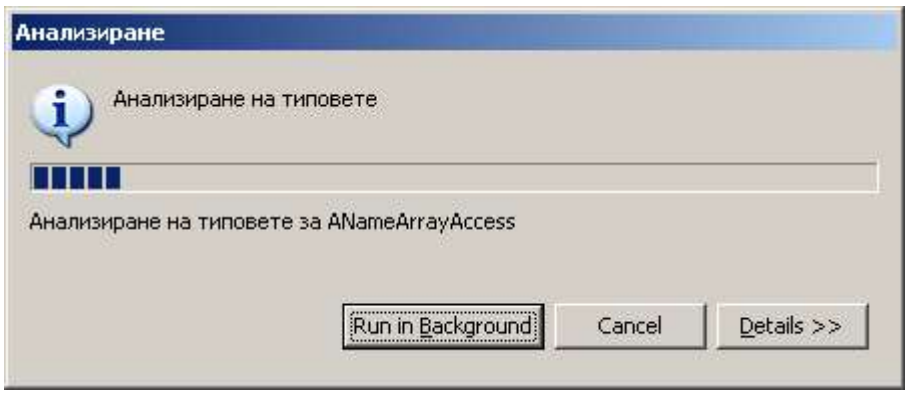
фиг. 2.2.б

2.3 Индикатор за прогреса на анализа.

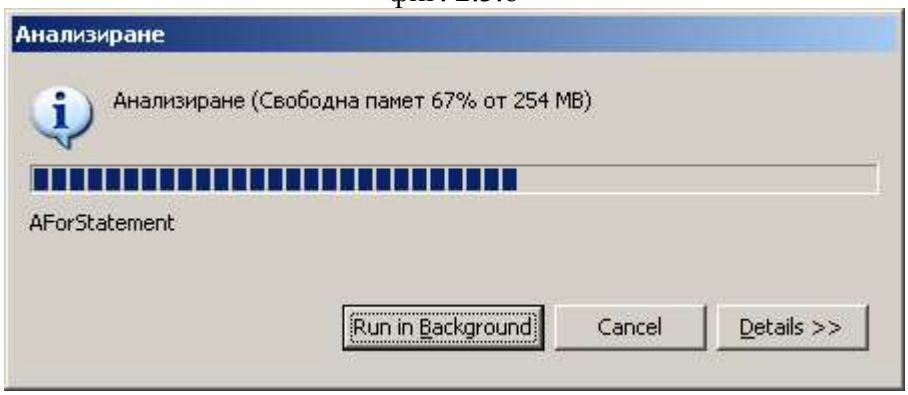
За да бъде покрито изискването "3. Удобство на употреба" в частта си за индикация на прогреса на процеса на анализ добавяме стандартни диалози за прогрес. (Естествено процеса на анализ същевременно поддържа механизъм за отчитане на дейността в реално време – **org.eclipse.core.runtime.IProgressMonitor**). За всяка фаза от анализа се представя отделен диалог за прогреса: синтактичен анализ (фиг. 2.3.а), семантичен анализ (анализ на типовете) (фиг. 2.3.б) и анализ на метрики (анализ) (фиг.2.3.в).



фиг. 2.3.а



фиг. 2.3.б



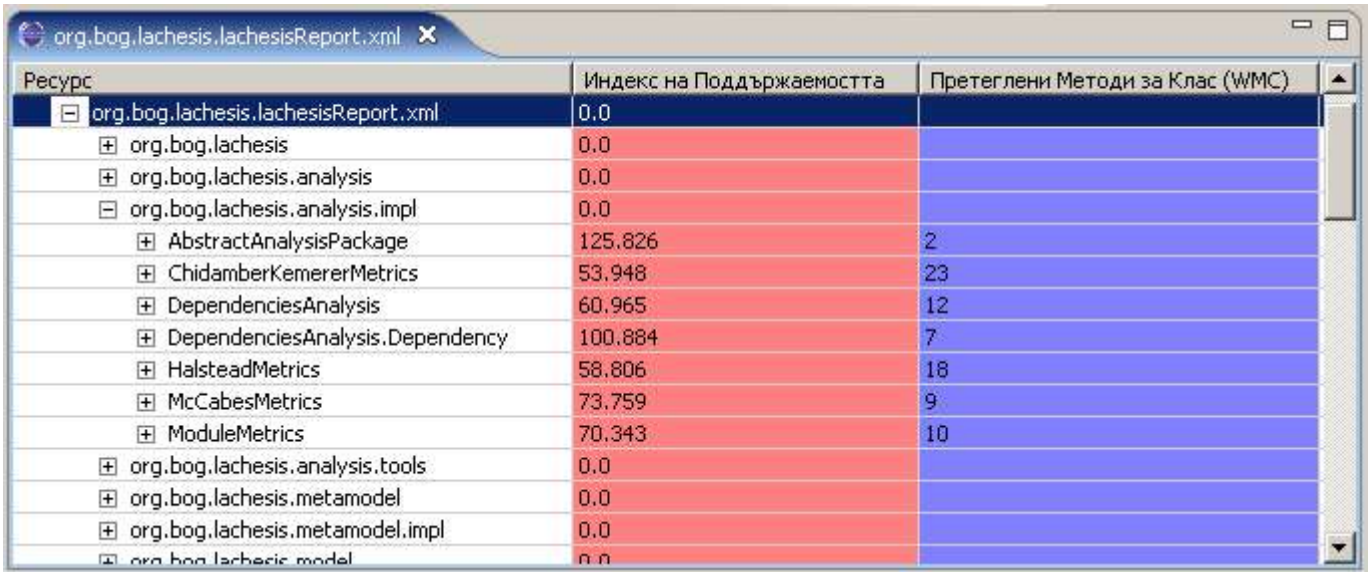
фиг. 2.3.в

2.4 Редактор на модел.

Въпреки, че моделът, който се получава след анализ не може да бъде модифициран, се използва редактор наместо изглед, тъй като е за предпочитане да се отваря отделен прозорец за всеки анализ (фиг. 2.4.а).

Извадка от plugin.xml:

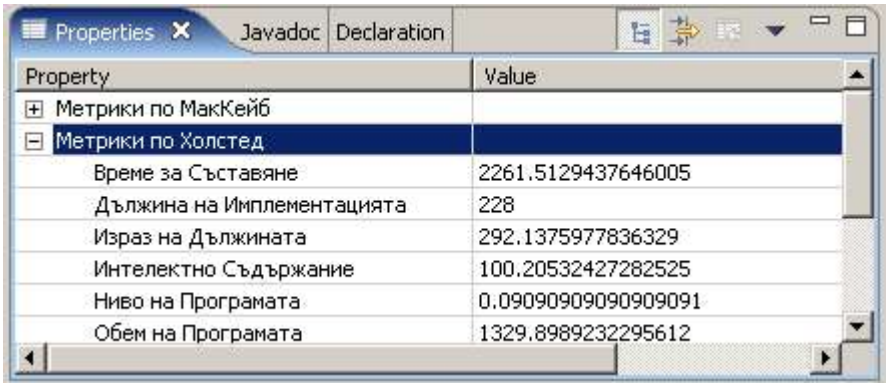
```
<extension point="org.eclipse.ui.editors">
   <editor
       name="Lachesis Metrics Models"
       icon="icons/sample.gif"
       class="org.bog.lachesis.eclipse.plugin.editors.LachesisEditor"
       id="org.bog.lachesis.eclipse.plugin.editors.LachesisEditor">
   </editor>
</extension>
```



фиг. 2.4.а

2.5 Изглед на метрики.

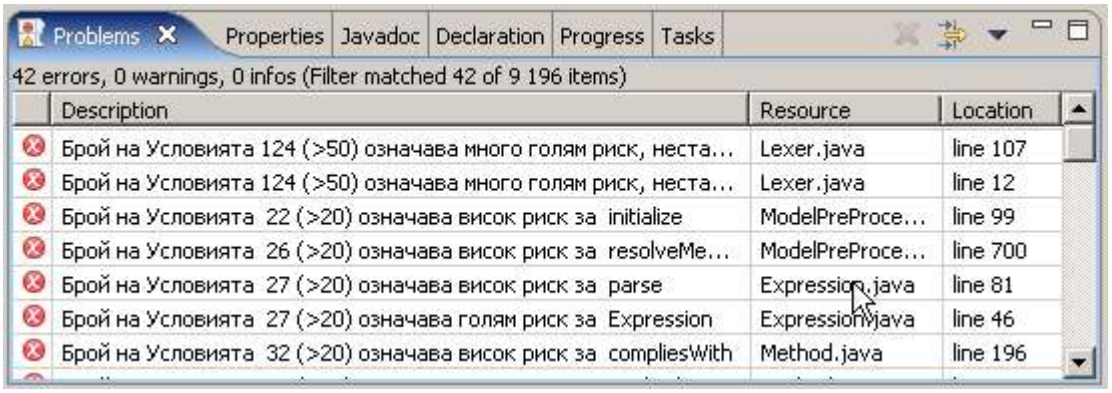
Тъй-като броят на метриките по дефиниция е значителен, а и винаги може да нараства допълнително, при добавяне на нови пакети от метрики или нови метрики в наличните пакети, е необходимо метриките да се показват в отделен прозорец, различен от този на модела. За целта се прави разширение което допринася нови типове обекти за репрезентация в стандартния изглед наречен "характеристики" (Properties) (фиг. 2.5.а). Броят на видимите характеристики в този изглед се контролира от същите настройки, които се ползват и при показване на колоните на редактора на модел.



фиг.2.5.а

2.6 Маркери на съобщения.

Средата Eclipse притежава вграден механизъм за представяне на съобщения директно свързани с определен ресурс (файл, клас, метод, и т.н.), и автоматичното навигиране към ресурса при селекция на дадено съобщение. За създаването на съобщение се създава съответен маркер. Всички съобщения се извеждат автоматично в стандартния изглед "Проблеми" (Problems) (фиг. 2.6.a).



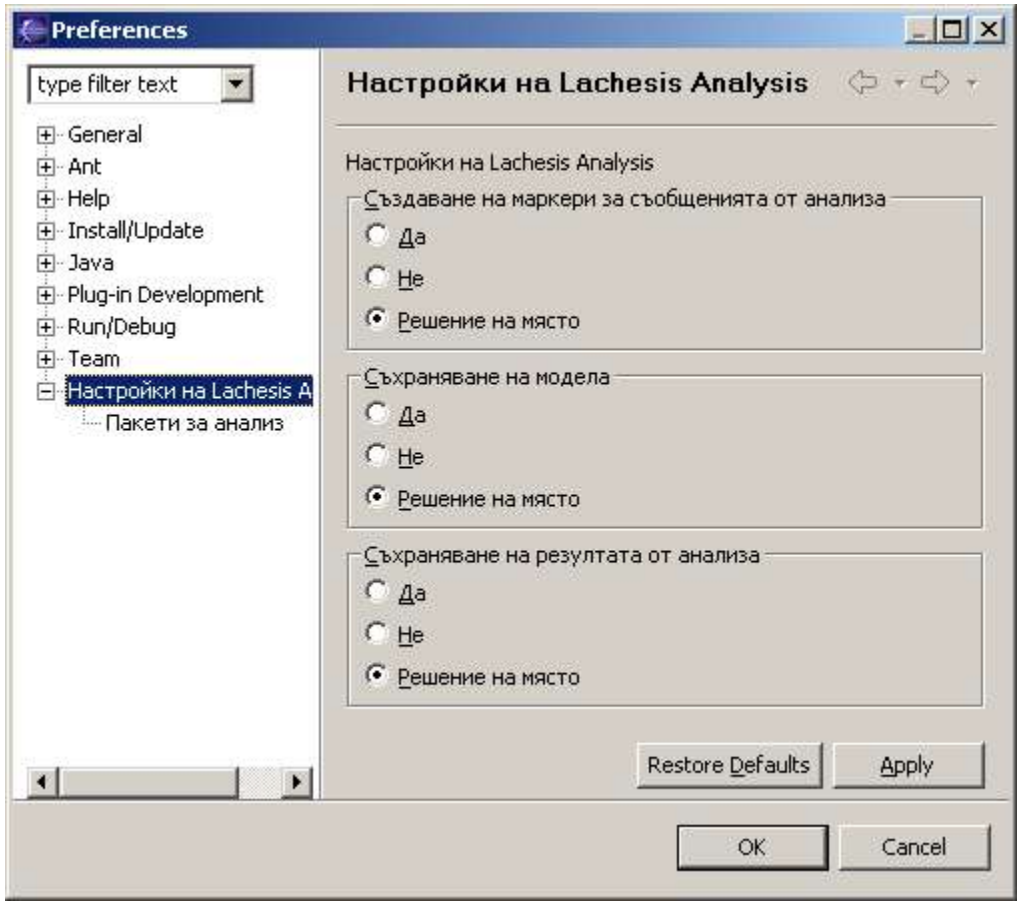
фиг. 2.6.a

2.7 Настройки.

Средата Eclipse предоставя стандартен подход за представяне на настройки на инструменти. Потребителският интерфейс дефинира две страници със специфични настройки: общи настройки и настройки за визуализиране на метриките.

Извадка от plugin.xml:

```
   <extension
         point="org.eclipse.ui.preferencePages">
      <page
            name="%preferences"
            class="org.bog.lachesis.eclipse.plugin.preferences.MainPage"
            id="org.bog.lachesis.eclipse.plugin.preferences.MainPage">
      </page>
      <page
            name="%preferences.analysis.packages"
            category="org.bog.lachesis.eclipse.plugin.preferences.MainP
age"
            class="org.bog.lachesis.eclipse.plugin.preferences.Analysis
PackagesPage"
            id="org.bog.lachesis.eclipse.plugin.preferences.AnalysisPac
kagesPage">
      </page>
   </extension>
```



фиг 2.7.а

2.8 Локализация на потребителския интерфейс.

Средата Eclipse предоставя удобен механизъм за локализация на програми. Процеса на локализация се изразява в създаване на речник и неговата употреба в първичния програмен код. Средата предоставя средства са създаване на речник на базата на автоматично извличане на символни низове от първичния програмен код (Source->Find Strings to Externalize). Инструмента извлича символните низове употребени в кода и редом с това заменя употребата им в кода със програмно извикване към метод на клас отговорен за изваждането на дума от речника на базата на ключ.

3 Възможност за разширение, чрез добавяне на граматика на програмен език.

Програмната реализация предвижда механизъм за лесно добавяне на поддръжка за произволен програмен език. Механизмът се базира на абстракциите "източник на модел" и "четец на модел" дефинирани съответно с Java интерфейсите:

org.bog.lachesis.model.IModelSource

org.bog.lachesis.model.IModelReader.

"Четец на модел" представлява интерфейс позволяващ дефиниране на входни ресурси – ресурси за анализ и допълнителни ресурси. Интерфейсът изисква за всеки ресурс да бъде предоставен обект от произволен Java клас. Този клас може да представлява файл, мрежова връзка, структура от данни в оперативната памет и т.н. Отговорността по интерпретация на типа на ресурса се пада на "четеца на модел". "Четец на модел" представлява интерфейс, който дефинира абстрактното действие "прочети модел" като основния параметър е инстанция имплементираща интерфейса "четец на модел". Като минимална реализация са представени и най-тривиалните представители на двете абстракции: "Файлов източник на модел" и "Четец на Java програмни ресурси", представени от двойката класове:

org.bog.lachesis.model.sources.FileModelSource
org.bog.lachesis.model.readers.JavaPackageModelReader.

За да се добави поддръжка за друг програмен език най-често ще се налага имплементация единствено на интерфейса **IModelReader**, тъй като почти при всички програмни езици първичния програмен код и допълнителни дефиниции се съхраняват във файлове.

Така дефинирани абстракциите позволяват дори ресурси, които са създадени на базата на мета-модели, които не могат да бъдат класифицирани като програмни езици; единственото изискване е тези мета-модели да дефинират модели, които са трансформируеми до модели базирани на езиково-независимия обектно-ориентиран мета-модел използван от семантичния анализ.

4 Добавяне на пакет от софтуерни метрики.

Реализацията на програмата предоставя проста, но гъвкава мета-дефиниция за описание на пакети от софтуерни метрики и метриките включени в пакетите.

Разширяемостта относно пакети софтуерни метрики е възможна благодарение на абстракцията "пакет със анализи" представена от интерфейса:

org.bog.lachesis.analysis.IAnalysisPackage.

Този интерфейс има две основни функции – доставяне на дескриптор на пакет от софтуерни метрики и извличане на резултат за дадена метрика, принадлежаща на пакета.

```
public AnalysisPackageDescriptor getAnalysisPackageDescriptor();
public Object getAnalysisResult(AnalysisDescriptor analysisDescriptor);
```

По този начин всеки пакет софтуерни метрики може самостоятелно да се грижи за предоставяне на дефиницията на метриките включени в него, както и стойността на всяка метрика за определен обект от езиково-независимия модел. Всеки пакет от софтуерни метрики има възможността да излъчи съобщение относно дадена метрика чрез класа **org.bog.lachesis.analysis.StatusMessage**.

За добавяне на пакет от софтуерни метрики е досатъчно да се предостави имплементация на интерфейса

org.bog.lachesis.analysis.IAnalysisPackage,

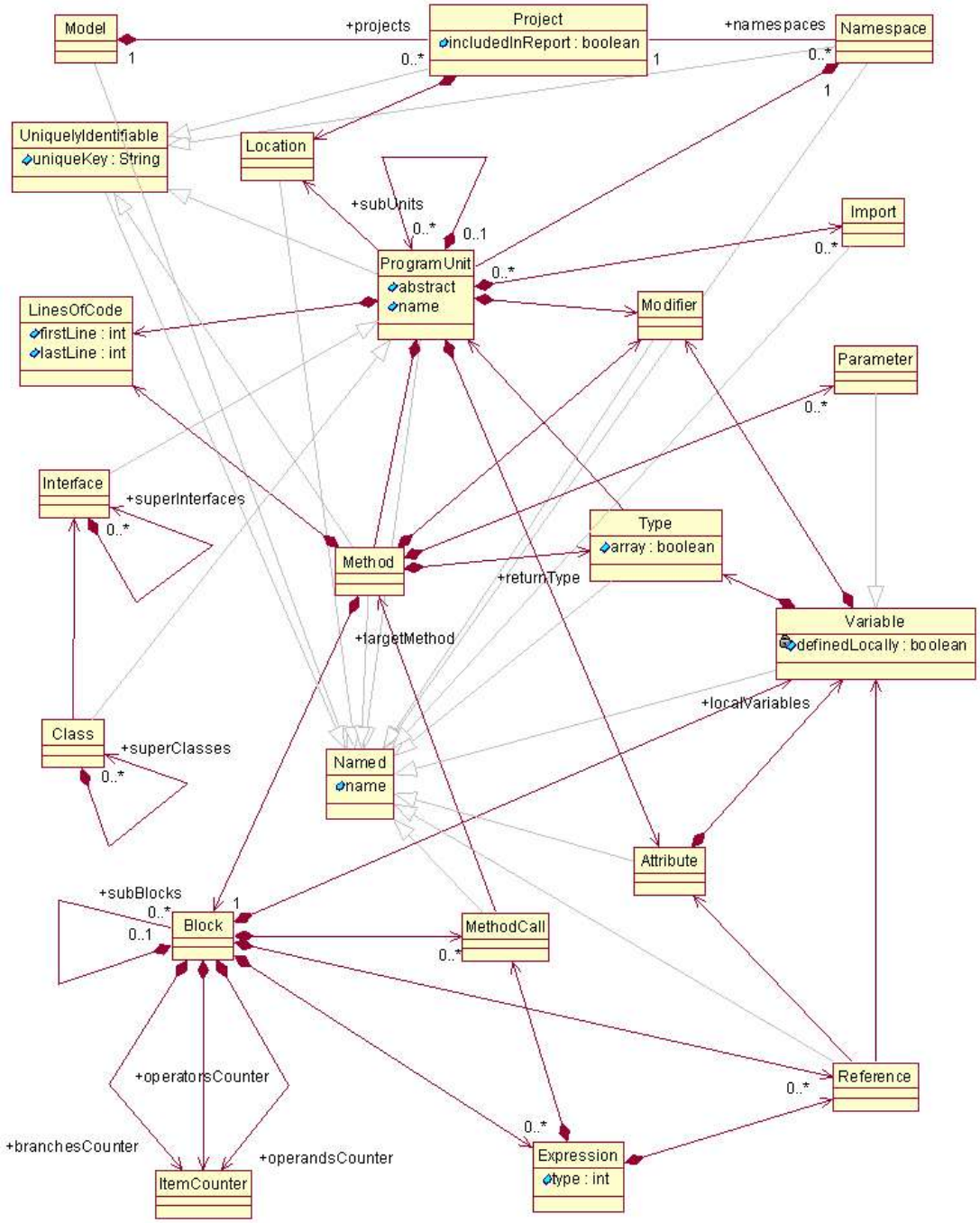
както и да се включи пакетът за изчисление в класа

org.bog.lachesis.analysis.tools.GlobalAnalysis

- извикващ всички анализи върху различните обекти от езиково-независимия модел.

5 Особености на програмната реализация.

5.1 Езиково-независим Обектно-Ориентиран мета-модел.



Фиг. 5.1

С цел разширяемост, от гледна точка на използувания програмен език за съставяне на ресурсите подложени на анализ, представянето на моделите на анализираните програми се извършва на базата на обектно-ориентиран мета-модел, чиято дефиниция не е зависима от един конкретен обектно-ориентиран език за програмиране. Мета-моделът в известна степен може да се разглежда като смес от подмножество на UML и мета-моделите на програмните езици Java и C#.

Репрезентацията на мета-модела в рамките на Java е осъществен посредством набор от интерфейсни класове, абстрактни класове и конкретни класове.

5.1.1 Интерфейси.

org.bog.lachesis.metamodel

Мета-моделът като дефиниция се представя от набор от интерфейси.

5.1.2 Абстрактни класове.

org.bog.lachesis.metamodel.impl

Абстрактната реализация на интерфейсите на мета-модела включва базова функционалност за поддръжка на структурите от данни обусловени от мета-модела, например асоциирането и композирането на обекти от други обекти. Абстрактната реализация предоставя и генерация на XML съдържание за всеки елемент от мета-модела.

5.2 Модели.

Модел представлява всяка имплементация на интерфейсите на мета-модела. Препоръчително е тази имплементация да наследява абстрактната имплементация на мета-модел, с цел повторна употреба на генераторите на XML съдържание.

5.2.1 Източници на модели.

Абстракцията източник на модел се дефинира от интерфейса:

org.bog.lachesis.model.IModelSource

Тривиална имплементация на интерфейса осигуряваща достъп до файлове:

org.bog.lachesis.model.sources.FileModelSource

5.2.2 Четци на модели.

Абстракцията четец на модел е представена от интерфейса:

org.bog.lachesis.model.IModelReader

Отговорността на всеки четец на модели е да предаде ресурсите за анализ и допълнителните ресурси на "разпознавател на модели". Разпознавателят на модели може да бъде машина за синтактичен разбор, когато трябва да се обработи първичен програмен код и де-компилатор, когато трябва да се обработи обектен програмен код.

Имплементация на интерфейса за работа с Java програмен код – първичен и обектен – е представена в класа:

org.bog.lachesis.model.readers.JavaPackageModelReader

Основната функция на този четец на модели е да предаде файловия ресурс на един от двата разпознаватели на модели – за първичен програмен код и за обектен програмен код (в случая това е Java bytecode).

5.2.2.1 Разпознаватели на модели на Java програми.

Кодът по извличане на модели от Java програмен код е съсредоточен в пакетите:

org.bog.lachesis.recognizers.sourcecode.java

org.bog.lachesis.recognizers.binarycode.java

5.2.2.1.1 Лексически и синтактичен анализ на първичен код.

За целите на разпознаването на първичния програмен код се използва класически подход: първичният програмен код се подлага последователно на лексически и синтактичен анализи. Последните се извършват на базата на дефиницията на граматиката на програмния език. Създаването на лексически и синтактичен анализатор на базата на спецификация на граматиката на програмен език, описана във вербална форма, е изключително енергоемка задача. За щастие съществуват множество проекти с отворен код, които предоставят т.н. генератори на синтактични анализатори (парсери; Parsers), които генератори приемат като входни данни дефиницията на формален език в някаква опростена декларативна форма. Най-често се използва Extended Backus-Naur Form (EBNF) – Разширена Бакус-Наурова Нотация.

Ето и по-известните създадени на програмен език Java парсер генератори:

- ANTLR – генерира LL(k) парсери, поддържа генериране на Абстрактно Синтактично Дърво (Abstract Syntax Tree - AST) [antlr].
- JavaCC – генерира LL(k) парсери, поддържа генериране на Абстрактно

Синтактично Дърво [javacc]

- SableCC - генерира LALR(1) парсери, поддържа генериране на Конкретно Синтактично Дърво (Concrete Syntax Tree – CST) [sablecc].

Първоначално беше използван ANTLR, но в процеса на разработка се оказа, че нуждата от работа с конкретни синтактични дървета е значителна и тъй-като ANTLR автоматично не предоставя такива, JavaCC също. Поради тази причина текущо се ползва парсер генериран с помощта на SableCC. Кодът на парсера се намира в пакети:

org.bog.lachesis.recognizers.sourcecode.java.sablecc.grammar

org.bog.lachesis.recognizers.sourcecode.java.sablecc.lexer

org.bog.lachesis.recognizers.sourcecode.java.sablecc.node

org.bog.lachesis.recognizers.sourcecode.java.sablecc.parser

Трансформацията от специфичен програмен модел предоставен от синтактичния анализатор (Конкретното Синтактично Дърво) към Езиково-независимия Обектно-Ориентиран модел се осъществява от класовете в пакет:
org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst

5.2.2.1.2 Лексически и синтактичен анализ на обектен код.

Макар и трудно забележимо, при обработката на обектен код отново имаме лексически и синтактичен анализ. В случая лексемите са примитивите на Java bytecoda-a, а граматиката е самия формат на bytecode-a.

Съществуват два популярни инструмента за анализ и манипулация на Java bytecode:

- Byte Code Engineering Library – проект на Apache Software Foundation [bcel].
- ASM, Java bytecode manipulation framework – проект на ObjectWeb Consortium.

Първоначално в текущата разработка беше включен BCEL, но в последствие се установи, че не може да анализира правилно Java 5 bytecode. Авторите на ASM твърдят, че времето на манипулиране на един клас е 10 по малко от времето за което BCEL извършва същата дейност. Освен това ASM поддържа bytecode на Java 5.

Трансформацията от специфичен програмен модел предоставен от анализатора на bytecode към Езиково-независимия Обектно-Ориентиран модел се осъществява от класа:

org.bog.lachesis.recognizers.binarycode.javabytecode.BCModelExtractor

5.2.2.2 Семантичен анализ. Пре-процесор на модели.

Пре-процесора на модели осъществява семантичен анализ над получения езиково-независим модел. В семантичния анализ се включват следните задачи:

- Анализ на използваните от една програмна единица външни за нея типове.
- Анализ на типовете определящи йерархията на класовете.
- Анализ на член-променливи – определяне на типовете на член-променливата.
- Анализ на методи – определяне на типа на връщана стойност на методите, определяне на типовете на параметрите на методите.
  - ✔ Анализ на блок на методи
    - Анализ на типовете на локалните променливи.
    - Анализ на достъпите до локални променливи.
    - Анализ на достъпите до член променливи на класове.
    - Анализ на извикванията на методи.
      - Анализ на аргументите на метод – определяне типа на изразите поставени като аргументи на метода.

Класът реализиращ семантичния анализ е:

org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor

5.3 Пакети от софтуерни метрики.

5.3.1 Дефиниция.

Абстракцията на "пакети от анализи" се намира в интерфейса:

org.bog.lachesis.analysis.IAnalysisPackage

Всеки пакет от софтуерни метрики притежава дескриптор описващ метриките, включени в пакета:

org.bog.lachesis.analysis.AnalysisPackageDescriptor

Дескриптора на пакети от своя страна съдържа множество дескриптори на метрики:

org.bog.lachesis.analysis.AnalysisDescriptor

Инфраструктурата за изчисление на софтуерни метрики дефинира и анстракция на монитор на прогреса на анализа:

org.bog.lachesis.analysis.IAnalysisMonitor

org.bog.lachesis.analysis.AbstractAnalysisMonitor

За този монитор в последствие се предлагат и имплементации: с елементарна конзолна нотификация и такава заглушаваща съобщенията за изменение на прогреса на анализа:

org.bog.lachesis.tools.ConsolePrintMonitor

org.bog.lachesis.tools.NullAnalysisMonitor

Инфраструктурата за анализ на метрики също така предоставя и механизъм за натрупване на съобщения получени по време на анализа – състояние и съобщение за състояние:

org.bog.lachesis.analysis.Status

org.bog.lachesis.analysis.StatusMessage

По време на анализ всички пакети от софтуерни метрики отнесени към даден обект от езиково-независимия модел се натрупват в:

org.bog.lachesis.analysis.tools.AnalysisRecord

5.3.2 Реализации.

Целият процес по анализ на метрики се управлява от класа:

org.bog.lachesis.analysis.tools.GlobalAnalysis

В този клас също се осъществява и генериране на XML съдържание за пакетите от софтуерни метрики.

5.3.2.1 Метрики на Halstead.

org.bog.lachesis.analysis.impl.HalsteadMetrics

5.3.2.2 Метрики на McCabe.

org.bog.lachesis.analysis.impl.McCabesMetrics

5.3.2.3 Обобщаващи метрики.

org.bog.lachesis.analysis.impl.ModuleMetrics

5.3.2.4 Анализ на Зависимостите.

org.bog.lachesis.analysis.impl.DependenciesAnalysis

5.3.2.5 Метрики на Chidamber и Kemerer, и Hednerson-Sellers.

org.bog.lachesis.analysis.impl.ChidamberKemererMetrics

5.4 Отчет на анализ.

Отчетът на анализ, както и на модел-а, над който е извършен анализа се осъществява посредством генерация на XML съдържание, запис на генерираното съдържание във файл и последващото му компресиране.

5.4.1 XML и ZIP.

Генерираното XML съдържание се записва във файл, който се компресира с помощта на вградения в средата Java инструментариум за работа със ZIP архиви. Класа извикващ функционалността за компресиране е:

org.bog.lachesis.reports.ZipTools

5.4.2 Времеви профили.

С цел мониторинг и усъвършенстване на програмния код на текущата разработка, много-таймерова система е включена в класа:

org.bog.lachesis.profiler.Profiler

5.5 Конзолен потребителски интерфейс.

Програмата предоставя и най-простия потребителски интерфейс – конзолния:

org.bog.lachesis.tools.Lachesis

5.6 Графичен потребителски интерфейс.

Построяването на графичния потребителски интерфейс е реализирано в проект наречен: **org.bog.lachesis.eclipse.plugin**. Представлява стандартна добавка към популярната платформа Eclipse.

6 Ръководство на потребителя.

1. Минимални изисквания:

- Операционна система, на която може да бъде инсталирана Java Virtual Machine.
- Няма други минимални изисквания.

За нормална работа обаче се препоръчва 256 MB обем памет за виртуалната машина на Java.

2. Инсталация на Java Virtual Machine:

За да инсталирате Java Virtual Machine трябва да се сдобиете със софтуерния пакет **J2SE Software Development Kit (SDK)**, част от **Java 2 Platform, Standard Edition, v 1.4.2 (J2SE)**.

- От инсталационен CD носител: **install/j2sdk-1_4_2_01-windows-i586.exe**
- От официалната страница в Интернет:

http://java.sun.com/j2se/1.4.2/download.html

6.1 Инсталиране на конзолно приложение.

1. Снабдяване с архив на програмата.

- От инсталационен CD носител: **install/org.bog.lachesis.eclipse.plugin_x.y.z.jar**
- От официалната страница в Интернет:

http://lachesis.sourceforge.net/download.html

2. Декомпресиране на архива.

Декомпресия на архива org.bog.lachesis.eclipse.plugin_x.y.z.jar можете да извършите с произволна програма способна да декомпресира ZIP архиви. Ако е необходимо може да промените разширението на файла от jar на zip за да бъде разпознат автоматично от вашата програма за де-архивиране.

- За Windows потребители: **WINZIP**, **WINRAR**, **WINACE** и др.
- За Unix/Linux: **unzip**

3. Инсталиране.

След декомпресиране на архива трябва да извлечете пълните пътища до файловете lib/lachesis.core.jar, lib/asm-2.1.jar, lib/asm-tree-2.1.jar и lib/log4j-1.2.8.jar, и да ги добавите към променливата CLASSPATH от обкръжението (Environment) на вашата операционна система. Пътищата се разделят с точка и запетая ";".

Пример за Windows потребители:

```
set CLASSPATH %CLASSPATH%;C:\deploy\lachesis\asm-2.1.jar;
C:\deploy\lachesis\asm-tree-2.1.jar;
C:\deploy\lachesis\lachesis.core.jar;C:\deploy\lachesis\log4j-1.2.8.jar
```

6.2 Инсталиране на графичен потребителски интерфейс.

За инсталирането на графичен потребителски интерфейс се нуждаете от предварително инсталирана работна среда Eclipse.

1. Снабдяване с архив на работната среда Eclipse.

- От инсталационен CD носител: **install/eclipse/eclipse-SDK-3.1-win32.zip**
- От официалната страница в Интернет: **http://eclipse.org/downloads/index.php**

2. Инсталация на работна среда Eclipse.

Архивът **eclipse-SDK-3.1-win32.zip** се декомпресира и работната среда е готова за употреба.

3. Снабдяване с архив на програмата.

- От инсталационен CD носител: **install/org.bog.lachesis.eclipse.plugin_x.y.z.jar**
- От официалната страница в Интернет:

http://lachesis.sourceforge.net/download.html

4. Инсталация на plug-in с основната програма и графичен потребителски интерфейс.

Архивът **org.bog.lachesis.eclipse.plugin_x.y.z.jar** се копира в директория **plugins** на декомпресираната работна среда Eclipse.

6.3 Работа с конзолно приложение.

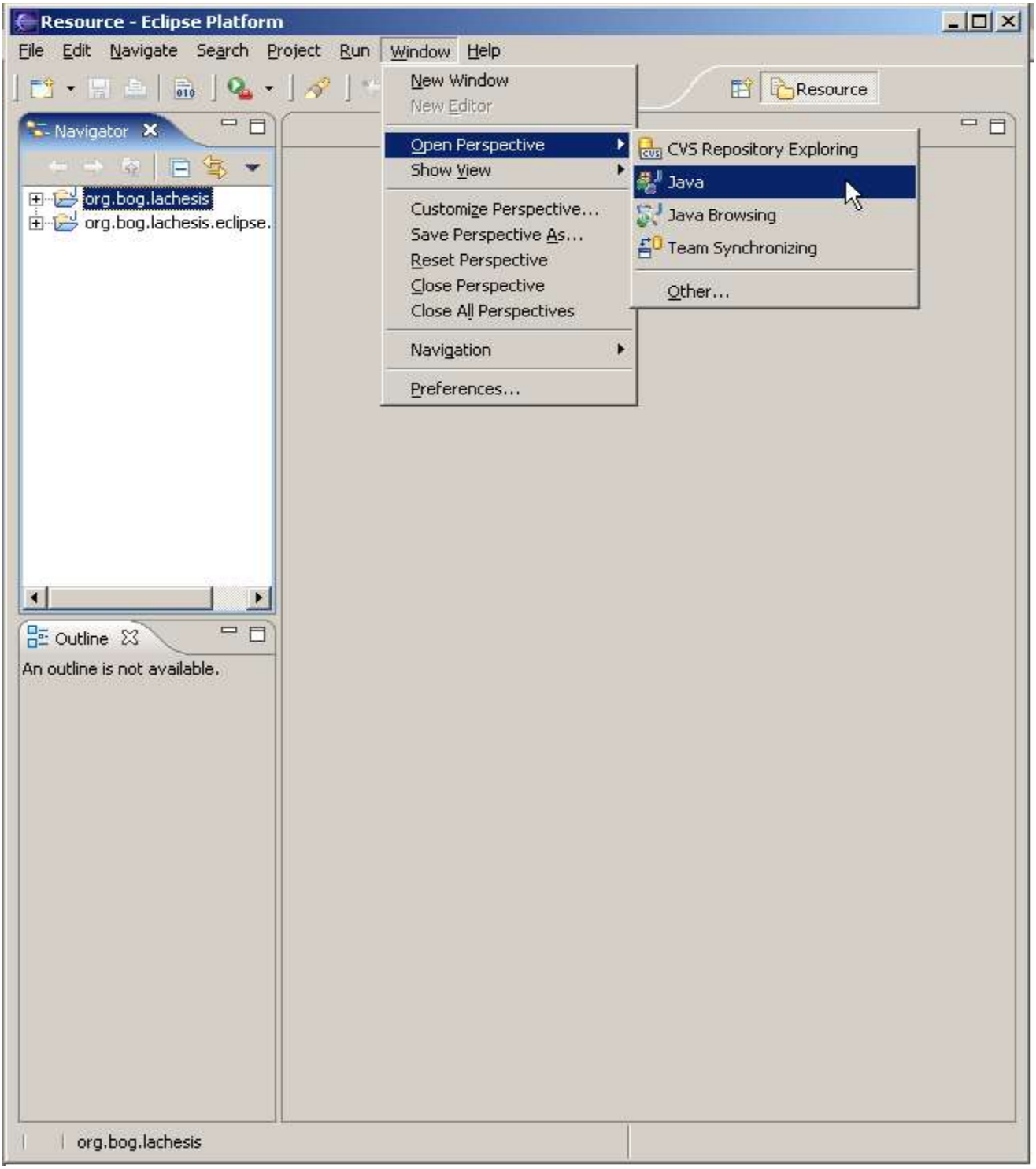
Конзолното приложение се извиква с командата:

java org.bog.lachesis.tools.Lachesis

6.4 Работа с графичен потребителски интерфейс.

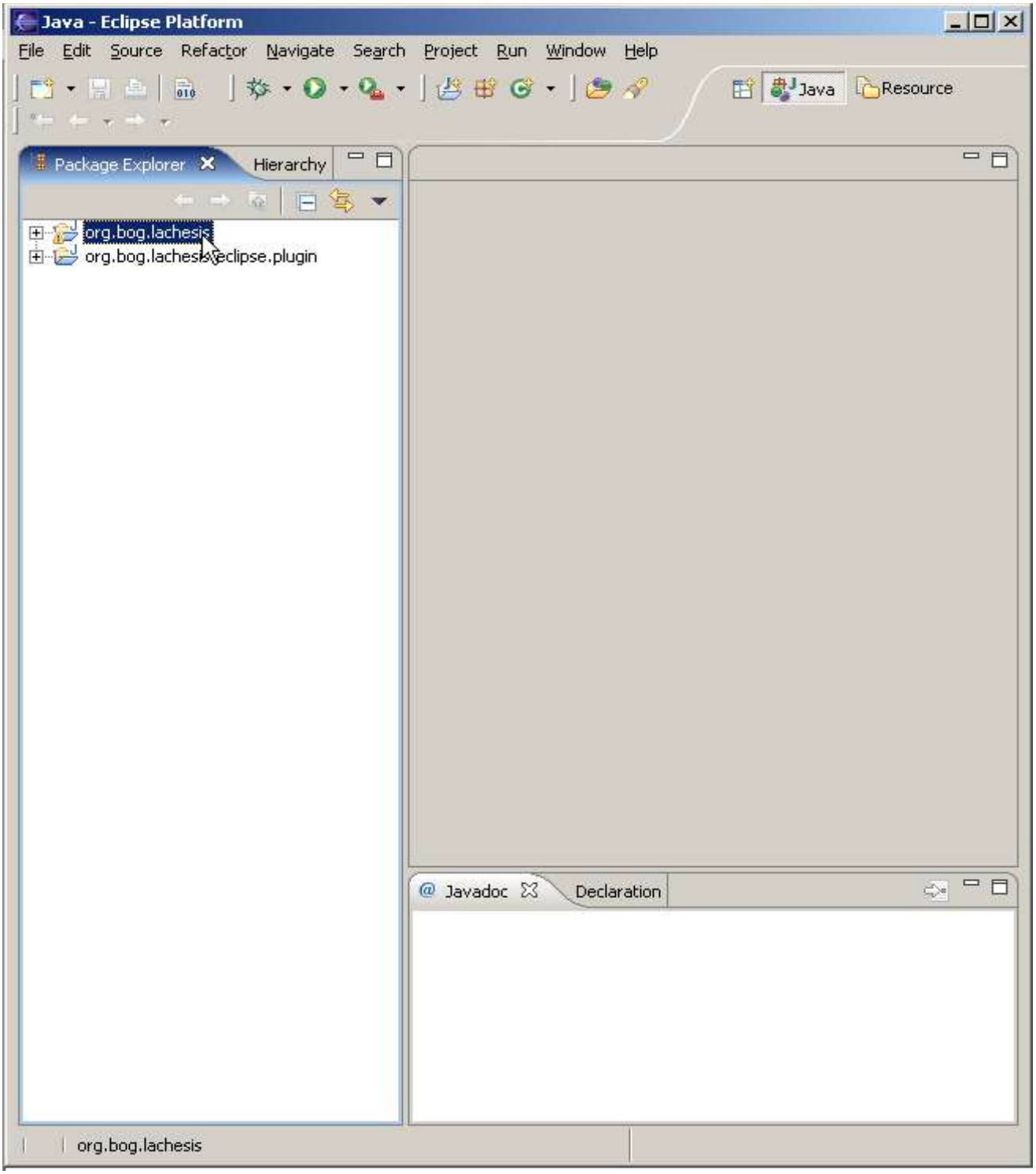
Следва постъпково описание на действията необходими да бъдат извършени от страна на потребителя за извличане на софтуерни метрики.

6.5 Отваряне на Java перспектива.



Фиг. 6.5

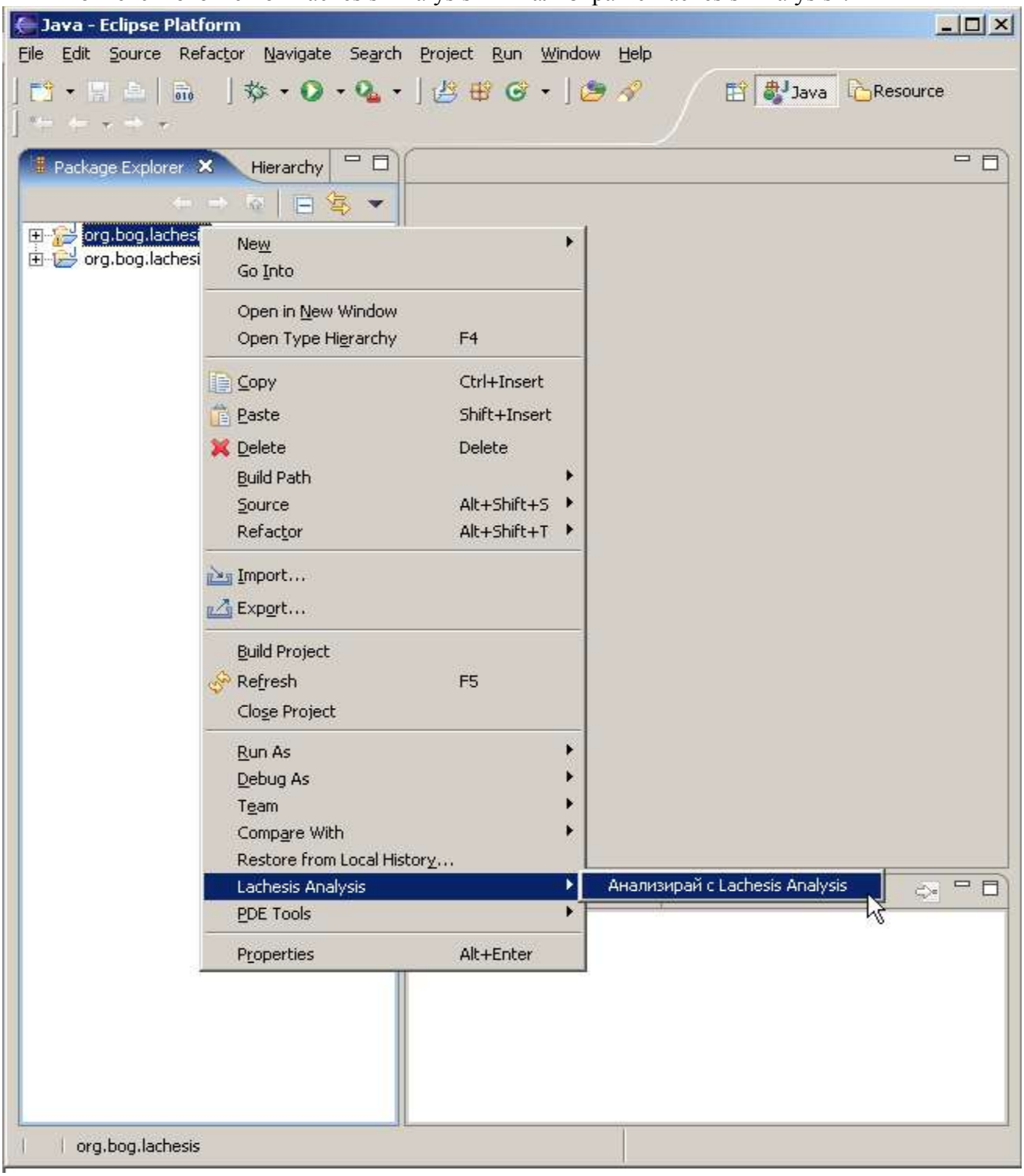
6.6 Избор на проект от изглед PackageExplorer.



Фиг. 6.6

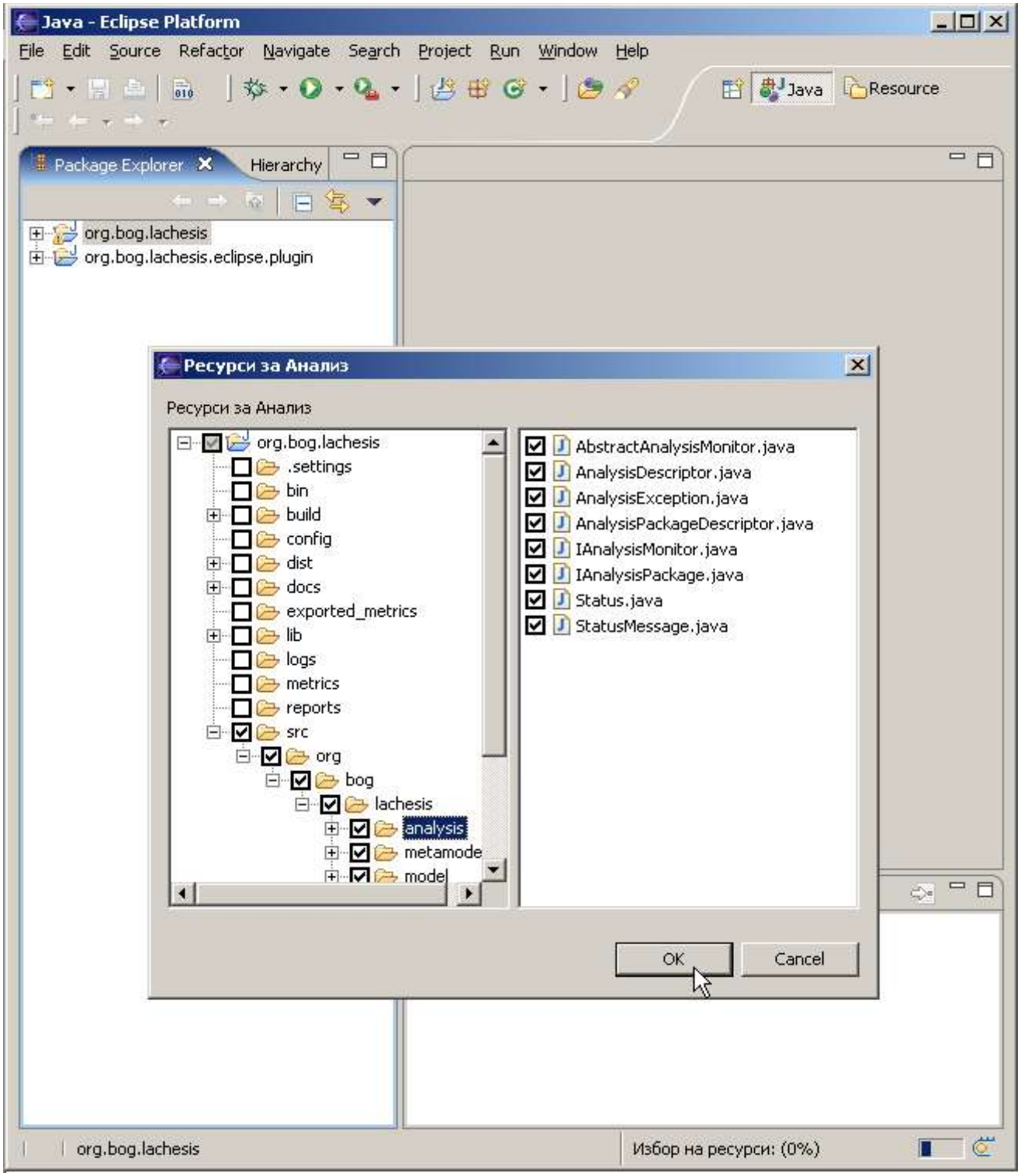
6.7 Стартиране на анализ.

Контекстното меню "Lachesis Analysis-> Анализирай с Lachesis Analysis".



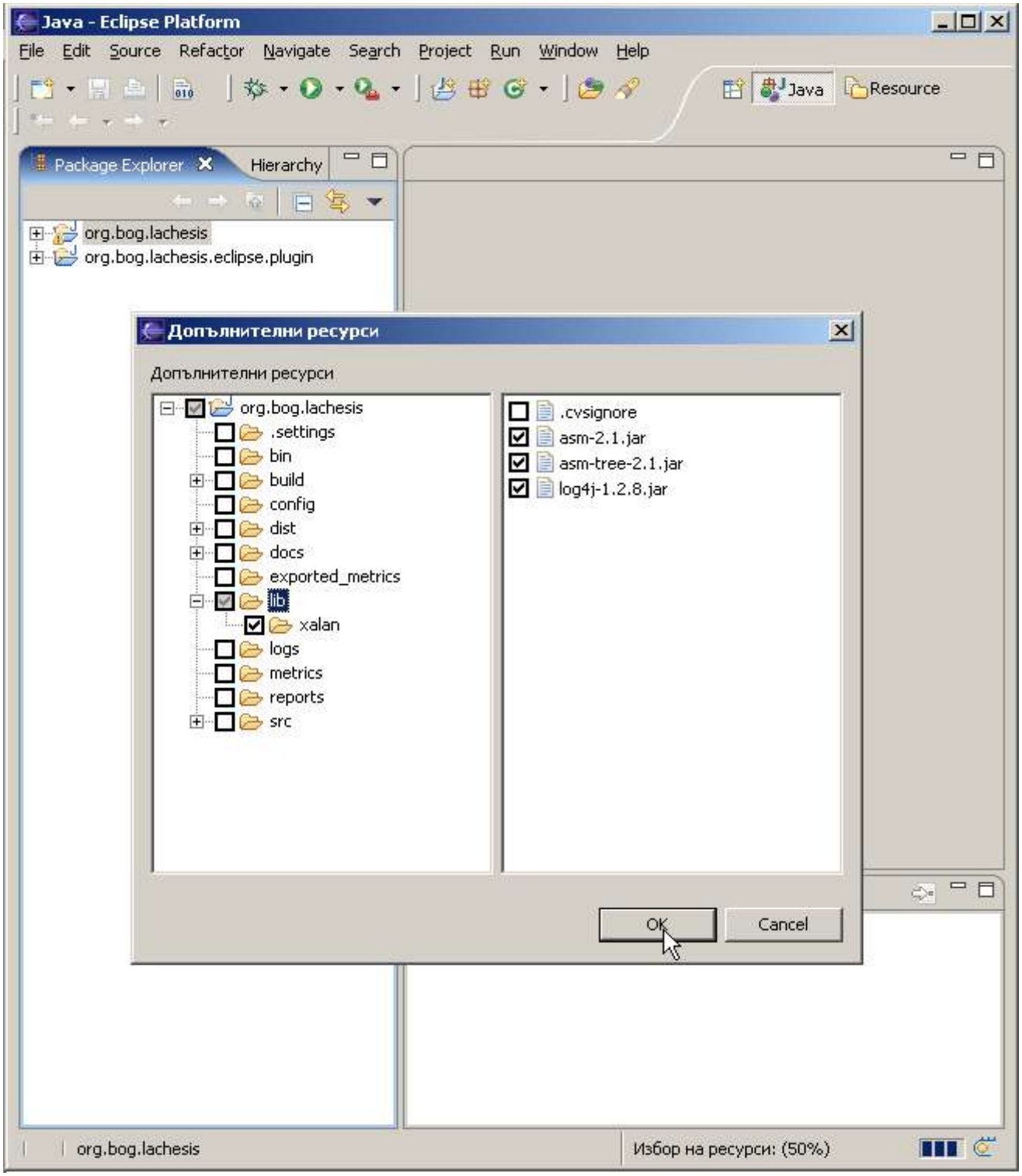
Фиг. 6.7

6.8 Избор на ресурси за анализ.



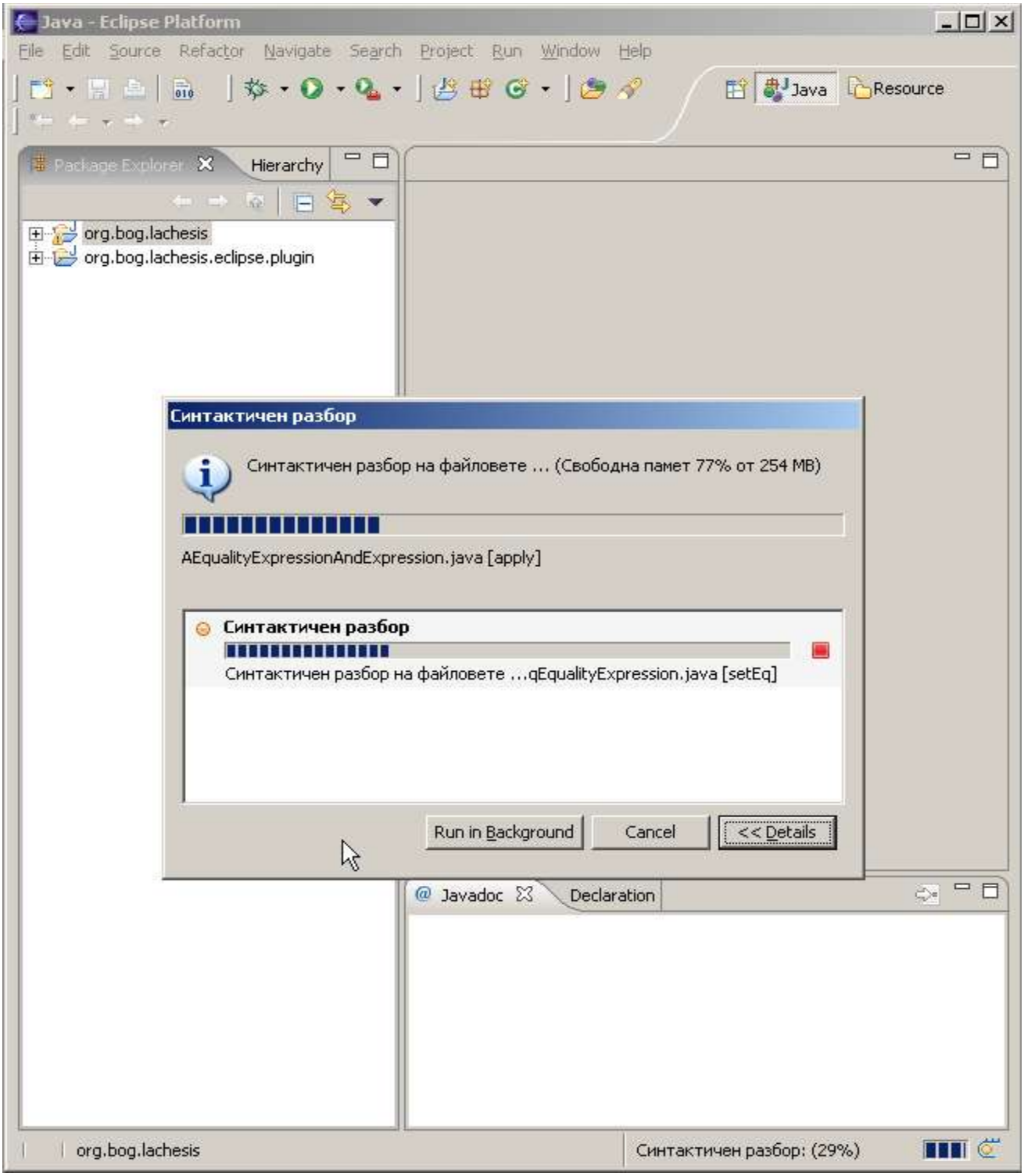
Фиг. 6.8.

6.9 Избор на допълнителни ресурси.



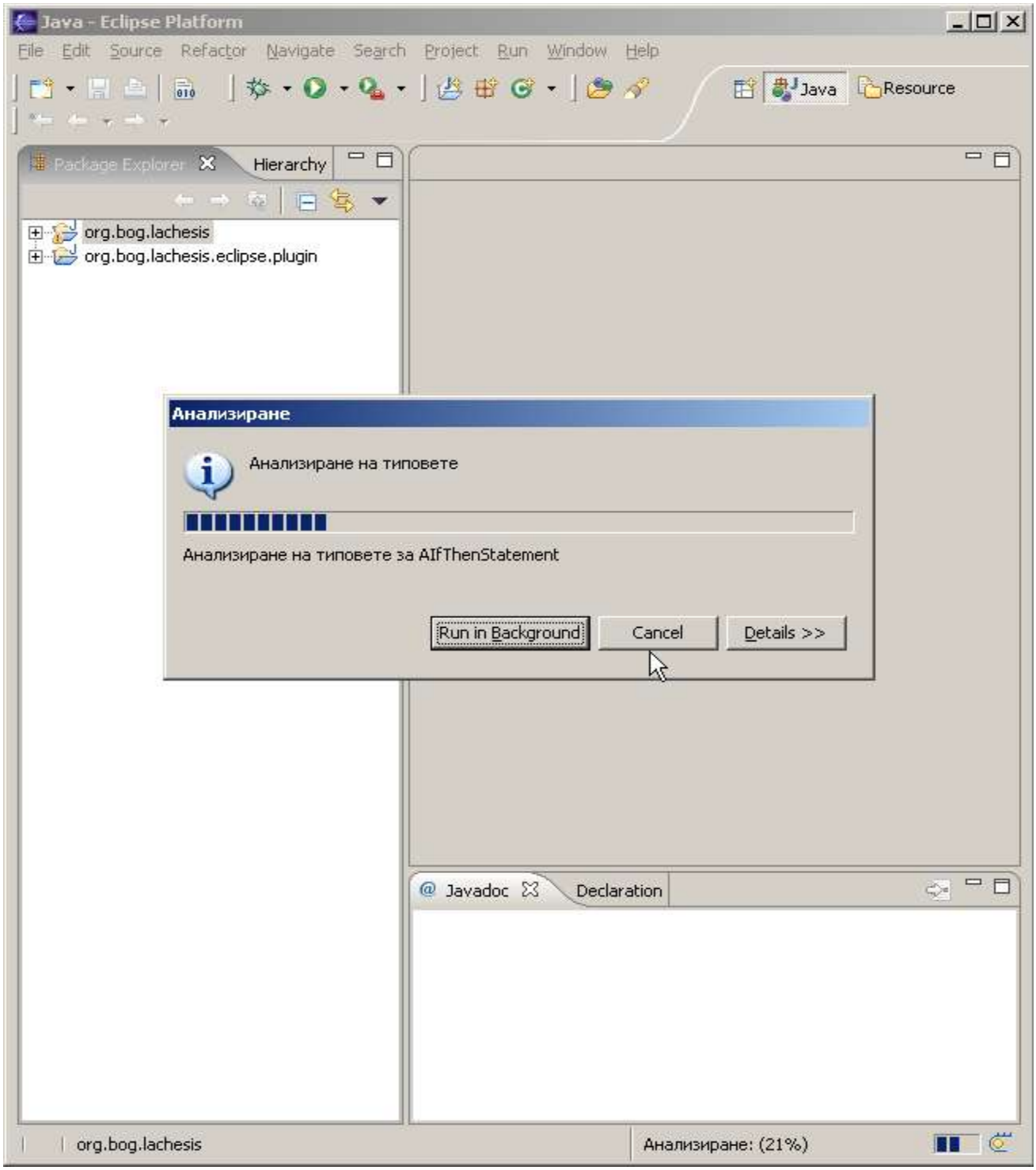
Фиг. 6.9.

6.10 Синтактичен анализ.



Фиг. 6.10.

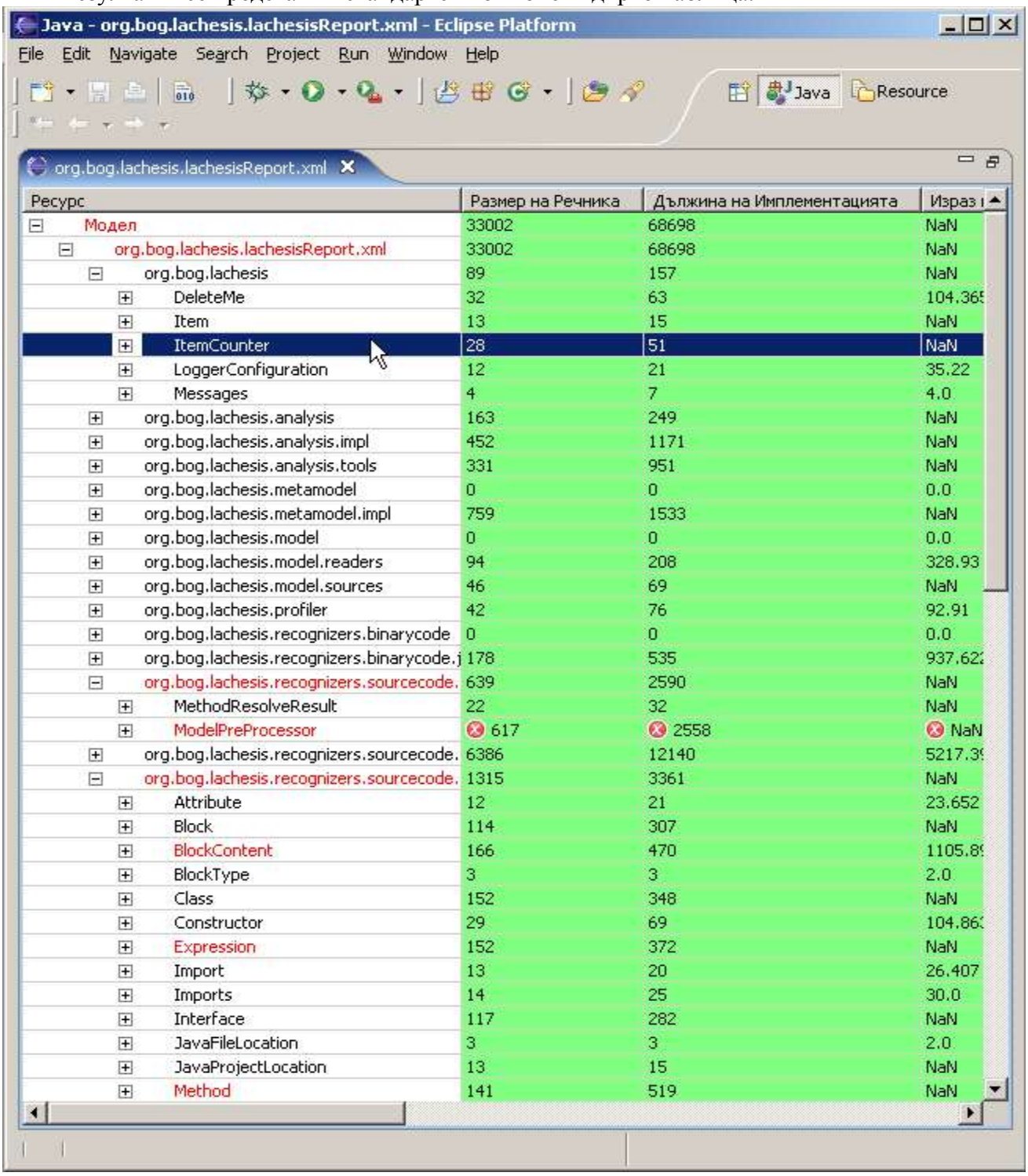
6.11 Семантичен анализ. Анализ на типовете.



Фиг. 6.11.

6.12 Изглед с резултат от анализа.

Резултатът се представя в стандартен компонент дърво-таблица.

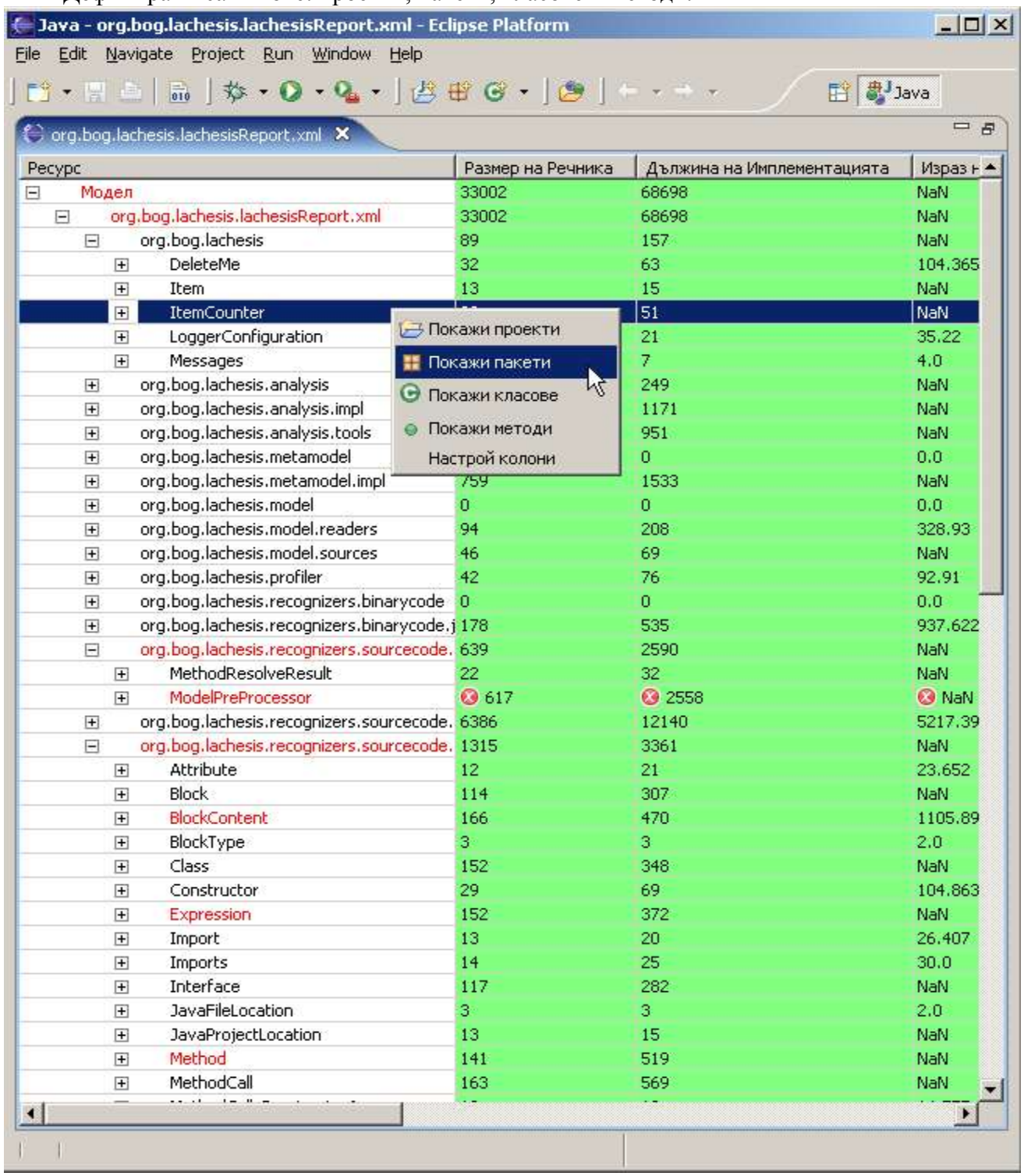


Ресурс	Размер на Речника	Дължина на Имплементацията	Израз
Модел	33002	68698	NaN
org.bog.lachesis.lachesisReport.xml	33002	68698	NaN
org.bog.lachesis	89	157	NaN
DeleteMe	32	63	104.365
Item	13	15	NaN
ItemCounter	28	51	NaN
LoggerConfiguration	12	21	35.22
Messages	4	7	4.0
org.bog.lachesis.analysis	163	249	NaN
org.bog.lachesis.analysis.impl	452	1171	NaN
org.bog.lachesis.analysis.tools	331	951	NaN
org.bog.lachesis.metamodel	0	0	0.0
org.bog.lachesis.metamodel.impl	759	1533	NaN
org.bog.lachesis.model	0	0	0.0
org.bog.lachesis.model.readers	94	208	328.93
org.bog.lachesis.model.sources	46	69	NaN
org.bog.lachesis.profiler	42	76	92.91
org.bog.lachesis.recognizers.binarycode	0	0	0.0
org.bog.lachesis.recognizers.binarycode.j	178	535	937.62
org.bog.lachesis.recognizers.sourcecode.	639	2590	NaN
MethodResolveResult	22	32	NaN
ModelPreProcessor	617	2558	NaN
org.bog.lachesis.recognizers.sourcecode.	6386	12140	5217.3
org.bog.lachesis.recognizers.sourcecode.	1315	3361	NaN
Attribute	12	21	23.652
Block	114	307	NaN
BlockContent	166	470	1105.8
BlockType	3	3	2.0
Class	152	348	NaN
Constructor	29	69	104.86
Expression	152	372	NaN
Import	13	20	26.407
Imports	14	25	30.0
Interface	117	282	NaN
JavaFileLocation	3	3	2.0
JavaProjectLocation	13	15	NaN
Method	141	519	NaN

Фиг. 6.12.

6.13 Автоматично разгръщане на елементи по тип.

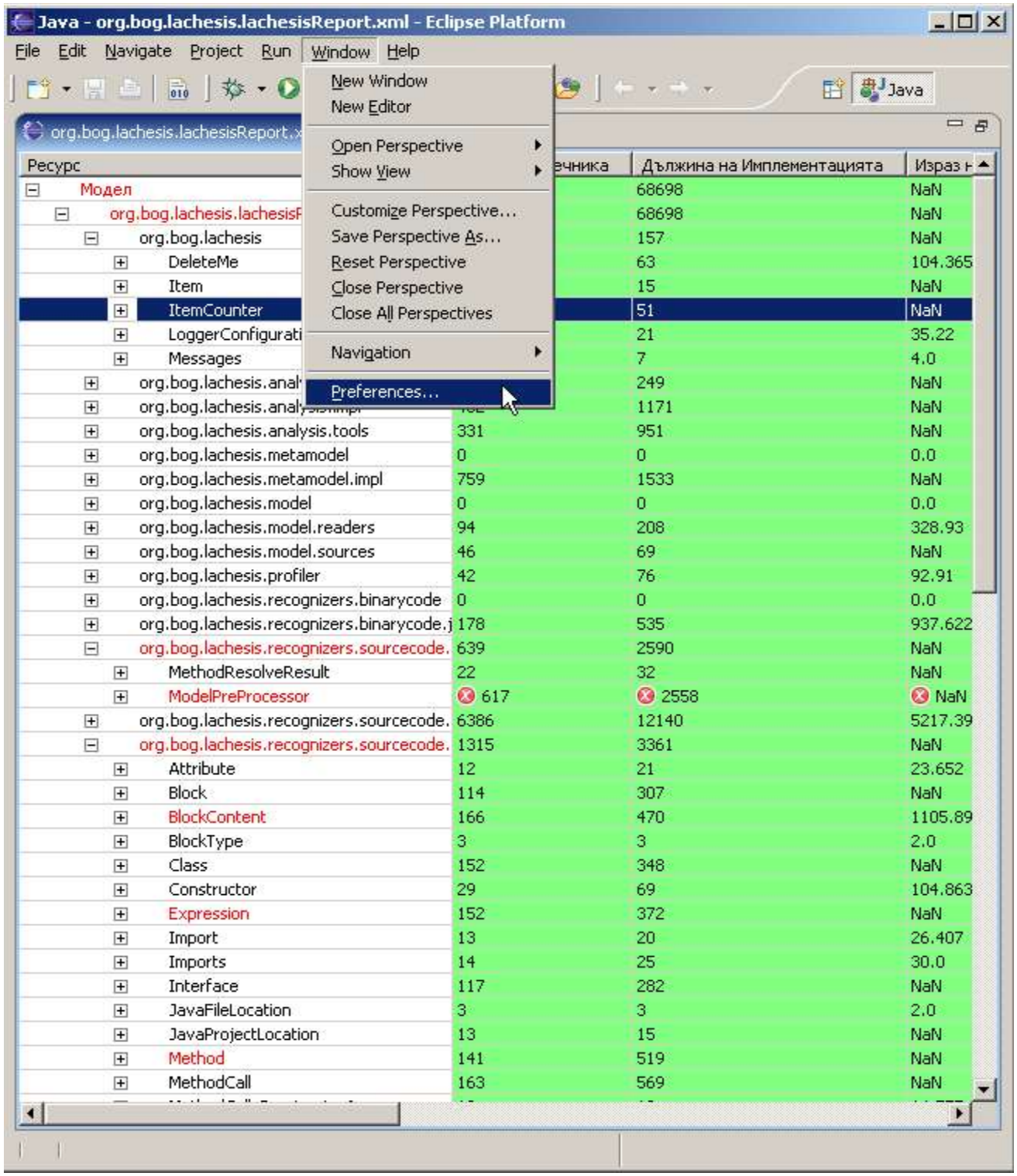
Дефинирани са типове: проекти, пакети, класове и методи.



Ресурс	Размер на Речника	Дължина на Имплементацията	Израз н
Модел	33002	68698	NaN
org.bog.lachesis.lachesisReport.xml	33002	68698	NaN
org.bog.lachesis	89	157	NaN
DeleteMe	32	63	104.365
Item	13	15	NaN
ItemCounter		51	NaN
LoggerConfiguration		21	35.22
Messages		7	4.0
org.bog.lachesis.analysis		249	NaN
org.bog.lachesis.analysis.impl		1171	NaN
org.bog.lachesis.analysis.tools		951	NaN
org.bog.lachesis.metamodel		0	0.0
org.bog.lachesis.metamodel.impl	759	1533	NaN
org.bog.lachesis.model	0	0	0.0
org.bog.lachesis.model.readers	94	208	328.93
org.bog.lachesis.model.sources	46	69	NaN
org.bog.lachesis.profiler	42	76	92.91
org.bog.lachesis.recognizers.binarycode	0	0	0.0
org.bog.lachesis.recognizers.binarycode.j	178	535	937.622
org.bog.lachesis.recognizers.sourcecode.	639	2590	NaN
MethodResolveResult	22	32	NaN
ModelPreProcessor	617	2558	NaN
org.bog.lachesis.recognizers.sourcecode.	6386	12140	5217.39
org.bog.lachesis.recognizers.sourcecode.	1315	3361	NaN
Attribute	12	21	23.652
Block	114	307	NaN
BlockContent	166	470	1105.89
BlockType	3	3	2.0
Class	152	348	NaN
Constructor	29	69	104.863
Expression	152	372	NaN
Import	13	20	26.407
Imports	14	25	30.0
Interface	117	282	NaN
JavaFileLocation	3	3	2.0
JavaProjectLocation	13	15	NaN
Method	141	519	NaN
MethodCall	163	569	NaN

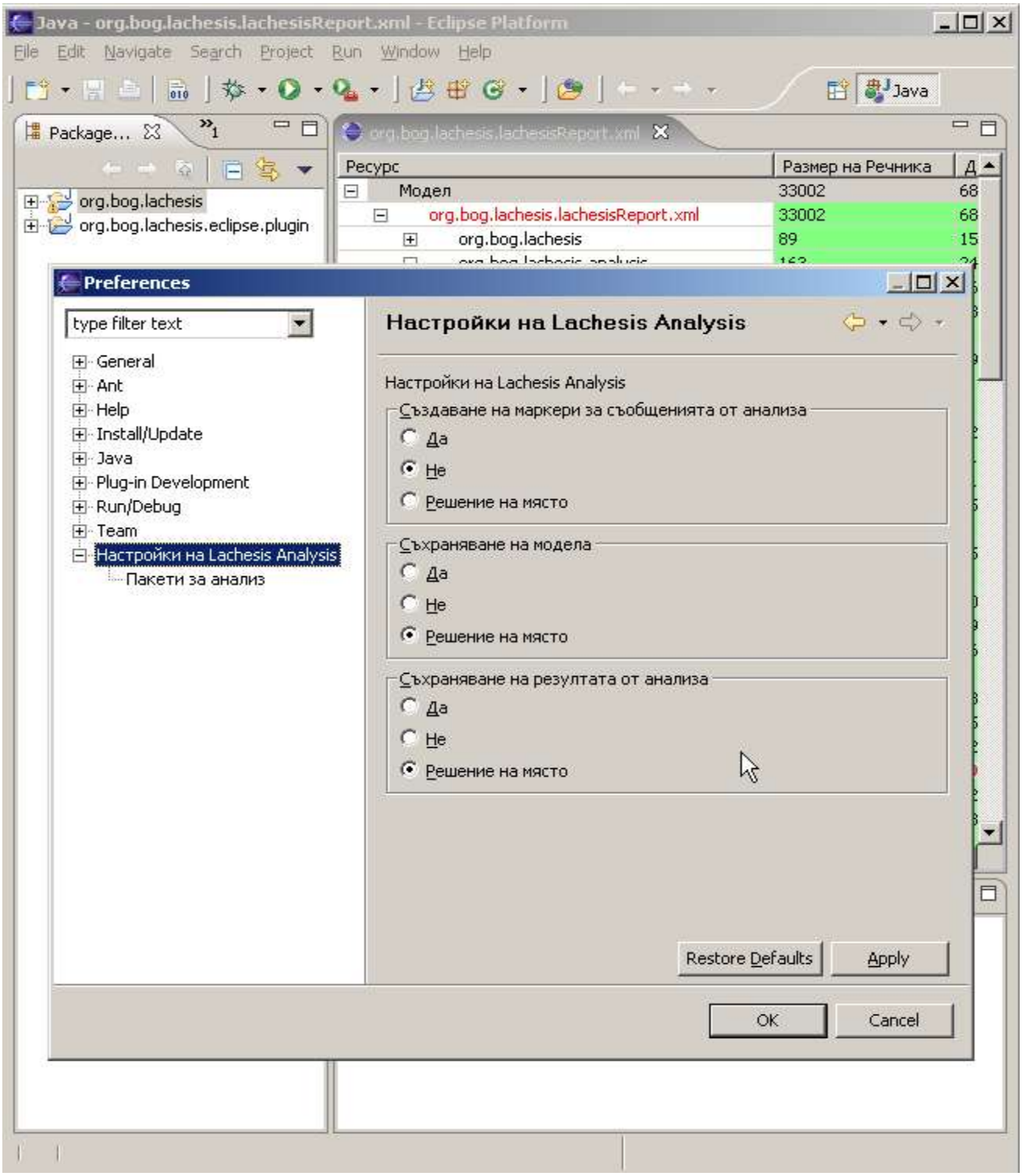
Фиг. 6.13.

6.14 Настройки.



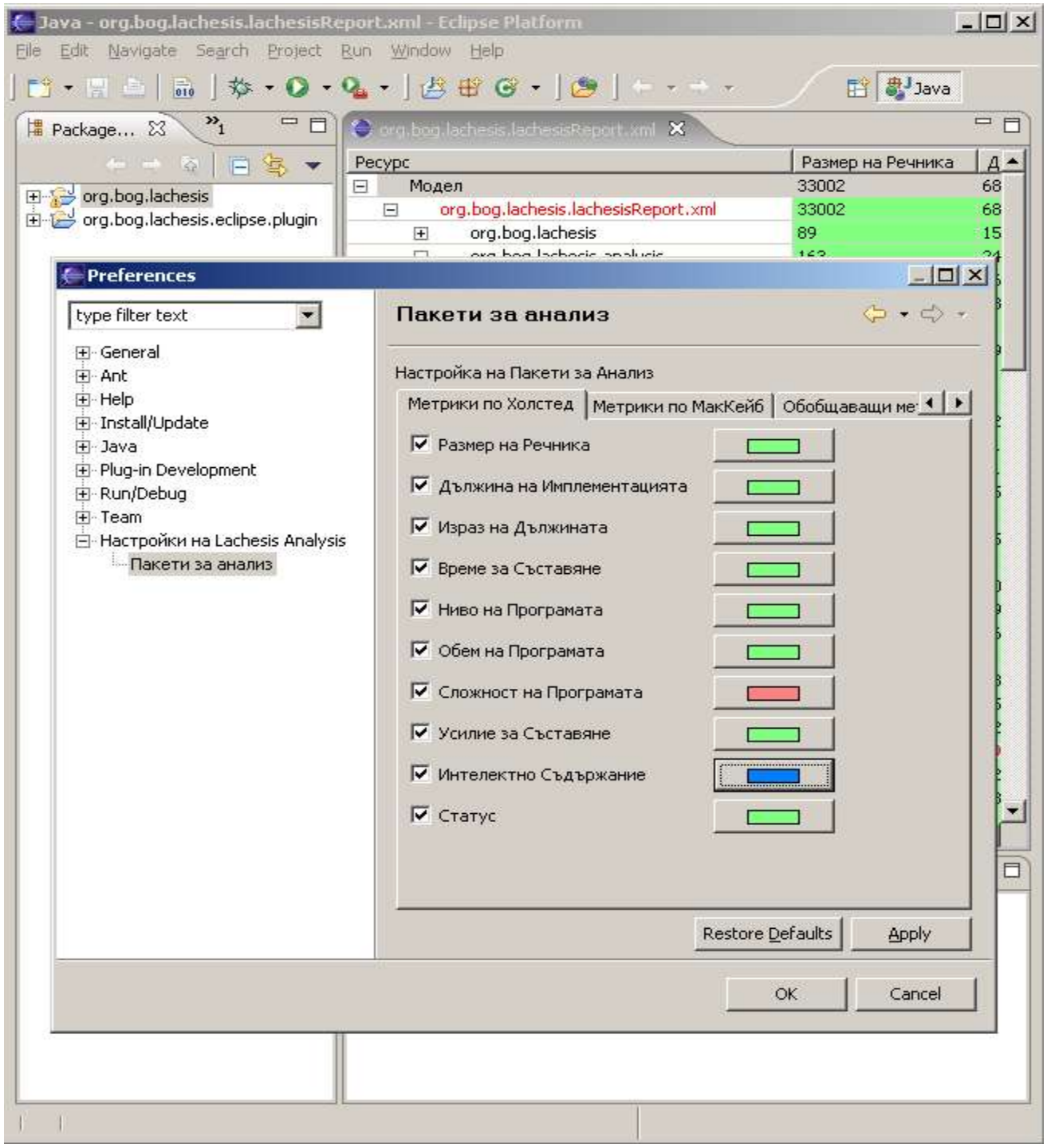
Фиг. 6.14.

6.14.1 Общи настройки.



Фиг. 6.14.1.

6.14.2 Настройки за показване и цвят на метрики.



Фиг. 6.14.2.

7 Заключение.

Предимства:

- Текущата разработка поддържа обширен набор от софтуерни метрики, което не е реализирано от нито един публичен продукт.
- Използвани са изцяло технологии с отворен код.
- Разработката е преносима, тъй-като се базира на изцяло преносими технологии като Java и Eclipse.
- Разработката е публикувана в публичното пространство под LGPL лиценз, което е предпоставка за интензивно развитие и усъвършенстване от страна на други разработчици, както се допринася и за по-разпространена употреба.

Недостатъци:

- Не са използват механизми, които предоставят възможността функционалността на приложението да бъде разширявана без модификацията на кода на основното приложение.
- Съществен проблем, но не на конкретната реализация а въобще, е трудността, а в някои случаи и невъзможността верността на изчисленията на някои метрики да бъдат доказани категорично. Това най-вече се дължи на факта, че някои от метриките реализирани в текущата разработка не са широко разпространени и съответно липсват алтернативни анализатори за тях.

Посоки за усъвършенстване:

- Може да се използва предоставения от инфраструктурата на Eclipse механизъм за разширение на функционалност, което ще премахне един от основните недостатъци.
- Подобрение в производителността в следствие на различни оптимизации може да се извърши, чрез подмяна на използвани библиотеки с техни по-високо продуктивни аналози (ако такива съществуват), както и чрез частични оптимизации на кода на текущата разработка.

8 Литература.

1. [Fowler 04] UML Distilled: A brief guide modeling language, Third Edition. Martin Fowler, 2004.
2. Николов Л., Бонев С., "Формални езици и езикови процесори", ТУ - София, 2005
3. [JLS 05] Java(TM) Language Specification, The (3rd Edition), james Gosling, Bill Joy, Guy Steele, Gilad Bracha. (http://java.sun.com/docs/books/jls/index.html)
4. [Halstead 77] Halstead, Maurice H. Elements of Software Science, Operating, and Programming Systems Series Volume 7. New York, NY: Elsevier, 1977.
5. [McCabe 89] McCabe, Thomas J. & Butler, Charles W. "Design Complexity Measurement and Testing." Communications of the ACM 32, 12 (December 1989): 1415-1425.
6. [McCabe 94] McCabe, Thomas J. & Watson, Arthur H. "Software Complexity." Crosstalk, Journal of Defense Software Engineering 7, 12 (December 1994): 5-9.
7. [Jones 94] Jones, Capers. "Software Metrics: Good, Bad, and Missing." Computer 27, 9 (September 1994): 98-100.
8. [Oman 91] Oman, P. HP-MAS: A Tool for Software Maintainability, Software Engineering (#91-08-TR). Moscow, ID: Test Laboratory, University of Idaho, 1991.
9. [Welker 95] Welker, Kurt D. & Oman, Paul W. "Software Maintainability Metrics Models in Practice." Crosstalk, Journal of Defense Software Engineering 8, 11 (November/December 1995): 19-23.
10. [IEEE 90] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.
11. [Evans 87] Evans, Michael W. & Marciniak, John. Software Quality Assurance and Management. New York, NY: John Wiley & Sons, Inc., 1987.
12. [Oman 94] Oman, P. & Hagemeister, J. "Constructing and Testing of Polynomials Predicting Software Maintainability." Journal of Systems and Software 24, 3 (March 1994): 251-266.
13. [Coleman 94] Coleman, Don, et al. "Using Metrics to Evaluate Software System Maintainability." Computer 27, 8 (August 1994): 44-49.
14. [Coleman 95] Coleman, Don; Lowther, Bruce; & Oman, Paul. "The Application of Software Maintainability Models in Industrial Software Systems." Journal of Systems Software 29, 1 (April 1995): 3-16.
15. [Pearse 95] Pearse, Troy & Oman, Paul. "Maintainability Measurements on Industrial Source Code Maintenance Activities," 295-303. Proceedings. of the International Conference on Software Maintenance. Opio, France, October 17-20, 1995. Los

Alamitos, CA: IEEE Computer Society Press, 1995.

16.[Martin 02] Robert C. Martin, "Agile Software Development: Principles, Patterns, and Practices", Prentice Hall, 2002.

17.[Martin 94] Robert C. Martin, "OO Design Quality Metrics. An Analysis of Dependencies.", 1994.

18.[Chidamber 94] S.R. Chidamber and C.F. Kemerer, "A Metrics suite for object Oriented Design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp. 476-493, June 1994. (doi:10.1109/32.295895 (http://dx.doi.org/10.1109/32.295895)).

19.[Chidamber 98] Shyam R. Chidamber, David P. Darcy, and Chris F. Kemerer. Managerial use of metrics for object-oriented software: An exploratory analysis. IEEE Transactions on Software Engineering, 24(8):629–639, 1998. (doi:10.1109/32.707698 (http://dx.doi.org/10.1109/32.707698)).

20.[Henderson 96] B. Henderson-sellers, "Object-Oriented Metrics Measures of Complexity", Prentice-Hall, 1996.

21.[Briand 98] L.C. Briand, J. Daly and J. Wusr, "A unified framework for cohesion measurement in object-oriented systems", Empirical Software Engineering, 3 (1), pp. 67-117, 1998.

22.[Chidamber 91] S.R. Chidamber and C.F. Kemerer, "Towards a Metrics Suite for Object-Oriented Design, Object-Oriented Programming Systems", Languages and Applications (OOPSLA), Special Issue of SIGPLAN Notices, Vol. 26, No. 10, pp. 197-211, October 1991.

23.[Li 93] W. Li and S. Henry, "Object oriented metrics that predict maintainability", Journal of Systems and Software, Vol. 23, pp. 111-122, February 1993.

24.[Hitz 95] M. Hitz and B. Montazeri, "Measuring coupling and cohesion in object oriented systems", Proceedings of the Int. Symposium on Applied Corporate Computing, pp. 25-27, October 1995.

25.[Bieman 95] J.M. Bieman and B.K. Kang, "Cohesion and reuse in an object-oriented system", Proceedings of the Symposium on Software Reusability (SSR'95), Seattle, WA, pp. 259-262, April 1995.

26.[Bardi 95] L. Badri, M. Badri and S. Ferdenache,"Control Metrics for Object-Oriented Systems", Proceedings of TOOLS (Technology Languages and Systems) Europe'95, Versailles, Prentice-Hall, March 1995.

27.[Glasberg 00] Glasberg, D., El-Emam, K., Memo, W., Madhavji, N., "Validating Object-Oriented Design Metrics on a Commercial Java Application", 44146 NRC/ERB-1080. September 2000.

28.[eclipse.org] The Eclipse Project (http://www.eclipse.org)

29.[antlr] ANTLR, ANother Tool for Language Recognition, (http://www.antlr.org)

30.[javacc] JavaCC, Java Compiler Compiler, (https://javacc.dev.java.net)

31.[sablecc] SableCC, (http://sablecc.org)
32.[bcel] BCEL, Byte Code Engineering Library, (http://jakarta.apache.org/bcel)
33.[asm] ASM, Java bytecode manipulation framework, (http://asm.objectweb.org)
34.[log4j] LOG4J, Logging Services for Java, (http://logging.apache.org/log4j)
35.[xalan-j] Xalan-Java, XSLT processor, (http://xml.apache.org/xalan-j)
36.[w3c] XML, XSL and XSLT tutorials, (http://www.w3schools.com)

9 Приложения.

9.1 Публикуване на проекта в SourceForge

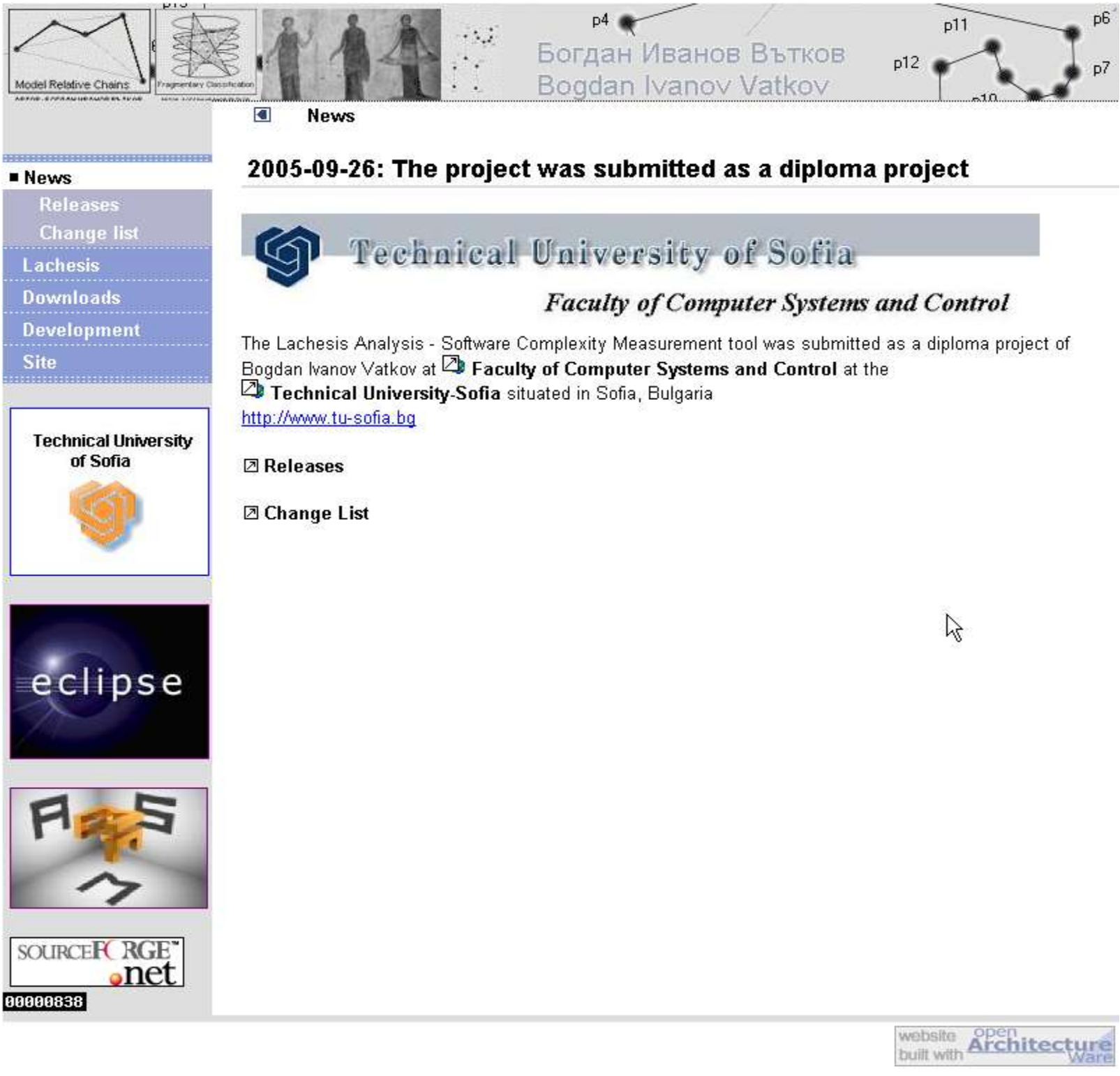
Проектът е публикуван в най-големия доставчик на ресурси за публикуване на проекти с отворен код – http://sourceforge.net.

В предложените от доставчика ресурси се включват:

- Работно пространство във CVS – Concurrent Versionig System.
- Работно пространство и инфраструктура за Интернет страница.
- Работно пространство за публикуване на компилирани версии на проекта.
- Нотификация по електронна поща за ново-публикувани версии на проекта.
- Публична форум система за организиране на дискусии относно проекта.
- Нотификация с електронна поща за новопостъпили коментари във публичния форум.

9.2 Официална страница в Интернет

Адресът на официалната страница на проекта в Интернет: **http://lachesis.sf.net**



9.3 Дефиниция на граматиката на Java програмен език за парсер генератор SableCC.

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * This file is part of J11.                                 *
 * See the file "J11-LICENSE" for Copyright information and the    *
 * terms and conditions for copying, distribution and            *
 * modification of J11.                                      *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

/*******************************************************************
 * Etienne Gagnon: I have built this grammar based to the         *
 * information on pages 19-23 of the "Inner Classes Specification  *
 * document.                                                 *
 * Sun has not released yet an official grammar for Java 1.1 and I *
 * suspect that the definition of Java identifiers has changed and *
 * is different from the Java 1.02 version. This intuition comes   *
 * from noticing the deprecation of Character.isJavaLetter() and   *
 * isJavaLetterOrDigit() methods.                              *
 * In this grammar, I have not changed the lexer. So I scan       *
 * Java 1.02 identifiers. This is the only documented definition   *
 * of identifiers that I have. If somebody has better information, *
 * please let me know.                                         *
 *******************************************************************/

/*
 * Project CKTEST
 * Changes to the original grammar are marked with a [CHANGED] tag.
 *
 */

Package org.bog.lachesis.recognizers.sourcecode.java.sablecc;

/*******************************************************************
 * Helpers                                                   *
 *******************************************************************/
Helpers

  unicode_input_character = [0..0xffff];
  ht  = 0x0009;
  lf  = 0x000a;
  ff  = 0x000c;
  cr  = 0x000d;
  sp  = ' ';

  line_terminator = lf | cr | cr lf;
  input_character = [unicode_input_character - [cr + lf]];

  not_star =    [input_character - '*'] | line_terminator;
  not_star_not_slash = [input_character - ['*' + '/']] | line_terminator;

  unicode_letter =
    [0x0041..0x005a] | [0x0061..0x007a] | [0x00aa..0x00aa] | [0x00b5..0x00b5] |
    [0x00ba..0x00ba] | [0x00c0..0x00d6] | [0x00d8..0x00f6] | [0x00f8..0x01f5] |
    [0x01fa..0x0217] | [0x0250..0x02a8] | [0x02b0..0x02b8] | [0x02bb..0x02c1] |
    [0x02d0..0x02d1] | [0x02e0..0x02e4] | [0x037a..0x037a] | [0x0386..0x0386] |
    [0x0388..0x038a] | [0x038c..0x038c] | [0x038e..0x03a1] | [0x03a3..0x03ce] |
    [0x03d0..0x03d6] | [0x03da..0x03da] | [0x03dc..0x03dc] | [0x03de..0x03de] |
    [0x03e0..0x03e0] | [0x03e2..0x03f3] | [0x0401..0x040c] | [0x040e..0x044f] |
    [0x0451..0x045c] | [0x045e..0x0481] | [0x0490..0x04c4] | [0x04c7..0x04c8] |
    [0x04cb..0x04cc] | [0x04d0..0x04eb] | [0x04ee..0x04f5] | [0x04f8..0x04f9] |
    [0x0531..0x0556] | [0x0559..0x0559] | [0x0561..0x0587] | [0x05d0..0x05ea] |
    [0x05f0..0x05f2] | [0x0621..0x063a] | [0x0640..0x064a] | [0x0671..0x06b7] |
    [0x06ba..0x06be] | [0x06c0..0x06ce] | [0x06d0..0x06d3] | [0x06d5..0x06d5] |
    [0x06e5..0x06e6] | [0x0905..0x0939] | [0x093d..0x093d] | [0x0958..0x0961] |
    [0x0985..0x098c] | [0x098f..0x0990] | [0x0993..0x09a8] | [0x09aa..0x09b0] |
    [0x09b2..0x09b2] | [0x09b6..0x09b9] | [0x09dc..0x09dd] | [0x09df..0x09e1] |
    [0x09f0..0x09f1] | [0x0a05..0x0a0a] | [0x0a0f..0x0a10] | [0x0a13..0x0a28] |
    [0x0a2a..0x0a30] | [0x0a32..0x0a33] | [0x0a35..0x0a36] | [0x0a38..0x0a39] |
    [0x0a59..0x0a5c] | [0x0a5e..0x0a5e] | [0x0a72..0x0a74] | [0x0a85..0x0a8b] |
    [0x0a8d..0x0a8d] | [0x0a8f..0x0a91] | [0x0a93..0x0aa8] | [0x0aaa..0x0ab0] |
    [0x0ab2..0x0ab3] | [0x0ab5..0x0ab9] | [0x0abd..0x0abd] | [0x0ae0..0x0ae0] |
    [0x0b05..0x0b0c] | [0x0b0f..0x0b10] | [0x0b13..0x0b28] | [0x0b2a..0x0b30] |
    [0x0b32..0x0b33] | [0x0b36..0x0b39] | [0x0b3d..0x0b3d] | [0x0b5c..0x0b5d] |
    [0x0b5f..0x0b61] | [0x0b85..0x0b8a] | [0x0b8e..0x0b90] | [0x0b92..0x0b95] |
    [0x0b99..0x0b9a] | [0x0b9c..0x0b9c] | [0x0b9e..0x0b9f] | [0x0ba3..0x0ba4] |
    [0x0ba8..0x0baa] | [0x0bae..0x0bb5] | [0x0bb7..0x0bb9] | [0x0c05..0x0c0c] |
    [0x0c0e..0x0c10] | [0x0c12..0x0c28] | [0x0c2a..0x0c33] | [0x0c35..0x0c39] |
    [0x0c60..0x0c61] | [0x0c85..0x0c8c] | [0x0c8e..0x0c90] | [0x0c92..0x0ca8] |
    [0x0caa..0x0cb3] | [0x0cb5..0x0cb9] | [0x0cde..0x0cde] | [0x0ce0..0x0ce1] |
    [0x0d05..0x0d0c] | [0x0d0e..0x0d10] | [0x0d12..0x0d28] | [0x0d2a..0x0d39] |
    [0x0d60..0x0d61] | [0x0e01..0x0e2e] | [0x0e30..0x0e30] | [0x0e32..0x0e33] |
    [0x0e40..0x0e46] | [0x0e81..0x0e82] | [0x0e84..0x0e84] | [0x0e87..0x0e88] |
    [0x0e8a..0x0e8a] | [0x0e8d..0x0e8d] | [0x0e94..0x0e97] | [0x0e99..0x0e9f] |
    [0x0ea1..0x0ea3] | [0x0ea5..0x0ea5] | [0x0ea7..0x0ea7] | [0x0eaa..0x0eab] |
    [0x0ead..0x0eae] | [0x0eb0..0x0eb0] | [0x0eb2..0x0eb3] | [0x0ebd..0x0ebd] |
    [0x0ec0..0x0ec4] | [0x0ec6..0x0ec6] | [0x0edc..0x0edd] | [0x0f40..0x0f47] |
    [0x0f49..0x0f69] | [0x10a0..0x10c5] | [0x10d0..0x10f6] | [0x1100..0x1159] |
```

```
    [0x115f..0x11a2] | [0x11a8..0x11f9] | [0x1e00..0x1e9b] | [0x1ea0..0x1ef9] |
    [0x1f00..0x1f15] | [0x1f18..0x1f1d] | [0x1f20..0x1f45] | [0x1f48..0x1f4d] |
    [0x1f50..0x1f57] | [0x1f59..0x1f59] | [0x1f5b..0x1f5b] | [0x1f5d..0x1f5d] |
    [0x1f5f..0x1f7d] | [0x1f80..0x1fb4] | [0x1fb6..0x1fbc] | [0x1fbe..0x1fbe] |
    [0x1fc2..0x1fc4] | [0x1fc6..0x1fcc] | [0x1fd0..0x1fd3] | [0x1fd6..0x1fdb] |
    [0x1fe0..0x1fec] | [0x1ff2..0x1ff4] | [0x1ff6..0x1ffc] | [0x207f..0x207f] |
    [0x2102..0x2102] | [0x2107..0x2107] | [0x210a..0x2113] | [0x2115..0x2115] |
    [0x2118..0x211d] | [0x2124..0x2124] | [0x2126..0x2126] | [0x2128..0x2128] |
    [0x212a..0x2131] | [0x2133..0x2138] | [0x3005..0x3005] | [0x3031..0x3035] |
    [0x3041..0x3094] | [0x309b..0x309e] | [0x30a1..0x30fa] | [0x30fc..0x30fe] |
    [0x3105..0x312c] | [0x3131..0x318e] | [0x4e00..0x9fa5] | [0xac00..0xd7a3] |
    [0xf900..0xfa2d] | [0xfb00..0xfb06] | [0xfb13..0xfb17] | [0xfb1f..0xfb28] |
    [0xfb2a..0xfb36] | [0xfb38..0xfb3c] | [0xfb3e..0xfb3e] | [0xfb40..0xfb41] |
    [0xfb43..0xfb44] | [0xfb46..0xfbb1] | [0xfbd3..0xfd3d] | [0xfd50..0xfd8f] |
    [0xfd92..0xfdc7] | [0xfdf0..0xfdfb] | [0xfe70..0xfe72] | [0xfe74..0xfe74] |
    [0xfe76..0xfefc] | [0xff21..0xff3a] | [0xff41..0xff5a] | [0xff66..0xffbe] |
    [0xffc2..0xffc7] | [0xffca..0xffcf] | [0xffd2..0xffd7] | [0xffda..0xffdc];

  unicode_digit =
    [0x0030..0x0039] | [0x0660..0x0669] | [0x06f0..0x06f9] | [0x0966..0x096f] |
    [0x09e6..0x09ef] | [0x0a66..0x0a6f] | [0x0ae6..0x0aef] | [0x0b66..0x0b6f] |
    [0x0be7..0x0bef] | [0x0c66..0x0c6f] | [0x0ce6..0x0cef] | [0x0d66..0x0d6f] |
    [0x0e50..0x0e59] | [0x0ed0..0x0ed9] | [0x0f20..0x0f29] | [0xff10..0xff19];

  java_letter = unicode_letter | '$' | '_';
  java_letter_or_digit = unicode_letter | unicode_digit | '$' | '_';

  non_zero_digit = ['1'..'9'];
  digit = ['0'..'9'];
  hex_digit = ['0'..'9'] | ['a'..'f'] | ['A'..'F'];
  octal_digit = ['0'..'7'];
  zero_to_three = ['0'..'3'];

  decimal_numeral = '0' | non_zero_digit digit*;
  hex_numeral = '0' ('x' | 'X') hex_digit+;
  octal_numeral = '0' octal_digit+;

  integer_type_suffix = 'l' | 'L';

  exponent_part = ('e' | 'E') ('+' | '-')? digit+;

  float_type_suffix = 'f' | 'F' | 'd' | 'D';

  single_character = [input_character - ["' + '\']];
  octal_escape = '\' (octal_digit octal_digit? | zero_to_three octal_digit octal_digit);
  escape_sequence = '\b' | '\t' | '\n' | '\f' | '\r' | '\"' | '\' '' | '\\' | octal_escape;
  string_character = [input_character - ['"' + '\']] | escape_sequence;

/**************************************************************************
 * Tokens                                                                 *
 **************************************************************************/
Tokens

  white_space = (sp | ht | ff | line_terminator)*;

  traditional_comment = '/*' not_star+ '*'+ (not_star_not_slash not_star* '*'+)* '/';
  documentation_comment =    '/**' '*'* (not_star_not_slash not_star* '*'+)* '/';

  /**************************************************************************************
   * Here, we take into account the possibilily that the line terminator might be missing *
   * at the end of the last line of a file. This is imprecise in the Java Language       *
   * Specification, because it is not clear if a line terminator should be added to the  *
   * last line, or not. "javac", the reference compiler, accepts and end-of-line-comment, *
   * even if the line terminator is missing from the last line.                          *
   **************************************************************************************/
  end_of_line_comment = '//' input_character* line_terminator?;

  assert = 'assert';                    /* [CHANGED] Leo: New keyword in JDK 1.4 (07/02/2003) */
  abstract = 'abstract';
  boolean = 'boolean';
  break = 'break';
  byte = 'byte';
  case = 'case';
  catch = 'catch';
  char = 'char';
  class = 'class';
  const = 'const';
  continue = 'continue';
  default = 'default';
  do = 'do';
  double = 'double';
  else = 'else';
  extends = 'extends';
  final = 'final';
  finally = 'finally';
  float = 'float';
  for = 'for';
  goto = 'goto';
  if = 'if';
  implements = 'implements';
```

```
import = 'import';
instanceof = 'instanceof';
int = 'int';
interface = 'interface';
long = 'long';
native = 'native';
new = 'new';
package = 'package';
private = 'private';
protected = 'protected';
public = 'public';
return = 'return';
short = 'short';
static = 'static';
strictfp = 'strictfp';     /* [CHANGED] Rolf: New keyword in java2 */
super = 'super';
switch = 'switch';
synchronized = 'synchronized';
this = 'this';
throw = 'throw';
throws = 'throws';
transient = 'transient';
try = 'try';
void = 'void';
volatile = 'volatile';
while = 'while';

true = 'true';
false = 'false';
null = 'null';

l_parenthese = '(';
r_parenthese = ')';
l_brace = '{';
r_brace = '}';
l_bracket = '[';
r_bracket = ']';
semicolon = ';';
comma = ',';
dot = '.';

assign = '=';
lt = '<';
gt = '>';
complement = '!';
bit_complement = '~';
question = '?';
colon = ':';

eq = '==';
lteq = '<=';
gteq ='>=';
neq = '!=';
and = '&&';
or = '||';
plus_plus = '++';
minus_minus = '--';

plus = '+';
minus = '-';
star = '*';
div = '/';
bit_and = '&';
bit_or = '|';
bit_xor = '^';
mod = '%';
shift_left = '<<';
signed_shift_right = '>>';
unsigned_shift_right = '>>>';

plus_assign = '+=';
minus_assign = '-=';
star_assign = '*=';
div_assign = '/=';
bit_and_assign = '&=';
bit_or_assign = '|=';
bit_xor_assign = '^=';
mod_assign = '%=';
shift_left_assign = '<<=';
signed_shift_right_assign = '>>=';
unsigned_shift_right_assign = '>>>=';

decimal_integer_literal = decimal_numeral integer_type_suffix?;
hex_integer_literal = hex_numeral integer_type_suffix?;
octal_integer_literal = octal_numeral integer_type_suffix?;

floating_point_literal =
    digit+ '.' digit* exponent_part? float_type_suffix? |
    '.' digit+ exponent_part? float_type_suffix? |
    digit+ exponent_part float_type_suffix? |
```

```
      digit+ exponent_part? float_type_suffix;

   character_literal = "'" (single_character | escape_sequence) "'";
   string_literal = '"' string_character* '"';

   identifier = java_letter java_letter_or_digit*;

/*************************************************************************
 * Ignored Tokens                                                        *
 *************************************************************************/
Ignored Tokens

  white_space,
  traditional_comment,
  documentation_comment,
  end_of_line_comment;


/*************************************************************************
 * Productions                                                           *
 *************************************************************************/
Productions

/*************************************************************************
19.2 Grammar from §2.3: The Syntactic Grammar §2.3
*************************************************************************/

goal =
   compilation_unit;

/*************************************************************************
19.3 Grammar from §3: Lexical Structure §3
*************************************************************************/

literal =
   {integer_literal}
      integer_literal |

   {floating_point_literal}
      floating_point_literal |

   {boolean_literal}
      boolean_literal |

   {character_literal}
      character_literal |

   {string_literal}
      string_literal |

   {null_literal}
      null_literal;

/*************************************************************************
19.4 Grammar from §4: Types, Values, and Variables §4
*************************************************************************/

type =
   {primitive_type}
      primitive_type |

   {reference_type}
      reference_type;

primitive_type =
   {numeric_type}
      numeric_type |

   {boolean}
      boolean;

numeric_type =
   {integral_type}
      integral_type |

   {floating_point_type}
      floating_point_type;

integral_type =
   {byte}
      byte |

   {short}
      short |

   {int}
      int |

   {long}
      long |
```

```
    {char}
      char;

floating_point_type =
    {float}
      float |

    {double}
      double;

reference_type =
    {class_or_interface_type}
      class_or_interface_type |

    {array_type}
      array_type;

class_or_interface_type =
    name;

class_type =
    class_or_interface_type;

interface_type =
    class_or_interface_type;

array_type =
    {primitive_type}
      primitive_type dims |

    {name}
      name dims;

/*************************************************************************
19.5 Grammar from §6: Names §6
*************************************************************************/

name =
    {simple_name}
      simple_name |

    {qualified_name}
      qualified_name;

simple_name =
    identifier;

qualified_name =
    name dot identifier;

/*************************************************************************
19.6 Grammar from §7: Packages §7
*************************************************************************/

compilation_unit =
    package_declaration? import_declaration* type_declaration*;

package_declaration =
    package name semicolon;

import_declaration =
    {single_type_import_declaration}
      single_type_import_declaration |

    {type_import_on_demand_declaration}
      type_import_on_demand_declaration;

single_type_import_declaration =
    import name semicolon;

type_import_on_demand_declaration =
    import name dot star semicolon;

type_declaration =
    {class_declaration}
      class_declaration |

    {interface_declaration}
      interface_declaration |

    {semicolon}
      semicolon;

/*************************************************************************
19.7 Productions Used Only in the LALR(1) Grammar
*************************************************************************/

modifier =
    {public}
```

```
    public |

  {protected}
    protected |

  {private}
    private |

  {static}
    static |

  {abstract}
    abstract |

  {final}
    final |

  {native}
    native |

  {synchronized}
    synchronized |

  {transient}
    transient |

  {volatile}
    volatile |

  {strictfp}      /* [CHANGED] Rolf */
    strictfp;

/************************************************************************
19.8.1 Grammar from §8.1: Class Declaration §8.1
************************************************************************/

class_declaration =
  modifier* [t_class]:class identifier P.super? interfaces? class_body;

super =
  extends class_type;

interfaces =
  implements interface_type_list;

interface_type_list =
  {interface_type}
    interface_type |

  {interface_type_list}
    interface_type_list comma interface_type;

class_body =
  l_brace class_body_declaration* r_brace;

class_body_declaration =
  {class_member_declaration}
    class_member_declaration |

  {static_initializer}
    static_initializer |

  {constructor_declaration}
    constructor_declaration |

  {block}
    block;

class_member_declaration =
  {field_declaration}
    field_declaration |

  {method_declaration}
    method_declaration |

  {class_declaration}
    class_declaration |

  {interface_declaration}
    interface_declaration |

  {semicolon}      /* [CHANGED] Rolf: A semicolon may appear here (See JDK sources) */
    semicolon;

/************************************************************************
19.8.2 Grammar from §8.3: Field Declarations §8.3
************************************************************************/

field_declaration =
  modifier* type variable_declarators semicolon;
```

```
variable_declarators =
    {variable_declarator}
      variable_declarator |

    {variable_declarators}
      variable_declarators comma variable_declarator;

variable_declarator =
    {variable_declarator_id}
      variable_declarator_id |

    {assign}
      variable_declarator_id assign variable_initializer;

variable_declarator_id =
    {identifier}
      identifier |

    {variable_declarator_id}
      variable_declarator_id l_bracket r_bracket;

variable_initializer =
    {expression}
      expression |

    {array_initializer}
      array_initializer;

/*********************************************************************
19.8.3 Grammar from §8.4: Method Declarations §8.4
*********************************************************************/

method_declaration =
    method_header method_body;

method_header =
    {type}
      modifier* type method_declarator P.throws? |

    {void}
      modifier* void method_declarator P.throws?;

method_declarator =
    {identifier}
      identifier l_parenthese formal_parameter_list? r_parenthese |

    {method_declarator}
      method_declarator l_bracket r_bracket;

formal_parameter_list =
    {formal_parameter}
      formal_parameter |

    {formal_parameter_list}
      formal_parameter_list comma formal_parameter;

formal_parameter =
    modifier* type variable_declarator_id;

throws =
    T.throws class_type_list;

class_type_list =
    {class_type}
      class_type |

    {class_type_list}
      class_type_list comma class_type;

method_body =
    {block}
      block |

    {semicolon}
      semicolon;

/*********************************************************************
19.8.4 Grammar from §8.5: Static Initializers §8.5
*********************************************************************/

static_initializer =
    static block;

/*********************************************************************
19.8.5 Grammar from §8.6: Constructor Declarations §8.6
*********************************************************************/

constructor_declaration =
    modifier* constructor_declarator P.throws? constructor_body;
```

```
constructor_declarator =
   simple_name l_parenthese formal_parameter_list? r_parenthese;

constructor_body =
   l_brace explicit_constructor_invocation? block_statement* r_brace;

explicit_constructor_invocation =
   {this}
      this l_parenthese argument_list? r_parenthese semicolon |

   {super}
      T.super l_parenthese argument_list? r_parenthese semicolon |

   {qualified}
      primary dot T.super l_parenthese argument_list? r_parenthese semicolon;

/************************************************************************
19.9.1 Grammar from §9.1: Interface Declarations §9.1
************************************************************************/

interface_declaration =
   modifier* interface identifier extends_interfaces? interface_body;

extends_interfaces =
   {extends}
      extends interface_type |

   {extends_interfaces}
      extends_interfaces comma interface_type;

interface_body =
   l_brace interface_member_declaration* r_brace;

interface_member_declaration =
   {constant_declaration}
      constant_declaration |

   {abstract_method_declaration}
      abstract_method_declaration |

   {class_declaration}
      class_declaration |

   {interface_declaration}
      interface_declaration |

   {semicolon}       /* [CHANGED] Rolf: A semicolon may appear here (See JDK sources) */
      semicolon;

constant_declaration =
   field_declaration;

abstract_method_declaration =
   method_header semicolon;

/************************************************************************
19.10 Grammar from §10: Arrays §10
************************************************************************/

array_initializer =
   l_brace variable_initializers? comma? r_brace;

variable_initializers =
   {variable_initializer}
      variable_initializer |

   {variable_initializers}
      variable_initializers comma variable_initializer;

/************************************************************************
19.11 Grammar from §14: Blocks and Statements §14
************************************************************************/

block =
   l_brace block_statement* r_brace;

block_statement =
   {local_variable_declaration_statement}
      local_variable_declaration_statement |

   {statement}
      statement |

   {class_declaration}
      class_declaration;

local_variable_declaration_statement =
   local_variable_declaration semicolon;
```

```
local_variable_declaration =
    modifier* type variable_declarators;

statement =
    {statement_without_trailing_substatement}
        statement_without_trailing_substatement |

    {labeled_statement}
        labeled_statement |

    {if_then_statement}
        if_then_statement |

    {if_then_else_statement}
        if_then_else_statement |

    {while_statement}
        while_statement |

    {for_statement}
        for_statement;

statement_no_short_if =
    {statement_without_trailing_substatement}
        statement_without_trailing_substatement |

    {labeled_statement_no_short_if}
        labeled_statement_no_short_if |

    {if_then_else_statement_no_short_if}
        if_then_else_statement_no_short_if |

    {while_statement_no_short_if}
        while_statement_no_short_if |

    {for_statement_no_short_if}
        for_statement_no_short_if;

statement_without_trailing_substatement =
    {block}
        block |

    {empty_statement}
        empty_statement |

    {expression_statement}
        expression_statement |

    {switch_statement}
        switch_statement |

    {do_statement}
        do_statement |

    {break_statement}
        break_statement |

    {continue_statement}
        continue_statement |

    {return_statement}
        return_statement |

    {synchronized_statement}
        synchronized_statement |

    {throw_statement}
        throw_statement |

    {try_statement}
        try_statement |

/* [CHANGED] Leo: New keyword in JDK 1.4 (07/02/2003) */
    {assert_statement_short}
        assert_statement_short |

    {assert_statement_long}
        assert_statement_long ;

empty_statement =
    semicolon;

labeled_statement =
    identifier colon statement;

labeled_statement_no_short_if =
    identifier colon statement_no_short_if;

expression_statement =
    statement_expression semicolon;
```

```
statement_expression =
    {assignment}
      assignment |

    {pre_increment_expression}
      pre_increment_expression |

    {pre_decrement_expression}
      pre_decrement_expression |

    {post_increment_expression}
      post_increment_expression |

    {post_decrement_expression}
      post_decrement_expression |

    {method_invocation}
      method_invocation |

    {class_instance_creation_expression}
      class_instance_creation_expression;

if_then_statement =
    if l_parenthese expression r_parenthese statement;

if_then_else_statement =
    if l_parenthese expression r_parenthese statement_no_short_if else statement;

if_then_else_statement_no_short_if =
    if l_parenthese expression r_parenthese [statement_no_short_if1]:statement_no_short_if else [statement_no_short_if2]:statement_no_short_if;

switch_statement =
    switch l_parenthese expression r_parenthese switch_block;

switch_block =
    l_brace switch_block_statement_group* switch_label* r_brace;

switch_block_statement_group =
    switch_label+ block_statement+;

switch_label =
    {case}
      case constant_expression colon |

    {default}
      default colon;

while_statement =
    while l_parenthese expression r_parenthese statement;

while_statement_no_short_if =
    while l_parenthese expression r_parenthese statement_no_short_if;

do_statement =
    do statement while l_parenthese expression r_parenthese semicolon;

for_statement =
    for l_parenthese for_init? [semicolon1]:semicolon expression? [semicolon2]:semicolon for_update? r_parenthese statement;

for_statement_no_short_if =
    for l_parenthese for_init? [semicolon1]:semicolon expression? [semicolon2]:semicolon for_update? r_parenthese statement_no_short_if;

for_init =
    {statement_expression_list}
      statement_expression_list |

    {local_variable_declaration}
      local_variable_declaration;

for_update =
    statement_expression_list;

statement_expression_list =
    {statement_expression}
      statement_expression |

    {statement_expression_list}
      statement_expression_list comma statement_expression;

break_statement =
    break identifier? semicolon;

continue_statement =
    continue identifier? semicolon;

return_statement =
    return expression? semicolon;

throw_statement =
```

```
    throw expression semicolon;

synchronized_statement =
    synchronized l_parenthese expression r_parenthese block;

try_statement =
    {try}
      try block catch_clause+ |

    {finally}
      try block catch_clause* P.finally;

catch_clause =
    catch l_parenthese formal_parameter r_parenthese block;

finally =
    T.finally block;

/* [CHANGED] Leo: New keyword in JDK 1.4 (07/02/2003) */
assert_statement_short =
    assert expression semicolon;

assert_statement_long =
    assert [left]:expression colon [right]:expression semicolon;


/*************************************************************************
19.12 Grammar from §15: Expressions §15
*************************************************************************/

primary =
    {primary_no_new_array}
      primary_no_new_array |

    {array_creation_expression}
      array_creation_expression;

primary_no_new_array =
    {literal}
      literal |

    {this}
      this |

    {l_parenthese}
      l_parenthese expression r_parenthese |

    {class_instance_creation_expression}
      class_instance_creation_expression |

    {field_access}
      field_access |

    {method_invocation}
      method_invocation |

    {array_access}
      array_access |

    {qualified_this}
      name dot this |

    {primitive_type}
      primitive_type dims? dot [t_class]:class |

    {named_type}
      name dims? dot [t_class]:class |

    {void}
      void dot [t_class]:class;

class_instance_creation_expression =
    {simple}
      new name l_parenthese argument_list? r_parenthese class_body? |

    {qualified}
      primary dot new identifier l_parenthese argument_list? r_parenthese class_body? |

                                  /* [CHANGED] Leo: Added the following production to enable creation of inner classes */

    {innerclass}
      name dot new identifier l_parenthese argument_list? r_parenthese class_body?;


argument_list =
    {expression}
      expression |

    {argument_list}
      argument_list comma expression;
```

```
array_creation_expression =
    {primitive_type}
      new primitive_type dim_expr+ dims? |

    {class_or_interface_type}
      new class_or_interface_type dim_expr+ dims? |

    {init_primitive}
      new primitive_type dims array_initializer |

    {init_class_interface}
      new class_or_interface_type dims array_initializer;

dim_expr =
   l_bracket expression r_bracket;

dims =
    {l_bracket}
      l_bracket r_bracket |

    {dims}
      dims l_bracket r_bracket;

field_access =
    {primary}
      primary dot identifier |

    {super}
      T.super dot identifier;

method_invocation =
    {name}
      name l_parenthese argument_list? r_parenthese |

    {primary}
      primary dot identifier l_parenthese argument_list? r_parenthese |

    {super}
      T.super dot identifier l_parenthese argument_list? r_parenthese;

array_access =
    {name}
      name l_bracket expression r_bracket |

    {primary_no_new_array}
      primary_no_new_array l_bracket expression r_bracket;

postfix_expression =
    {primary}
      primary |

    {name}
      name |

    {post_increment_expression}
      post_increment_expression |

    {post_decrement_expression}
      post_decrement_expression;

post_increment_expression =
    postfix_expression plus_plus;

post_decrement_expression =
    postfix_expression minus_minus;

unary_expression =
    {pre_increment_expression}
      pre_increment_expression |

    {pre_decrement_expression}
      pre_decrement_expression |

    {plus}
      plus unary_expression |

    {minus}
      minus unary_expression |

    {unary_expression_not_plus_minus}
      unary_expression_not_plus_minus;

pre_increment_expression =
    plus_plus unary_expression;

pre_decrement_expression =
    minus_minus unary_expression;

unary_expression_not_plus_minus =
```

```
    {postfix_expression}
      postfix_expression |

    {bit_complement}
      bit_complement unary_expression |

    {complement}
      complement unary_expression |

    {cast_expression}
      cast_expression;

cast_expression =
    {primitive_type}
      l_parenthese primitive_type dims? r_parenthese unary_expression |

    {expression}
      l_parenthese expression r_parenthese unary_expression_not_plus_minus |

    {name}
      l_parenthese name dims r_parenthese unary_expression_not_plus_minus;

multiplicative_expression =
    {unary_expression}
      unary_expression |

    {star}
      multiplicative_expression star unary_expression |

    {div}
      multiplicative_expression div unary_expression |

    {mod}
      multiplicative_expression mod unary_expression;

additive_expression =
    {multiplicative_expression}
      multiplicative_expression |

    {plus}
      additive_expression plus multiplicative_expression |

    {minus}
      additive_expression minus multiplicative_expression;

shift_expression =
    {additive_expression}
      additive_expression |

    {shift_left}
      shift_expression shift_left additive_expression |

    {signed_shift_right}
      shift_expression signed_shift_right additive_expression |

    {unsigned_shift_right}
      shift_expression unsigned_shift_right additive_expression;

relational_expression =
    {shift_expression}
      shift_expression |

    {lt}
      relational_expression lt shift_expression |

    {gt}
      relational_expression gt shift_expression |

    {lteq}
      relational_expression lteq shift_expression |

    {gteq}
      relational_expression gteq shift_expression |

    {instanceof}
      relational_expression instanceof reference_type;

equality_expression =
    {relational_expression}
      relational_expression |

    {eq}
      equality_expression eq relational_expression |

    {neq}
      equality_expression neq relational_expression;

and_expression =
    {equality_expression}
      equality_expression |
```

```
    {and_expression}
      and_expression bit_and equality_expression;

exclusive_or_expression =
    {and_expression}
      and_expression |

    {exclusive_or_expression}
      exclusive_or_expression bit_xor and_expression;

inclusive_or_expression =
    {exclusive_or_expression}
      exclusive_or_expression |

    {inclusive_or_expression}
      inclusive_or_expression bit_or exclusive_or_expression;

conditional_and_expression =
    {inclusive_or_expression}
      inclusive_or_expression |

    {conditional_and_expression}
      conditional_and_expression and inclusive_or_expression;

conditional_or_expression =
    {conditional_and_expression}
      conditional_and_expression |

    {conditional_or_expression}
      conditional_or_expression or conditional_and_expression;

conditional_expression =
    {conditional_or_expression}
      conditional_or_expression |

    {question}
      conditional_or_expression question expression colon conditional_expression;

assignment_expression =
    {conditional_expression}
      conditional_expression |

    {assignment}
      assignment;

assignment =
   left_hand_side assignment_operator assignment_expression;

left_hand_side =
    {name}
      name |

    {field_access}
      field_access |

    {array_access}
      array_access;

assignment_operator =
    {assign}
      assign |

    {star_assign}
      star_assign |

    {div_assign}
      div_assign |

    {mod_assign}
      mod_assign |

    {plus_assign}
      plus_assign |

    {minus_assign}
      minus_assign |

    {shift_left_assign}
      shift_left_assign |

    {signed_shift_right_assign}
      signed_shift_right_assign |

    {unsigned_shift_right_assign}
      unsigned_shift_right_assign |

    {bit_and_assign}
      bit_and_assign |
```

```
    {bit_xor_assign}
      bit_xor_assign |

    {bit_or_assign}
      bit_or_assign;

expression =
   assignment_expression;

constant_expression =
   expression;

/*********************************************************************
Litterals
*********************************************************************/

boolean_literal =
   {true} true |
   {false} false;

null_literal =
   null;

integer_literal =
   {decimal} decimal_integer_literal |
   {hex} hex_integer_literal |
   {octal} octal_integer_literal;
```

9.4 Разпечатка на първичния програмен код на проекта.

Пакет org.bog.lachesis

```
package org.bog.lachesis;

import java.io.Serializable;

public class Item implements Serializable {
		private String value;
		private int usage = 0;

		public Item(String value) {
			this.value = value;
		}
		public void setUsage(int usage) {
			this.usage = usage;
		}

		public int getUsage() {
			return usage;
		}

		public void incUsage() {
			usage++;
		}
		public void incUsage(int increment) {
			usage+=increment;
		}

		public String getValue() {
			return value;
		}
}


package org.bog.lachesis;

import java.io.Serializable;
import java.util.Hashtable;
import java.util.Iterator;

import org.apache.log4j.Logger;

public class ItemCounter implements Serializable {
		private static Logger log = Logger.getLogger(ItemCounter.class);

		private Hashtable items = new Hashtable();

		public Hashtable getItems() {
			return items;
		}

		public int getUniqueItemsCount() {
			return items.size();
		}
```

```
		public int getUsedItemsCount() {
			int result = 0;
			Iterator iter = items.values().iterator();
			while (iter.hasNext()) {
				result += ((Item) iter.next()).getUsage();
			}
			return result;
		}

		public void addUniqueItem(String key) {
			if (items.get(key) == null)
				items.put(key, new Item(key));
		}

		public void incItemUsage(String key) {
			addUniqueItem(key);
			((Item) items.get(key)).incUsage();
		}

		public int getItemUsage(String key) {
			addUniqueItem(key);
			return ((Item) items.get(key)).getUsage();
		}

		public void incItemUsage(String key, int increment) {
			addUniqueItem(key);
			((Item) items.get(key)).incUsage(increment);
			log.debug("incItemUsage: item = " + key); //$NON-NLS-1$
		}

		public void addItemCounter(ItemCounter counter) {
			for (Iterator iter = counter.getItems().values().iterator(); iter.hasNext();) {
				Item item = (Item) iter.next();
				incItemUsage(item.getValue(), item.getUsage());
			}
		}
}


package org.bog.lachesis;

import java.io.InputStream;
import java.util.Properties;

public class LoggerConfiguration {
		public static void initialize() {
	String config_file = "log4j.properties"; //$NON-NLS-1$
	try {
		InputStream iStream = LoggerConfiguration.class.getResourceAsStream("/org/bog/lachesis/log4j.properties"); //$NON-NLS-1$
		Properties properties = new Properties();
		properties.load(iStream);
	    org.apache.log4j.PropertyConfigurator.configure(properties);
	} catch (Throwable t) {
	    System.err
	            .println("Configuration of LOG4J failed for the following reason:"); //$NON-NLS-1$
	    t.printStackTrace();
	    return;
	}
		}
}

package org.bog.lachesis;

import java.util.MissingResourceException;
import java.util.ResourceBundle;

public class Messages {
		private static final String BUNDLE_NAME = "org.bog.lachesis.messages"; //$NON-NLS-1$

		private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle(BUNDLE_NAME);

		private Messages() {
		}

		public static String getString(String key) {
			try {
				return RESOURCE_BUNDLE.getString(key);
			} catch (MissingResourceException e) {
				return '!' + key + '!';
			}
		}
}
```

Пакет org.bog.lachesis.analysis

```
package org.bog.lachesis.analysis;

import org.bog.lachesis.Messages;

public abstract class AbstractAnalysisMonitor implements IAnalysisMonitor {
```

```
		private String taskName;
		/* (non-Javadoc)
		 * @see org.bog.lachesis.analysis.IAnalysisMonitor#processingResource(java.lang.String, int)
		 */
		public final void processingResource(String message, int workAmount) {
			processingPart(workAmount);
			double totalMemory = Runtime.getRuntime().totalMemory();
			double freeMemory = Runtime.getRuntime().freeMemory();
			double maximumMemory = Runtime.getRuntime().maxMemory();
			String memory = Messages.getString("AbstractAnalysisMonitor.0") + (int)(((maximumMemory+freeMemory-totalMemory)/maximumMemory)
*100.) //$NON-NLS-1$
					+ Messages.getString("AbstractAnalysisMonitor.1")+ (Runtime.getRuntime().maxMemory() / 1048576) + Messages.getString
("AbstractAnalysisMonitor.2"); //$NON-NLS-1$ //$NON-NLS-2$

			setTaskImpl(taskName + memory);
			setSubTask(message);
		}

		/* (non-Javadoc)
		 * @see org.bog.lachesis.analysis.IAnalysisMonitor#processingPart(int)
		 */
		public abstract void processingPart(int worked);
		/* (non-Javadoc)
		 * @see org.bog.lachesis.analysis.IAnalysisMonitor#isCanceled()
		 */
		public abstract boolean isCanceled();

		/* (non-Javadoc)
		 * @see org.bog.lachesis.analysis.IAnalysisMonitor#setBeginTask(java.lang.String, int)
		 */
		public void setBeginTask(String taskName, int count) {
			this.taskName = taskName;
			setBeginTaskImpl(taskName, count);
		}

		protected abstract void setBeginTaskImpl(String taskName, int count);

		/* (non-Javadoc)
		 * @see org.bog.lachesis.analysis.IAnalysisMonitor#setTask(java.lang.String)
		 */
		public void setTask(String taskName) {
			this.taskName = taskName;
			setTaskImpl(taskName);
		}

		protected abstract void setTaskImpl(String taskName);

		/* (non-Javadoc)
		 * @see org.bog.lachesis.analysis.IAnalysisMonitor#setSubTask(java.lang.String)
		 */
		public abstract void setSubTask(String taskName);
}

package org.bog.lachesis.analysis;

public class AnalysisDescriptor {
		private AnalysisPackageDescriptor analysisPackageDescriptor;
    private String name;
    private String description;
    private Double recomendedMinimum;
    private Double recomendedMaximum;


		/**
     * @param name
     * @param description
     * @param recomendedBoundaryDown
     * @param recomendedBoundaryUp
     */
    public AnalysisDescriptor(String name, String description,
          Double recomendedBoundaryDown, Double recomendedBoundaryUp, AnalysisPackageDescriptor analysisPackageDescriptor) {
        this(name,description, analysisPackageDescriptor);
        this.recomendedMinimum = recomendedBoundaryDown;
        this.recomendedMaximum = recomendedBoundaryUp;
    }
    /**
     * @param name
     * @param description
     */
    public AnalysisDescriptor(String name, String description, AnalysisPackageDescriptor analysisPackageDescriptor) {
        super();
        this.name = name;
        this.description = description;
        this.analysisPackageDescriptor = analysisPackageDescriptor;
    }
    /**
     * @return Returns the description.
     */
    public String getDescription() {
```

```
        return description;
    }
    /**
     * @param description The description to set.
     */
    public void setDescription(String description) {
        this.description = description;
    }
    /**
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }
    /**
     * @param name The name to set.
     */
    public void setName(String name) {
        this.name = name;
    }
    /**
     * @return Returns the recomendedBoundaryDown.
     */
    public Double getRecomendedMinimum() {
        return recomendedMinimum;
    }
    /**
     * @param recomendedBoundaryDown The recomendedBoundaryDown to set.
     */
    public void setRecomendedMinimum(Double recomendedMinimum) {
        this.recomendedMinimum = recomendedMinimum;
    }
    /**
     * @return Returns the recomendedBoundaryUp.
     */
    public Double getRecomendedMaximum() {
        return recomendedMaximum;
    }
    /**
     * @param recomendedBoundaryUp The recomendedBoundaryUp to set.
     */
    public void setRecomendedMaximum(Double recomendedMaximum) {
        this.recomendedMaximum = recomendedMaximum;
    }
                public AnalysisPackageDescriptor getAnalysisPackageDescriptor() {
                                return analysisPackageDescriptor;
                }
}

package org.bog.lachesis.analysis;

public class AnalysisException extends Exception {
                private static final long serialVersionUID = 1L;

                public AnalysisException(String message) {
                                super(message);
                }
}


package org.bog.lachesis.analysis;

import java.util.ArrayList;
import java.util.List;

public class AnalysisPackageDescriptor {
    private String name;
    private String description;

    private List analysisDescriptors;

    /**
     * @param name
     * @param description
     */
    public AnalysisPackageDescriptor(String name, String description) {
        super();
        analysisDescriptors = new ArrayList();
        this.name = name;
        this.description = description;
    }
    /**
     * @return Returns the description.
     */
    public String getDescription() {
        return description;
    }
    /**
     * @param description The description to set.
     */
```

```
    public void setDescription(String description) {
        this.description = description;
    }
    /**
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }
    /**
     * @param name The name to set.
     */
    public void setName(String name) {
        this.name = name;
    }
                public List getAnalysisDescriptors() {
                                return analysisDescriptors;
                }
                public void setAnalysisDescriptors(List analysisDescriptors) {
                                this.analysisDescriptors = analysisDescriptors;
                }
}

package org.bog.lachesis.analysis;

public interface IAnalysisMonitor {

                public abstract void processingResource(String message, int workAmount);

                public abstract void processingPart(int worked);

                public abstract boolean isCanceled();

                public abstract void setBeginTask(String taskName, int count);

                public abstract void setTask(String taskName);

                public abstract void setSubTask(String taskName);

                public abstract void setSubTaskAddition(String addition);

}


package org.bog.lachesis.analysis;

import java.util.List;

public interface IAnalysisPackage {
    public List getAnalysisDescrptors();
    public AnalysisPackageDescriptor getAnalysisPackageDescriptor();
    public Object getAnalysisResult(AnalysisDescriptor analysisDescriptor);
    public Status getStatus();
}

package org.bog.lachesis.analysis;

import java.util.List;

public interface IAnalysisPackage {
    public List getAnalysisDescrptors();
    public AnalysisPackageDescriptor getAnalysisPackageDescriptor();
    public Object getAnalysisResult(AnalysisDescriptor analysisDescriptor);
    public Status getStatus();
}


package org.bog.lachesis.analysis;

import org.bog.lachesis.Messages;


public class StatusMessage
{
    public final static int INFO = 0;
    public final static int WARNING = 1;
    public final static int ERROR = 2;
    public final static int FATAL = 3;

    int type = INFO;
    String message;

    /**
     * @return Returns the message.
     */
    public String getMessage()
    {
        return message;
    }
```

```
    /**
     * @return Returns the type.
     */
    public int getType()
    {
        return type;
    }

    /**
     * @param type
     * @param message
     */
    public StatusMessage(int type, String message)
    {
        super();
        this.type = type;
        this.message = message;
    }
    public boolean isInfo() {
                    return (type == INFO);
    }
    public boolean isError() {
                    return ((type == ERROR) || (type == FATAL));
    }
    public boolean isFatal() {
                    return (type == FATAL);
    }
    public boolean isWarning() {
                    return (type == WARNING);
    }
    public String toString() {
                    String typeTitle = Messages.getString("StatusMessage.0"); //$NON-NLS-1$
                    if(type == WARNING)
                                    typeTitle = Messages.getString("StatusMessage.1"); //$NON-NLS-1$
                    else if(type == ERROR)
                                    typeTitle = Messages.getString("StatusMessage.2"); //$NON-NLS-1$
                    else if(type == FATAL)
                                    typeTitle = Messages.getString("StatusMessage.3"); //$NON-NLS-1$
                    return "["+typeTitle+"] "+message;  //$NON-NLS-1$ //$NON-NLS-2$
    }
}
```

Пакет org.bog.lachesis.analysis.impl

```
package org.bog.lachesis.analysis.impl;

import java.util.HashMap;

import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.IAnalysisPackage;

abstract class AbstractAnalysisPackage implements IAnalysisPackage {
                    HashMap analysisResults = new HashMap();

                    public Object getAnalysisResult(AnalysisDescriptor analysisDescriptor) {
                                    return analysisResults.get(analysisDescriptor);
                    }

                    public void setAnalysisResult(AnalysisDescriptor analysisDescriptor, Object result) {
                                    analysisResults.put(analysisDescriptor, result);
                    }
}
```

```
package org.bog.lachesis.analysis.impl;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.ItemCounter;
import org.bog.lachesis.Messages;
import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.Status;
import org.bog.lachesis.metamodel.IAttribute;
import org.bog.lachesis.metamodel.IReference;
import org.bog.lachesis.metamodel.IClass;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IMethodCall;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IType;
import org.bog.lachesis.metamodel.IVariable;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor;


public class ChidamberKemererMetrics extends AbstractAnalysisPackage {
                    public final static AnalysisPackageDescriptor PACKAGE_DESCRIPTOR = new AnalysisPackageDescriptor(Messages
                                                    .getString("ChidamberKemererMetrics.0"), Messages.getString("ChidamberKemererMetrics.1")); //$NON-NLS-1$ //$NON-NLS-2$
```

```
	private static Logger log = Logger.getLogger(ChidamberKemererMetrics.class);

	private Status status = new Status();

	private int weightedMethodsPerClass; // WMC  > 20 - Wraning, > 35 - Error

	private int depthOfInheritanceTree; // DIT

	private int numberOfChildren; // NOC

	private int couplingBetweenObjectClasses; // CBO

	private int responseForAClass; // RFC

	private double lackOfCohesionOfMethods2; // LCOM2

	private double lackOfCohesionOfMethods3; // LCOM3

	private static AnalysisDescriptor WMC = new AnalysisDescriptor(
						Messages.getString("ChidamberKemererMetrics.2"), Messages.getString("ChidamberKemererMetrics.3"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor DIT = new AnalysisDescriptor(
						Messages.getString("ChidamberKemererMetrics.4"), Messages.getString("ChidamberKemererMetrics.5"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor NOC = new AnalysisDescriptor(
						Messages.getString("ChidamberKemererMetrics.6"), Messages.getString("ChidamberKemererMetrics.7"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor CBO = new AnalysisDescriptor(
						Messages.getString("ChidamberKemererMetrics.8"), Messages.getString("ChidamberKemererMetrics.9"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor RFC = new AnalysisDescriptor(
						Messages.getString("ChidamberKemererMetrics.10"), Messages.getString("ChidamberKemererMetrics.11"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor LCOM2 = new AnalysisDescriptor(
						Messages.getString("ChidamberKemererMetrics.12"), Messages.getString("ChidamberKemererMetrics.13"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor LCOM3 = new AnalysisDescriptor(
						Messages.getString("LCOM3.Henderson-Sellers"), Messages.getString("LCOM3 Henderson-Sellers"), PACKAGE_DESCRIPTOR);

	private static AnalysisDescriptor STATUS_DESCRIPTOR = new AnalysisDescriptor(
						Messages.getString("ChidamberKemererMetrics.14"), Messages.getString("ChidamberKemererMetrics.15"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	static {
			List analysisDescriptors = PACKAGE_DESCRIPTOR.getAnalysisDescriptors();

			analysisDescriptors.add(WMC);
			analysisDescriptors.add(DIT);
			analysisDescriptors.add(CBO);
			analysisDescriptors.add(NOC);
			analysisDescriptors.add(RFC);
			analysisDescriptors.add(LCOM2);
			analysisDescriptors.add(LCOM3);
			analysisDescriptors.add(STATUS_DESCRIPTOR);
	}

	/*
	 * (non-Javadoc)
	 *
	 * @see org.bog.lachesis.metrics.IMetricBundle#getMetricDescrptors()
	 */
	public List getAnalysisDescrptors() {
			return PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
	}

	/*
	 * (non-Javadoc)
	 *
	 * @see org.bog.lachesis.metrics.IMetricBundle#getMetric(org.bog.lachesis.metrics.MetricDescriptor)
	 */
	private void populateAnalysisResults() {
			setAnalysisResult(WMC, new Integer(getWeightedMethodsPerClass()));
			setAnalysisResult(DIT, new Integer(getDepthOfInheritanceTree()));
			setAnalysisResult(CBO, new Integer(getCouplingBetweenObjectClasses()));
			setAnalysisResult(NOC, new Integer(getNumberOfChildren()));
			setAnalysisResult(RFC, new Integer(getResponseForAClass()));
			setAnalysisResult(LCOM2, new Double(getLackOfCohesionOfMethods2()));
			setAnalysisResult(LCOM3, new Double(getLackOfCohesionOfMethods3()));
			setAnalysisResult(STATUS_DESCRIPTOR, status.toString());
	}

	/*
	 * (non-Javadoc)
```

```
	 *
	 * @see org.bog.lachesis.metrics.IMetricBundle#getMessages()
	 */
	public Status getStatus() {
		return status;
	}

	public static IProgramUnit getParentProgramUnit(IProgramUnit programUnit) {
		IProgramUnit parent = null;
		if (programUnit instanceof IClass) {
			IClass aclass = (IClass) programUnit;
			for (Iterator iter = aclass.getSuperClasses().values().iterator(); iter.hasNext();) {
				parent = (IProgramUnit) iter.next();
				if (parent != programUnit)
					return parent;
			}
		}
		return null;
	}

	private static List getDirectChildren(IProgramUnit programUnit, ModelPreProcessor modelPreProcessor, Status status) throws Exception {
		List result = new ArrayList();

		Iterator it = modelPreProcessor.getProgramUnits().valuesSubIterator();
		while (it.hasNext()) {
			IProgramUnit unit = (IProgramUnit) it.next();
			IProgramUnit parent = getParentProgramUnit(unit);
			if ((parent != null) && (parent.equals(programUnit)))
				result.add(unit);
		}
		log.debug("getDirectChildren for " + programUnit.getName() + " is " + result.size()); //$NON-NLS-1$ //$NON-NLS-2$
		return result;
	}

	private static int getPathToRoot(IProgramUnit programUnit, ModelPreProcessor modelTools, Status status) throws Exception {
		IProgramUnit temp = getParentProgramUnit(programUnit);

		if (temp != null) {
			return 1 + getPathToRoot(temp, modelTools, status);
		}
		return 0;
	}

	public static void analyzeUniqueType(IType type, HashMap uniqueTypes, IClass hostClass) {
		if (type.getProgramUnit() == null)
			uniqueTypes.put(type.getName(), type);
		else if (type.getProgramUnit() != hostClass)
			uniqueTypes.put(type.getProgramUnit(), type);
	}

	public static ChidamberKemererMetrics analyze(IClass theClass, ModelPreProcessor modelTools) {
		ChidamberKemererMetrics result = new ChidamberKemererMetrics();
		result.weightedMethodsPerClass = theClass.getMethods().size();
		try {
			//
			// Analyze DIT
			//
			result.depthOfInheritanceTree = getPathToRoot(theClass, modelTools, result.getStatus());
			//
			// Analyze NOC
			//
			result.numberOfChildren = getDirectChildren(theClass, modelTools, result.getStatus()).size();
		} catch (Exception e) {
			try {
				result.status.addWarn(Messages.getString("ChidamberKemererMetrics.18") + theClass.getFullyQualifiedName()); //
$NON-NLS-1$
				e.printStackTrace();
			} catch (Exception e1) {
				result.status.addWarn(Messages.getString("ChidamberKemererMetrics.18") + theClass.getName()); //$NON-NLS-1$
				e1.printStackTrace();
			}
		}

		//
		// Analyze RFC and CBO
		// - fix required: should ignore types mathing supers of the current
		// class
		// - fix required: should ignore types located in the platform
		// (java.lang?)

		HashMap uniqueRemoteMethods = new HashMap();
		HashMap uniqueReferredTypes = new HashMap();

		for (Iterator iter = theClass.getMethods().iterator(); iter.hasNext();) {
			IMethod method = (IMethod) iter.next();

			// count unique types for local variables
			List variables = method.getBlock().getAllCode(IVariable.class);
			for (Iterator iterator = variables.iterator(); iterator.hasNext();)
				analyzeUniqueType(((IVariable) iterator.next()).getType(), uniqueReferredTypes, theClass);
```

```
                                        // count unique types for formal parameters
                                        for (Iterator iterator = method.getParameters().iterator(); iterator.hasNext();)
                                                analyzeUniqueType(((IVariable) iterator.next()).getType(), uniqueReferredTypes, theClass);

                                        // count unique type for return type
                                        analyzeUniqueType(method.getReturningType(), uniqueReferredTypes, theClass);

                                        // count method calls
                                        List methodCalls = method.getBlock().getAllCode(IMethodCall.class);
                                        for (Iterator iterator = methodCalls.iterator(); iterator.hasNext();) {
                                                IMethodCall methodCall = (IMethodCall) iterator.next();
                                                if (methodCall.getTargetMethod() == null)
                                                        uniqueRemoteMethods.put(methodCall.getName(), methodCall);
                                                else if (methodCall.getTargetMethod().getEnclosingProgramUnit() != theClass)
                                                        uniqueRemoteMethods.put(methodCall.getTargetMethod(), methodCall);
                                        }
                                }
                                result.responseForAClass = theClass.getMethods().size() + uniqueRemoteMethods.size();
                                result.couplingBetweenObjectClasses = uniqueReferredTypes.size();

                                //
                                // Analyze LCOM
                                //
                                if (theClass.getName().equals("AbstractAnalysisPackage"))
                                        System.out.println("debug here 1");
                                double m = theClass.getMethods().size();
                                double a = theClass.getAttributes().size();
                                ItemCounter attributeReferrers = new ItemCounter();

                                for (Iterator iter = theClass.getMethods().iterator(); iter.hasNext();) {
                                        IMethod method = (IMethod) iter.next();

                                        HashMap uniqueAttributeAccesses = new HashMap();
                                        // count attribute accesses
                                        List attributeAccesses = method.getBlock().getAllCode(IReference.class);
                                        for (Iterator iterator = attributeAccesses.iterator(); iterator.hasNext();) {
                                                IReference attributeAccess = (IReference) iterator.next();
                                                // count only resolved attribute accesses
                                                if (attributeAccess.getTargetAttribute() != null)
                                                        uniqueAttributeAccesses.put(attributeAccess.getTargetAttribute(), attributeAccess.getTargetAttribute
()); 
                                        }
                                        for (Iterator iterator = uniqueAttributeAccesses.values().iterator(); iterator.hasNext();) {
                                                IAttribute attribute = (IAttribute) iterator.next();
                                                attributeReferrers.incItemUsage(attribute.getName());
                                        }
                                }
                                int sumMA = 0;
                                for (Iterator iter = theClass.getAttributes().keySet().iterator(); iter.hasNext();) {
                                        String attributName = (String) iter.next();
                                        sumMA += attributeReferrers.getItemUsage(attributName);
                                }
                                if ((a != 0) && (m != 1)) {
                                        result.lackOfCohesionOfMethods2 = (double) (1 - sumMA/(m*a))*100.;
                                        result.lackOfCohesionOfMethods3 = (double) (m - sumMA / a) / (m - 1);
                                }

                                result.populateAnalysisResults();
                                return result;
                }

                /**
                 * @return Returns the couplingBetweenObjectClasses.
                 */
                public int getCouplingBetweenObjectClasses() {
                        return couplingBetweenObjectClasses;
                }

                /**
                 * @param couplingBetweenObjectClasses
                 *            The couplingBetweenObjectClasses to set.
                 */
                public void setCouplingBetweenObjectClasses(int couplingBetweenObjectClasses) {
                        this.couplingBetweenObjectClasses = couplingBetweenObjectClasses;
                }

                /**
                 * @return Returns the depthOfInheritanceTree.
                 */
                public int getDepthOfInheritanceTree() {
                        return depthOfInheritanceTree;
                }

                /**
                 * @param depthOfInheritanceTree
                 *            The depthOfInheritanceTree to set.
                 */
                public void setDepthOfInheritanceTree(int depthOfInheritanceTree) {
                        this.depthOfInheritanceTree = depthOfInheritanceTree;
```

```
		}

		/**
		 * @return Returns the lackOfCohesionOfMethods.
		 */
		public double getLackOfCohesionOfMethods2() {
			return lackOfCohesionOfMethods2;
		}
		public double getLackOfCohesionOfMethods3() {
			return lackOfCohesionOfMethods3;
		}

		/**
		 * @param lackOfCohesionOfMethods
		 *            The lackOfCohesionOfMethods to set.
		 */
		public void setLackOfCohesionOfMethods2(int lackOfCohesionOfMethods2) {
			this.lackOfCohesionOfMethods2 = lackOfCohesionOfMethods2;
		}
		public void setLackOfCohesionOfMethods3(int lackOfCohesionOfMethods3) {
			this.lackOfCohesionOfMethods3 = lackOfCohesionOfMethods3;
		}

		/**
		 * @return Returns the numberOfChildren.
		 */
		public int getNumberOfChildren() {
			return numberOfChildren;
		}

		/**
		 * @param numberOfChildren
		 *            The numberOfChildren to set.
		 */
		public void setNumberOfChildren(int numberOfChildren) {
			this.numberOfChildren = numberOfChildren;
		}

		/**
		 * @return Returns the responseForAClass.
		 */
		public int getResponseForAClass() {
			return responseForAClass;
		}

		/**
		 * @param responseForAClass
		 *            The responseForAClass to set.
		 */
		public void setResponseForAClass(int responseForAClass) {
			this.responseForAClass = responseForAClass;
		}

		/**
		 * @return Returns the weightedMethodsPerClass.
		 */
		public int getWeightedMethodsPerClass() {
			return weightedMethodsPerClass;
		}

		/**
		 * @param weightedMethodsPerClass
		 *            The weightedMethodsPerClass to set.
		 */
		public void setWeightedMethodsPerClass(int weightedMethodsPerClass) {
			this.weightedMethodsPerClass = weightedMethodsPerClass;
		}

		public AnalysisPackageDescriptor getAnalysisPackageDescriptor() {
			return PACKAGE_DESCRIPTOR;
		}
}

package org.bog.lachesis.analysis.impl;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.Messages;
import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.Status;
import org.bog.lachesis.analysis.tools.AnalysisRecord;
import org.bog.lachesis.metamodel.IImport;
import org.bog.lachesis.metamodel.IImports;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor;
```

```
public class DependenciesAnalysis extends AbstractAnalysisPackage {
	public final static AnalysisPackageDescriptor PACKAGE_DESCRIPTOR = new AnalysisPackageDescriptor(Messages.getString("DependenciesAnalysis.0"),
Messages.getString("DependenciesAnalysis.1")); //$NON-NLS-1$ //$NON-NLS-2$

	private static Logger log = Logger.getLogger(DependenciesAnalysis.class);

	private static AnalysisDescriptor AFFERENT_ELEMENTS = new AnalysisDescriptor(Messages.getString("DependenciesAnalysis.2"), Messages.getString
("DependenciesAnalysis.3"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor EFFERENT_ELEMENTS = new AnalysisDescriptor(Messages.getString("DependenciesAnalysis.4"), Messages.getString
("DependenciesAnalysis.5"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor AFFERENT_ELEMENTS_COUNT = new AnalysisDescriptor(Messages.getString("Afferent Elements Count"),
Messages.getString("Af El Count Desc"), PACKAGE_DESCRIPTOR);

	private static AnalysisDescriptor EFFERENT_ELEMENTS_COUNT = new AnalysisDescriptor(Messages.getString("Efferent Elements Count"),
Messages.getString("Ef El Count Desc"), PACKAGE_DESCRIPTOR);

	private static AnalysisDescriptor ABSTRACTNESS = new AnalysisDescriptor(Messages.getString("Abstractness"), Messages.getString("Abstractness"),
PACKAGE_DESCRIPTOR);

	private static AnalysisDescriptor INSTABILITY = new AnalysisDescriptor(Messages.getString("Instability"), Messages.getString("Instability"),
PACKAGE_DESCRIPTOR);

	// Package Dependency Cycles: Package dependency cycles are reported along with the hierarchical paths of packages participating in package dependency cycles.

	private static AnalysisDescriptor STATUS = new AnalysisDescriptor(Messages.getString("DependenciesAnalysis.6"), Messages.getString
("DependenciesAnalysis.7"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private List afferentElements = new ArrayList();

	private List efferentElements = new ArrayList();

	private Status status = new Status();

	static {
		List analysisDescriptors = PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
		analysisDescriptors.add(ABSTRACTNESS);
		analysisDescriptors.add(INSTABILITY);
		analysisDescriptors.add(AFFERENT_ELEMENTS_COUNT);
		analysisDescriptors.add(AFFERENT_ELEMENTS);
		analysisDescriptors.add(EFFERENT_ELEMENTS_COUNT);
		analysisDescriptors.add(EFFERENT_ELEMENTS);
		analysisDescriptors.add(STATUS);
	}

	public static class Dependency {
		private boolean isResolved;

		private String name;

		private Object reference;

		/**
		 * @return Returns the isResolved.
		 */
		public boolean isResolved() {
			return isResolved;
		}

		/**
		 * @param isResolved
		 *            The isResolved to set.
		 */
		public void setResolved(boolean isResolved) {
			this.isResolved = isResolved;
		}

		/**
		 * @return Returns the name.
		 */
		public String getName() {
			return name;
		}

		/**
		 * @param name
		 *            The name to set.
		 */
		public void setName(String name) {
			this.name = name;
		}

		/**
		 * @return Returns the reference.
		 */
		public Object getReference() {
			return reference;
```

```
		}

		/**
		 * @param reference
		 *            The reference to set.
		 */
		public void setReference(Object reference) {
			this.reference = reference;
		}

		public String toString() {
			return name;
		}
	}

	/*
	 * public Dependency createDependency() { return new Dependency(); }
	 */
	public static DependenciesAnalysis analyzeDependencies(HashMap globalAnalysis, ModelPreProcessor modelTools, IProgramUnit programUnit) {
		DependenciesAnalysis dependenciesAnalysis = (DependenciesAnalysis) globalAnalysis.get(programUnit);
		if (dependenciesAnalysis == null)
			dependenciesAnalysis = new DependenciesAnalysis();
		log.debug("DependencyAnalysis for " + programUnit.getUniqueKey()); //$NON-NLS-1$

		IImports imports = programUnit.getImports();

		if (imports != null) {
			for (Iterator iterator = imports.getImports().values().iterator(); iterator.hasNext();) {
				IImport _import = (IImport) iterator.next();
				// System.out.println("Import " + _import.getName());
				if (_import.getName().endsWith("*")) { // wildcard is imported //$NON-NLS-1$
					log.debug("Import " + _import.getName() + " ignored"); //$NON-NLS-1$ //$NON-NLS-2$
					// BLOCK THAT FOR NOW - WE SHOULD MEASURE EFFECTIVE
					// DEPENDENCIES AND IMPORT WITH WILDCARD IS NOT SUCH !!!!
					/*
					 * String namespaceKey = _import.getName().substring(
					 * _import.getName().length() - 2); if
					 * (namespaces.containsKey(namespaceKey)) {
					 * dependency.setResolved(true);
					 * dependency.setReference(namespaces.get(namespaceKey)); }
					 */
				} else { // fully qualified type is imported
					Dependency dependency = dependenciesAnalysis.createEfferentDependency(globalAnalysis,
modelTools, programUnit, _import
.getName());
					if (dependency != null) {
						dependenciesAnalysis.efferentElements.add(dependency);
						log.debug("Dependency: " + dependency.getName() + ", " //$NON-NLS-1$ //$NON-
NLS-2$
+ (String) (dependency.isResolved() ? "resolved" :
"NOT RESOLVED")); //$NON-NLS-1$ //$NON-NLS-2$
					}
				}
			}
		}
		dependenciesAnalysis.populateAnalysisResults();
		return dependenciesAnalysis;
	}

	private Dependency createEfferentDependency(HashMap globalAnalysis, ModelPreProcessor modelTools, IProgramUnit programUnit,
			String efferentProgramUnitName) {
		Dependency dependency = new Dependency();
		dependency.setName(efferentProgramUnitName);

		if (modelTools.getProgramUnits().containsKey(efferentProgramUnitName)) {
			dependency.setResolved(true);
			IProgramUnit efferentProgramUnit = (IProgramUnit) modelTools.getProgramUnits().getCompatibleResultForKey(
					efferentProgramUnitName, programUnit.getEnclosingNamespace().getEnclosingProject());
			log.debug("Efferent found: " + efferentProgramUnit.getUniqueKey()); //$NON-NLS-1$

			dependency.setReference(efferentProgramUnit);
			// register this programUnit as afferent dependency to
			// the efferent programUnit
			DependenciesAnalysis oppositeEnd = null;
			AnalysisRecord analysisPackage = (AnalysisRecord) globalAnalysis.get(efferentProgramUnit);
			if (analysisPackage != null)
				oppositeEnd = (DependenciesAnalysis) analysisPackage.getAnalysisPackage(PACKAGE_DESCRIPTOR);
			if (oppositeEnd == null)
				oppositeEnd = new DependenciesAnalysis();
			Dependency oppositeEndDependency = new Dependency();
			oppositeEndDependency.setName(programUnit.getFullyQualifiedName());
			oppositeEndDependency.setResolved(true);
			oppositeEndDependency.setReference(programUnit);
			oppositeEnd.afferentElements.add(oppositeEndDependency);
		}
		return dependency;
	}

	/*
	 * (non-Javadoc)
```

```
		 *
		 * @see org.bog.lachesis.metrics.IMetricBundle#getMetricDescrptors()
		 */
		public List getAnalysisDescrptors() {
			return PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see org.bog.lachesis.metrics.IMetricBundle#getMetric(org.bog.lachesis.metrics.MetricDescriptor)
		 */
		private void populateAnalysisResults() {
			setAnalysisResult(ABSTRACTNESS, getAbstractness());
			setAnalysisResult(INSTABILITY, getInstability());
			setAnalysisResult(AFFERENT_ELEMENTS, getAfferentElements());
			setAnalysisResult(EFFERENT_ELEMENTS, getEfferentElements());
			setAnalysisResult(AFFERENT_ELEMENTS_COUNT, new Integer(getAfferentElements().size()));
			setAnalysisResult(EFFERENT_ELEMENTS_COUNT, new Integer(getEfferentElements().size()));
			setAnalysisResult(STATUS, status.toString());
		}

		private Object getAbstractness() {
			// TODO Auto-generated method stub
			return null;
		}

		private Object getInstability() {
			// TODO Auto-generated method stub
			return null;
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see org.bog.lachesis.metrics.IMetricBundle#getStatus()
		 */
		public Status getStatus() {
			return status;
		}

		/**
		 * @return Returns the afferentElements.
		 */
		public List getAfferentElements() {
			return afferentElements;
		}

		/**
		 * @param afferentElements
		 *            The afferentElements to set.
		 */
		public void setAfferentElements(List afferentElements) {
			this.afferentElements = afferentElements;
		}

		/**
		 * @return Returns the efferentElements.
		 */
		public List getEfferentElements() {
			return efferentElements;
		}

		/**
		 * @param efferentElements
		 *            The efferentElements to set.
		 */
		public void setEfferentElements(List efferentElements) {
			this.efferentElements = efferentElements;
		}

		public AnalysisPackageDescriptor getAnalysisPackageDescriptor() {
			return PACKAGE_DESCRIPTOR;
		}
}


/*
 * Created on 2005-6-7
 *
 */
package org.bog.lachesis.analysis.impl;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.Messages;
```

```
import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.Status;
import org.bog.lachesis.analysis.tools.AnalysisRecord;
import org.bog.lachesis.metamodel.IImport;
import org.bog.lachesis.metamodel.IImports;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor;

/**
 * @author Administrator
 *
 */
public class DependenciesAnalysis extends AbstractAnalysisPackage {
	public final static AnalysisPackageDescriptor PACKAGE_DESCRIPTOR = new AnalysisPackageDescriptor(Messages.getString("DependenciesAnalysis.0"),
Messages.getString("DependenciesAnalysis.1")); //$NON-NLS-1$ //$NON-NLS-2$

	private static Logger log = Logger.getLogger(DependenciesAnalysis.class);

	private static AnalysisDescriptor AFFERENT_ELEMENTS = new AnalysisDescriptor(Messages.getString("DependenciesAnalysis.2"), Messages.getString
("DependenciesAnalysis.3"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor EFFERENT_ELEMENTS = new AnalysisDescriptor(Messages.getString("DependenciesAnalysis.4"), Messages.getString
("DependenciesAnalysis.5"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor AFFERENT_ELEMENTS_COUNT = new AnalysisDescriptor(Messages.getString("Afferent Elements Count"),
Messages.getString("Af El Count Desc"), PACKAGE_DESCRIPTOR);

	private static AnalysisDescriptor EFFERENT_ELEMENTS_COUNT = new AnalysisDescriptor(Messages.getString("Efferent Elements Count"),
Messages.getString("Ef El Count Desc"), PACKAGE_DESCRIPTOR);

	private static AnalysisDescriptor ABSTRACTNESS = new AnalysisDescriptor(Messages.getString("Abstractness"), Messages.getString("Abstractness"),
PACKAGE_DESCRIPTOR);

	private static AnalysisDescriptor INSTABILITY = new AnalysisDescriptor(Messages.getString("Instability"), Messages.getString("Instability"),
PACKAGE_DESCRIPTOR);

	// Package Dependency Cycles: Package dependency cycles are reported along with the hierarchical paths of packages participating in package dependency cycles.

	private static AnalysisDescriptor STATUS = new AnalysisDescriptor(Messages.getString("DependenciesAnalysis.6"), Messages.getString
("DependenciesAnalysis.7"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private List afferentElements = new ArrayList();

	private List efferentElements = new ArrayList();

	private Status status = new Status();

	static {
		List analysisDescriptors = PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
		analysisDescriptors.add(ABSTRACTNESS);
		analysisDescriptors.add(INSTABILITY);
		analysisDescriptors.add(AFFERENT_ELEMENTS_COUNT);
		analysisDescriptors.add(AFFERENT_ELEMENTS);
		analysisDescriptors.add(EFFERENT_ELEMENTS_COUNT);
		analysisDescriptors.add(EFFERENT_ELEMENTS);
		analysisDescriptors.add(STATUS);
	}

	public static class Dependency {
		private boolean isResolved;

		private String name;

		private Object reference;

		/**
		 * @return Returns the isResolved.
		 */
		public boolean isResolved() {
			return isResolved;
		}

		/**
		 * @param isResolved
		 *            The isResolved to set.
		 */
		public void setResolved(boolean isResolved) {
			this.isResolved = isResolved;
		}

		/**
		 * @return Returns the name.
		 */
		public String getName() {
			return name;
		}

		/**
		 * @param name
```

```
                         *            The name to set.
                         */
                        public void setName(String name) {
                                this.name = name;
                        }

                        /**
                         * @return Returns the reference.
                         */
                        public Object getReference() {
                                return reference;
                        }

                        /**
                         * @param reference
                         *            The reference to set.
                         */
                        public void setReference(Object reference) {
                                this.reference = reference;
                        }

                        public String toString() {
                                return name;
                        }
                }

                /*
                 * public Dependency createDependency() { return new Dependency(); }
                 */
                public static DependenciesAnalysis analyzeDependencies(HashMap globalAnalysis, ModelPreProcessor modelTools, IProgramUnit programUnit) {
                        DependenciesAnalysis dependenciesAnalysis = (DependenciesAnalysis) globalAnalysis.get(programUnit);
                        if (dependenciesAnalysis == null)
                                dependenciesAnalysis = new DependenciesAnalysis();
                        log.debug("DependencyAnalysis for " + programUnit.getUniqueKey()); //$NON-NLS-1$

                        IImports imports = programUnit.getImports();

                        if (imports != null) {
                                for (Iterator iterator = imports.getImports().values().iterator(); iterator.hasNext();) {
                                        IImport _import = (IImport) iterator.next();
                                        // System.out.println("Import " + _import.getName());
                                        if (_import.getName().endsWith("*")) { // wildcard is imported //$NON-NLS-1$
                                                log.debug("Import " + _import.getName() + " ignored"); //$NON-NLS-1$ //$NON-NLS-2$
                                                // BLOCK THAT FOR NOW - WE SHOULD MEASURE EFFECTIVE
                                                // DEPENDENCIES AND IMPORT WITH WILDCARD IS NOT SUCH !!!!
                                                /*
                                                 * String namespaceKey = _import.getName().substring(
                                                 * _import.getName().length() - 2); if
                                                 * (namespaces.containsKey(namespaceKey)) {
                                                 * dependency.setResolved(true);
                                                 * dependency.setReference(namespaces.get(namespaceKey)); }
                                                 */
                                        } else { // fully qualified type is imported
                                                Dependency dependency = dependenciesAnalysis.createEfferentDependency(globalAnalysis,
modelTools, programUnit, _import
                                                                                                .getName());
                                                if (dependency != null) {
                                                        dependenciesAnalysis.efferentElements.add(dependency);
                                                        log.debug("Dependency: " + dependency.getName() + ", " //$NON-NLS-1$ //$NON-
NLS-2$
                                                                                                        + (String) (dependency.isResolved() ? "resolved" :
"NOT RESOLVED")); //$NON-NLS-1$ //$NON-NLS-2$
                                                }
                                        }
                                }
                        }
                        dependenciesAnalysis.populateAnalysisResults();
                        return dependenciesAnalysis;
                }

                private Dependency createEfferentDependency(HashMap globalAnalysis, ModelPreProcessor modelTools, IProgramUnit programUnit,
                                String efferentProgramUnitName) {
                        Dependency dependency = new Dependency();
                        dependency.setName(efferentProgramUnitName);

                        if (modelTools.getProgramUnits().containsKey(efferentProgramUnitName)) {
                                dependency.setResolved(true);
                                IProgramUnit efferentProgramUnit = (IProgramUnit) modelTools.getProgramUnits().getCompatibleResultForKey(
                                                efferentProgramUnitName, programUnit.getEnclosingNamespace().getEnclosingProject());
                                log.debug("Efferent found: " + efferentProgramUnit.getUniqueKey()); //$NON-NLS-1$

                                dependency.setReference(efferentProgramUnit);
                                // register this programUnit as afferent dependency to
                                // the efferent programUnit
                                DependenciesAnalysis oppositeEnd = null;
                                AnalysisRecord analysisPackage = (AnalysisRecord) globalAnalysis.get(efferentProgramUnit);
                                if (analysisPackage != null)
                                        oppositeEnd = (DependenciesAnalysis) analysisPackage.getAnalysisPackage(PACKAGE_DESCRIPTOR);
                                if (oppositeEnd == null)
                                        oppositeEnd = new DependenciesAnalysis();
```

```
                                        Dependency oppositeEndDependency = new Dependency();
                                        oppositeEndDependency.setName(programUnit.getFullyQualifiedName());
                                        oppositeEndDependency.setResolved(true);
                                        oppositeEndDependency.setReference(programUnit);
                                        oppositeEnd.afferentElements.add(oppositeEndDependency);
                                }
                                return dependency;
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see org.bog.lachesis.metrics.IMetricBundle#getMetricDescrptors()
                         */
                        public List getAnalysisDescrptors() {
                                return PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see org.bog.lachesis.metrics.IMetricBundle#getMetric(org.bog.lachesis.metrics.MetricDescriptor)
                         */
                        private void populateAnalysisResults() {
                                setAnalysisResult(ABSTRACTNESS, getAbstractness());
                                setAnalysisResult(INSTABILITY, getInstability());
                                setAnalysisResult(AFFERENT_ELEMENTS, getAfferentElements());
                                setAnalysisResult(EFFERENT_ELEMENTS, getEfferentElements());
                                setAnalysisResult(AFFERENT_ELEMENTS_COUNT, new Integer(getAfferentElements().size()));
                                setAnalysisResult(EFFERENT_ELEMENTS_COUNT, new Integer(getEfferentElements().size()));
                                setAnalysisResult(STATUS, status.toString());
                        }

                        private Object getAbstractness() {
                                // TODO Auto-generated method stub
                                return null;
                        }

                        private Object getInstability() {
                                // TODO Auto-generated method stub
                                return null;
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see org.bog.lachesis.metrics.IMetricBundle#getStatus()
                         */
                        public Status getStatus() {
                                return status;
                        }

                        /**
                         * @return Returns the afferentElements.
                         */
                        public List getAfferentElements() {
                                return afferentElements;
                        }

                        /**
                         * @param afferentElements
                         *            The afferentElements to set.
                         */
                        public void setAfferentElements(List afferentElements) {
                                this.afferentElements = afferentElements;
                        }

                        /**
                         * @return Returns the efferentElements.
                         */
                        public List getEfferentElements() {
                                return efferentElements;
                        }

                        /**
                         * @param efferentElements
                         *            The efferentElements to set.
                         */
                        public void setEfferentElements(List efferentElements) {
                                this.efferentElements = efferentElements;
                        }

                        public AnalysisPackageDescriptor getAnalysisPackageDescriptor() {
                                return PACKAGE_DESCRIPTOR;
                        }
}

package org.bog.lachesis.analysis.impl;

import java.util.List;
```

```
import org.bog.lachesis.Messages;
import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.Status;


public class McCabesMetrics extends AbstractAnalysisPackage {
	public final static AnalysisPackageDescriptor PACKAGE_DESCRIPTOR = new AnalysisPackageDescriptor(Messages.getString("McCabesMetrics.0"),
Messages.getString("McCabesMetrics.1")); //$NON-NLS-1$ //$NON-NLS-2$

	private Status status = new Status();

	private static AnalysisDescriptor DECISION_DENSITY = new AnalysisDescriptor(Messages.getString("McCabesMetrics.2"), Messages.getString
("McCabesMetrics.3"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor CYCLOMATIC_COMPLEXITY = new AnalysisDescriptor(Messages.getString("McCabesMetrics.4"), Messages.getString
("McCabesMetrics.5"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private static AnalysisDescriptor STATUS_DESCRIPTOR = new AnalysisDescriptor(Messages.getString("McCabesMetrics.6"), Messages.getString
("McCabesMetrics.7"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

	private double decisionDensity;

	private int cyclomaticComplexity;

	static {
		List analysisDescriptors = PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
		analysisDescriptors.add(CYCLOMATIC_COMPLEXITY);
		analysisDescriptors.add(DECISION_DENSITY);
		analysisDescriptors.add(STATUS_DESCRIPTOR);
	}

	private McCabesMetrics() {
	}

	public static McCabesMetrics analyzeSummaryMcCabesMetrics(List measures) {
		McCabesMetrics mcCabesMeasures = new McCabesMetrics();
		for (int i = 0; i < measures.size(); i++) {
			McCabesMetrics temp = (McCabesMetrics) measures.get(i);
			if (temp.cyclomaticComplexity > mcCabesMeasures.cyclomaticComplexity) {
				mcCabesMeasures.cyclomaticComplexity = temp.cyclomaticComplexity;
				mcCabesMeasures.decisionDensity = temp.decisionDensity;
			}
		}
		int cc = mcCabesMeasures.cyclomaticComplexity;

		if ((cc >= 21) && (cc <= 50))
			mcCabesMeasures.status.addError(Messages.getString("McCabesMetrics.8") + cc + Messages.getString("McCabesMetrics.9")); //
$NON-NLS-1$ //$NON-NLS-2$
		if ((cc > 50))
			mcCabesMeasures.status.addFatal(Messages.getString("McCabesMetrics.10") + cc + Messages.getString("McCabesMetrics.11")); //
$NON-NLS-1$ //$NON-NLS-2$
		mcCabesMeasures.populateAnalysisResults();
		return mcCabesMeasures;

	}

	public static McCabesMetrics analyzeMcCabesMetrics(int cc, int lloc) {
		McCabesMetrics mcCabesMeasures = new McCabesMetrics();
		mcCabesMeasures.cyclomaticComplexity = cc;
		mcCabesMeasures.decisionDensity = (double) cc / (double) lloc;

		if ((cc >= 1) && (cc <= 10))
			mcCabesMeasures.status.addInfo(Messages.getString("McCabesMetrics.12")); //$NON-NLS-1$
		if ((cc >= 11) && (cc <= 20))
			mcCabesMeasures.status.addWarn(Messages.getString("McCabesMetrics.13")); //$NON-NLS-1$
		if ((cc >= 21) && (cc <= 50))
			mcCabesMeasures.status.addError(Messages.getString("McCabesMetrics.14") + cc + Messages.getString("McCabesMetrics.15")); //
$NON-NLS-1$ //$NON-NLS-2$
		if ((cc > 50))
			mcCabesMeasures.status.addFatal(Messages.getString("McCabesMetrics.16") + cc + Messages.getString("McCabesMetrics.17")); //
$NON-NLS-1$ //$NON-NLS-2$

		mcCabesMeasures.populateAnalysisResults();
		return mcCabesMeasures;
	}

	/**
	 * @return Returns the decisionDensity.
	 */
	public double getDecisionDensity() {
		return decisionDensity;
	}

	/**
	 * @return Returns the messages.
	 */
	public Status getStatus() {
		return status;
```

```
		}

		/**
		 * @return Returns the cyclomaticComplexity.
		 */
		public int getCyclomaticComplexity() {
			return cyclomaticComplexity;
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see bog.lachesis.metrics.IMetricBundle#getMetricDescrptors()
		 */
		public List getAnalysisDescrptors() {
			return PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
		}

		public AnalysisPackageDescriptor getAnalysisPackageDescriptor() {
			return PACKAGE_DESCRIPTOR;
		}

		private void populateAnalysisResults() {
			setAnalysisResult(DECISION_DENSITY, new Double(getDecisionDensity()));
			setAnalysisResult(CYCLOMATIC_COMPLEXITY, new Integer(getCyclomaticComplexity()));
			setAnalysisResult(STATUS_DESCRIPTOR, status.toString());
		}
}

package org.bog.lachesis.analysis.impl;

import java.util.List;

import org.bog.lachesis.Messages;
import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.Status;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor;


public class ModuleMetrics extends AbstractAnalysisPackage {
		public final static AnalysisPackageDescriptor PACKAGE_DESCRIPTOR = new AnalysisPackageDescriptor(Messages.getString("ModuleMetrics.0"),
Messages.getString("ModuleMetrics.1")); //$NON-NLS-1$ //$NON-NLS-2$

		private Status status = new Status();

		private static AnalysisDescriptor LOC = new AnalysisDescriptor(Messages.getString("ModuleMetrics.2"), Messages.getString("ModuleMetrics.3"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

		private static AnalysisDescriptor CC = new AnalysisDescriptor(Messages.getString("ModuleMetrics.4"), Messages.getString("ModuleMetrics.5"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

		private static AnalysisDescriptor MI = new AnalysisDescriptor(Messages.getString("ModuleMetrics.6"), Messages.getString("ModuleMetrics.7"),
PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

		private static AnalysisDescriptor STATUS_DESCRIPTOR = new AnalysisDescriptor(Messages.getString("ModuleMetrics.8"), Messages.getString
("ModuleMetrics.9"), PACKAGE_DESCRIPTOR); //$NON-NLS-1$ //$NON-NLS-2$

		private int linesOfCode = 0;

		private int classCount = 0;

		private double maintainabilityIndex = 0;

		static {
			List analysisDescriptors = PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
			analysisDescriptors.add(CC);
			analysisDescriptors.add(LOC);
			analysisDescriptors.add(MI);
			analysisDescriptors.add(STATUS_DESCRIPTOR);
		}

		private ModuleMetrics() {

		}

		public static ModuleMetrics analyzeModuleMetrics(IProgramUnit programUnit, List halsteadMetrics, List mcCabesMetrics,
				ModelPreProcessor modelPreProcessor) {
			ModuleMetrics moduleMetrics = new ModuleMetrics();

			moduleMetrics.status.addStatus(modelPreProcessor.getStatus(programUnit));
			moduleMetrics.classCount = 1;
			moduleMetrics.linesOfCode = programUnit.getLastLineNumber() - programUnit.getFirstLineNumber();

			double aveHV = 0.0;
			double aveMC = 0.0;
			int count = 0;
			for (int i = 0; i < halsteadMetrics.size(); i++) {
				if(((HalsteadMetrics) halsteadMetrics.get(i)).getProgramVolume() != Double.NaN)
```

```
                                                aveHV += ((HalsteadMetrics) halsteadMetrics.get(i)).getProgramVolume();
                                        count++;
                                }
                                aveHV = aveHV / count;

                                count = 0;
                                for (int i = 0; i < mcCabesMetrics.size(); i++) {
                                        if (((McCabesMetrics) mcCabesMetrics.get(i)).getCyclomaticComplexity() != Double.NaN)
                                                aveMC += ((McCabesMetrics) mcCabesMetrics.get(i)).getCyclomaticComplexity();
                                        count++;
                                }
                                aveMC = aveMC / count;

                                // 171 - 5.2 * ln(aveHV) - 0.23 * aveMC - 16.2 * ln (aveLOC) + 50 * sin
                                // (sqrt(2.4 * perCM))

                                moduleMetrics.maintainabilityIndex = 171 - 5.2 * Math.log(aveHV) - 0.23 * aveMC - 16.2 * Math.log(moduleMetrics.linesOfCode);
                                if (moduleMetrics.maintainabilityIndex < 30)
                                        moduleMetrics.status.addError(Messages.getString("ModuleMetrics.10") + moduleMetrics.maintainabilityIndex +
Messages.getString("ModuleMetrics.11")); //$NON-NLS-1$ //$NON-NLS-2$

                                moduleMetrics.populateAnalysisResults();
                                return moduleMetrics;
                        }

                        public static ModuleMetrics analyzeSummaryModuleMetrics(List metrics) {
                                ModuleMetrics moduleMetrics = new ModuleMetrics();
                                for (int i = 0; i < metrics.size(); i++) {
                                        ModuleMetrics temp = (ModuleMetrics) metrics.get(i);
                                        moduleMetrics.classCount += temp.classCount;
                                        moduleMetrics.linesOfCode += temp.linesOfCode;
                                }
                                moduleMetrics.populateAnalysisResults();
                                return moduleMetrics;
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see bog.lachesis.metrics.IMetricBundle#getMetric(bog.lachesis.metrics.MetricDescriptor)
                         */
                        private void populateAnalysisResults() {
                                setAnalysisResult(LOC, new Integer(getLinesOfCode()));
                                setAnalysisResult(MI, new Double(getMaintainabilityIndex()));
                                setAnalysisResult(CC, new Integer(getClassCount()));
                                setAnalysisResult(STATUS_DESCRIPTOR, status.toString());
                        }

                        /**
                         * @return Returns the linesOfCode.
                         */
                        public int getLinesOfCode() {
                                return linesOfCode;
                        }

                        /**
                         * @return Returns the maintainabilityIndex.
                         */
                        public double getMaintainabilityIndex() {
                                return maintainabilityIndex;
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see org.bog.lachesis.metrics.IMetricBundle#getMessages()
                         */
                        public Status getStatus() {
                                return status;
                        }

                        public int getClassCount() {
                                return classCount;
                        }

                        public AnalysisPackageDescriptor getAnalysisPackageDescriptor() {
                                return PACKAGE_DESCRIPTOR;
                        }

                        public List getAnalysisDescrptors() {
                                return PACKAGE_DESCRIPTOR.getAnalysisDescriptors();
                        }

}
```

Пакет org.bog.lachesis.analysis.tools

```
package org.bog.lachesis.analysis.tools;

import java.util.HashMap;
```

```
import java.util.Iterator;
import java.util.Map;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.metamodel.IUniquelyIdentifiable;

public class AnalysisHashMap extends HashMap {

		private static final long serialVersionUID = 1L;

		private static Logger log = Logger.getLogger(AnalysisHashMap.class);

		public SubIterator valuesSubIterator() {
			return new SubIterator(values().iterator());
		}

		public void addResultForKey(Object item) throws Exception {
			Object key;
			if (item instanceof IProgramUnit)
				key = ((IProgramUnit) item).getFullyQualifiedName();
			else if (item instanceof IMethod)
				key = ((IMethod) item).getEnclosingProgramUnit().getFullyQualifiedName() + "." + ((IMethod) item).getName(); //$NON-NLS-1$
			else if (item instanceof INamespace)
				key = ((INamespace) item).getName();
			else
				throw new Exception("Incompatible item: " + item.getClass().getName()); //$NON-NLS-1$
			HashMap map = this;
			if (containsKey(key)) {
				Object element = get(key);
				if (element instanceof HashMap) {
					// if we have already multiple result then pick up the map
					map = (HashMap) element;
				} else {
					// if we had single result up till now then create a map
					HashMap multipleResults = new HashMap();
					map.put(key, multipleResults);
					map = multipleResults;
					map.put(((IUniquelyIdentifiable) element).getUniqueKey(), element);
				}
				key = ((IUniquelyIdentifiable) item).getUniqueKey();
			}

			map.put(key, item);
		}

		public Object getResultsForKey(Object key) {
			return get(key);
		}

		public Object getFirstResultForKey(Object key) {
			Object result = get(key);
			if ((result != null) && (result instanceof HashMap))
				return ((HashMap) result).values().toArray()[0];
			return result;
		}

		public Object getCompatibleResultForKey(Object key, IProject project) {
			if (hasSingleResultForKey(key))
				return getFirstResultForKey(key);
			else {
				HashMap results = (HashMap) getResultsForKey(key);
				IUniquelyIdentifiable aResult = null;
				log.debug("START " + results.values().size()); //$NON-NLS-1$
				for (Iterator it = results.values().iterator(); it.hasNext();) {
					aResult = (IUniquelyIdentifiable) it.next();
					log
.debug("COPE " + aResult.getUniqueKey() + " and " + project.getUniqueKey() + " ["
+ ((String) aResult.getUniqueKey()).endsWith((String) project.getUniqueKey()) + "]"); //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$ //$NON-NLS-4$
					if (((String) aResult.getUniqueKey()).endsWith((String) project.getUniqueKey()))
						return aResult;
				}
				return aResult;
			}
		}

		public boolean hasResultForKey(Object key) {
			return containsKey(key);
		}

		public boolean hasMultipleResultsForKey(Object key) {
			Object result = get(key);
			return (result instanceof HashMap);
		}

		public boolean hasSingleResultForKey(Object key) {
			Object result = get(key);
```

```
                        return !(result instanceof HashMap);
                }

                public AnalysisHashMap() {
                        super();
                }

                public AnalysisHashMap(int initialCapacity, float loadFactor) {
                        super(initialCapacity, loadFactor);
                }

                public AnalysisHashMap(int initialCapacity) {
                        super(initialCapacity);
                }

                public AnalysisHashMap(Map m) {
                        super(m);
                }

}

package org.bog.lachesis.analysis.tools;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.IAnalysisPackage;
import org.bog.lachesis.analysis.impl.ChidamberKemererMetrics;
import org.bog.lachesis.analysis.impl.DependenciesAnalysis;
import org.bog.lachesis.analysis.impl.HalsteadMetrics;
import org.bog.lachesis.analysis.impl.McCabesMetrics;
import org.bog.lachesis.analysis.impl.ModuleMetrics;


public class AnalysisRecord {
                private static List analysisPackageDescriptors = new ArrayList();
                private HashMap analysisPackages = new HashMap();

   static {
                // should this be replaced by extension point mechanism ? - or pluggable registryProvider
                analysisPackageDescriptors.add(HalsteadMetrics.PACKAGE_DESCRIPTOR);
                analysisPackageDescriptors.add(McCabesMetrics.PACKAGE_DESCRIPTOR);
                analysisPackageDescriptors.add(ModuleMetrics.PACKAGE_DESCRIPTOR);
                analysisPackageDescriptors.add(DependenciesAnalysis.PACKAGE_DESCRIPTOR);
                analysisPackageDescriptors.add(ChidamberKemererMetrics.PACKAGE_DESCRIPTOR);
   }

   public boolean hasErrors() {
                for (Iterator iter = analysisPackages.values().iterator(); iter.hasNext();) {
                                                IAnalysisPackage analysisPackage = (IAnalysisPackage) iter.next();
                                                if (analysisPackage.getStatus().hasErrors())
                                                                return true;
                        }
      return false;
   }
                public static List getAnalysisPackageDescriptors() {
                        return analysisPackageDescriptors;
                }
                public HashMap getAnalysisPackages() {
                        return analysisPackages;
                }
                public IAnalysisPackage getAnalysisPackage(AnalysisPackageDescriptor analysisPackageDescriptor) {
                        return (IAnalysisPackage) analysisPackages.get(analysisPackageDescriptor);
                }
                public void setAnalysisPackages(HashMap analysisPackages) {
                        this.analysisPackages = analysisPackages;
                }
                public void setAnalysisPackage(IAnalysisPackage analysisPackage) {
                        analysisPackages.put(analysisPackage.getAnalysisPackageDescriptor(), analysisPackage);
                }
}

package org.bog.lachesis.analysis.tools;

import java.io.BufferedWriter;
import java.io.IOException;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.ItemCounter;
import org.bog.lachesis.Messages;
import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.IAnalysisMonitor;
```

```
import org.bog.lachesis.analysis.IAnalysisPackage;
import org.bog.lachesis.analysis.StatusMessage;
import org.bog.lachesis.analysis.impl.ChidamberKemererMetrics;
import org.bog.lachesis.analysis.impl.DependenciesAnalysis;
import org.bog.lachesis.analysis.impl.HalsteadMetrics;
import org.bog.lachesis.analysis.impl.McCabesMetrics;
import org.bog.lachesis.analysis.impl.ModuleMetrics;
import org.bog.lachesis.metamodel.IBlock;
import org.bog.lachesis.metamodel.IClass;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor;
import org.bog.lachesis.reports.XMLTools;


public class GlobalAnalysis implements ISerializable2XML {
	private HashMap globalAnalysis;

	private ModelPreProcessor modelPreProcessor;

	public AnalysisRecord getAnalysisRecord(Object key) {
		return (AnalysisRecord) globalAnalysis.get(key);
	}

	public static boolean isIncludedInAnalysis(Object element) {
		IProject project = null;
		if (element instanceof IMethod) {
			project = ((IMethod) element).getEnclosingProgramUnit().getEnclosingNamespace().getEnclosingProject();
		} else if (element instanceof IProgramUnit) {
			project = ((IProgramUnit) element).getEnclosingNamespace().getEnclosingProject();
		} else if (element instanceof INamespace) {
			project = ((INamespace) element).getEnclosingProject();
		} else if (element instanceof IProject) {
			project = (IProject) element;
		}
		return !((project != null) && (!project.isIncludedInReport()));
	}

	private static void sumRecursivelyAllItemCountersInBlock(IBlock block, ItemCounter operands, ItemCounter operators, ItemCounter branches) {
		operands.addItemCounter(block.getOperands());
		operators.addItemCounter(block.getOperators());
		branches.addItemCounter(block.getBranches());
		for (Iterator iter = block.getBlocks().iterator(); iter.hasNext();) {
			sumRecursivelyAllItemCountersInBlock((IBlock) iter.next(), operands, operators, branches);
		}
	}

	public void analyzeMetrics(IModel model, IAnalysisMonitor analysisMonitor) throws InterruptedException, InvocationTargetException {
		try {
			globalAnalysis = new HashMap();
			modelPreProcessor = new ModelPreProcessor(model);
			modelPreProcessor.initialize(analysisMonitor);

			analysisMonitor.setBeginTask(Messages.getString("GlobalAnalysis.0"), modelPreProcessor.getProgramUnits().size()); //$NON-NLS-
1$

			List modelHalsteadMeasures = new ArrayList();
			List modelMcCabesMeasures = new ArrayList();
			List modelModelMetrics = new ArrayList();
			for (Iterator it = model.getProjects().values().iterator(); it.hasNext();) {
				IProject project = (IProject) it.next();
				if (!isIncludedInAnalysis(project))
					continue;
				List projectHalsteadMeasures = new ArrayList();
				List projectMcCabesMeasures = new ArrayList();
				List projectModuleMetrics = new ArrayList();

				for (Iterator it1 = project.getNamespaces().values().iterator(); it1.hasNext();) {
					INamespace namespace = (INamespace) it1.next();

					List namespaceHalsteadMeasures = new ArrayList();
					List namespaceModuleMetrics = new ArrayList();
					List namespaceMcCabesMeasures = new ArrayList();

					for (Iterator it2 = namespace.getProgramUnits().values().iterator(); it2.hasNext();) {
						IProgramUnit programUnit = (IProgramUnit) it2.next();

						Iterator it3 = programUnit.getMethods().iterator();
						List classHalsteadMeasures = new ArrayList();
						List classMcCabesMeasures = new ArrayList();

						analysisMonitor.processingResource(programUnit.getName(), 1);
						if (analysisMonitor.isCanceled())
							throw new InterruptedException("canceled"); //$NON-NLS-1$

						if (programUnit instanceof IClass)
```

```
                                                                for (; it3.hasNext();) {
                                                                        IMethod method = (IMethod) it3.next();
                                                                        if (method.getName().equals("Messages"))
                                                                                System.out.println("debug here");

                                                                        IBlock block = method.getBlock();
                                                                        ItemCounter operands = new ItemCounter();
                                                                        ItemCounter operators = new ItemCounter();
                                                                        ItemCounter branches = new ItemCounter();

                                                                        sumRecursivelyAllItemCountersInBlock(block,
operands, operators, branches);

                                                                        HalsteadMetrics halstead =
HalsteadMetrics.analyzeHalsteadMetricsInMethod(operators.getUniqueItemsCount(),

operands.getUniqueItemsCount(), operators.getUsedItemsCount(), operands.getUsedItemsCount(), 10);
                                                                        McCabesMetrics mccabe =
McCabesMetrics.analyzeMcCabesMetrics(branches.getUsedItemsCount(), block
                                                                                        .getLinesOfCode());

                                                                        classHalsteadMeasures.add(halstead);
                                                                        classMcCabesMeasures.add(mccabe);

                                                                        AnalysisRecord methodAnalysisPackage = new
AnalysisRecord();
                                                                        methodAnalysisPackage.setAnalysisPackage(halstead);
                                                                        methodAnalysisPackage.setAnalysisPackage(mccabe);
                                                                        globalAnalysis.put(method, methodAnalysisPackage);

                                                                }

                                                        AnalysisRecord programUnitAnalysisPackage = new AnalysisRecord();
                                                        programUnitAnalysisPackage.setAnalysisPackage(HalsteadMetrics
                                                                        .analyzeHalsteadMetricsInProgramUnit
(classHalsteadMeasures));
                                                        programUnitAnalysisPackage.setAnalysisPackage
(McCabesMetrics.analyzeSummaryMcCabesMetrics(classMcCabesMeasures));

                                                        programUnitAnalysisPackage.setAnalysisPackage
(DependenciesAnalysis.analyzeDependencies(globalAnalysis,
                                                                        modelPreProcessor, programUnit));

                                                        if (programUnit instanceof IClass)
                                                                programUnitAnalysisPackage.setAnalysisPackage
(ChidamberKemererMetrics.analyze((IClass) programUnit,
                                                                                modelPreProcessor));

                                                        // ---------
                                                        ModuleMetrics moduleMetrics = ModuleMetrics.analyzeModuleMetrics(programUnit,
classHalsteadMeasures,
                                                                        classMcCabesMeasures, modelPreProcessor);
                                                        programUnitAnalysisPackage.setAnalysisPackage(moduleMetrics);
                                                        // ---------

                                                        globalAnalysis.put(programUnit, programUnitAnalysisPackage);

                                                        namespaceHalsteadMeasures.add(programUnitAnalysisPackage.getAnalysisPackage
(HalsteadMetrics.PACKAGE_DESCRIPTOR));
                                                        namespaceMcCabesMeasures.add(programUnitAnalysisPackage.getAnalysisPackage
(McCabesMetrics.PACKAGE_DESCRIPTOR));
                                                        namespaceModuleMetrics.add(programUnitAnalysisPackage.getAnalysisPackage
(ModuleMetrics.PACKAGE_DESCRIPTOR));

                                                }
                                                AnalysisRecord namespaceAnalysisPackage = new AnalysisRecord();
                                                namespaceAnalysisPackage.setAnalysisPackage(HalsteadMetrics.analyzeSummaryHalsteadMetrics
(namespaceHalsteadMeasures));
                                                namespaceAnalysisPackage.setAnalysisPackage(McCabesMetrics.analyzeSummaryMcCabesMetrics
(namespaceMcCabesMeasures));
                                                namespaceAnalysisPackage.setAnalysisPackage(ModuleMetrics.analyzeSummaryModuleMetrics
(namespaceModuleMetrics));

                                                globalAnalysis.put(namespace, namespaceAnalysisPackage);

                                                projectHalsteadMeasures.add(namespaceAnalysisPackage.getAnalysisPackage
(HalsteadMetrics.PACKAGE_DESCRIPTOR));
                                                projectMcCabesMeasures.add(namespaceAnalysisPackage.getAnalysisPackage
(McCabesMetrics.PACKAGE_DESCRIPTOR));
                                                projectModuleMetrics.add(namespaceAnalysisPackage.getAnalysisPackage
(ModuleMetrics.PACKAGE_DESCRIPTOR));
                                        }
                                        AnalysisRecord projectAnalysisPackage = new AnalysisRecord();
                                        projectAnalysisPackage.setAnalysisPackage(HalsteadMetrics.analyzeSummaryHalsteadMetrics
(projectHalsteadMeasures));
                                        projectAnalysisPackage.setAnalysisPackage(McCabesMetrics.analyzeSummaryMcCabesMetrics
(projectMcCabesMeasures));
                                        projectAnalysisPackage.setAnalysisPackage(ModuleMetrics.analyzeSummaryModuleMetrics(projectModuleMetrics));
                                        globalAnalysis.put(project, projectAnalysisPackage);

                                        modelHalsteadMeasures.add(projectAnalysisPackage.getAnalysisPackage
```

```
(HalsteadMetrics.PACKAGE_DESCRIPTOR));
                                                modelMcCabesMeasures.add(projectAnalysisPackage.getAnalysisPackage
(McCabesMetrics.PACKAGE_DESCRIPTOR));
                                                modelModelMetrics.add(projectAnalysisPackage.getAnalysisPackage(ModuleMetrics.PACKAGE_DESCRIPTOR));
                                        }
                                        AnalysisRecord modelAnalysisPackage = new AnalysisRecord();
                                        modelAnalysisPackage.setAnalysisPackage(HalsteadMetrics.analyzeSummaryHalsteadMetrics(modelHalsteadMeasures));
                                        modelAnalysisPackage.setAnalysisPackage(McCabesMetrics.analyzeSummaryMcCabesMetrics(modelMcCabesMeasures));
                                        modelAnalysisPackage.setAnalysisPackage(ModuleMetrics.analyzeSummaryModuleMetrics(modelModelMetrics));
                                        globalAnalysis.put(model, modelAnalysisPackage);
                                } catch (InterruptedException e) {
                                        throw e;
                                } catch (Exception e) {
                                        throw new InvocationTargetException(e);
                                }

                }

                /**
                 * @return Returns the measurements.
                 */
                public HashMap getGlobalAnalysis() {
                        return globalAnalysis;
                }

                /**
                 * @param measurements
                 *            The measurements to set.
                 */
                public void setGlobalAnalysis(HashMap measurements) {
                        this.globalAnalysis = measurements;
                }

                /**
                 * @return Returns the modelTools.
                 */
                public ModelPreProcessor getModelPreProcessor() {
                        return modelPreProcessor;
                }

                public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                        StringBuffer offset = new StringBuffer();
                        for (int i = 0; i < nestingLevel; i++)
                                offset.append(nestingLevelOffsetString);
                        bw.write("<?xml version=\"1.0\" encoding=\"" + XMLTools.DEFAULT_ENCODING + "\"?>\n");
                        bw.write(offset.toString());
                        bw.write("<globalAnalysis>\n"); //$NON-NLS-1$

                        bw.write(offset.toString() + nestingLevelOffsetString);
                        bw.write("<ModelPreProcessorStatus>\n"); //$NON-NLS-1$
                        for (Iterator i = modelPreProcessor.getPreProcessorStatus().getMessages().listIterator(); i.hasNext();) {
                                StatusMessage message = (StatusMessage) i.next();
                                bw.write(offset.toString() + nestingLevelOffsetString + nestingLevelOffsetString);
                                bw.write("<StatusMessage>"); //$NON-NLS-1$
                                bw.write(XMLTools.escapeForXML(message.getMessage()));
                                bw.write("</StatusMessage>\n"); //$NON-NLS-1$
                        }
                        bw.write(offset.toString() + nestingLevelOffsetString);
                        bw.write("</ModelPreProcessorStatus>\n"); //$NON-NLS-1$

                        for (Iterator iter = modelPreProcessor.getModel().getProjects().values().iterator(); iter.hasNext();) {
                                IProject project = (IProject) iter.next();
                                if (!project.isIncludedInReport())
                                        continue;

                                bw.write(offset.toString() + nestingLevelOffsetString);
                                bw.write("<project name=\"" + XMLTools.escapeForXML(project.getName()) + "\">\n"); //$NON-NLS-1$ //$NON-NLS-2$
                                toXML(getAnalysisRecord(project), nestingLevel + 2, nestingLevelOffsetString, bw);
                                for (Iterator iterator = project.getNamespaces().values().iterator(); iterator.hasNext();) {
                                        INamespace namespace = (INamespace) iterator.next();
                                        bw.write(offset.toString() + nestingLevelOffsetString + nestingLevelOffsetString);
                                        bw.write("<namespace name=\"" + XMLTools.escapeForXML(namespace.getName()) + "\">\n"); //$NON-NLS-1$ //
$NON-NLS-2$
                                        toXML(getAnalysisRecord(namespace), nestingLevel + 3, nestingLevelOffsetString, bw);
                                        for (Iterator it = namespace.getProgramUnits().values().iterator(); it.hasNext();) {
                                                IProgramUnit programUnit = (IProgramUnit) it.next();
                                                bw.write(offset.toString() + nestingLevelOffsetString + nestingLevelOffsetString);
                                                bw.write("<programUnit name=\"" + XMLTools.escapeForXML(programUnit.getFullyQualifiedName
()) + "\">\n"); //$NON-NLS-1$ //$NON-NLS-2$
                                                toXML(getAnalysisRecord(programUnit), nestingLevel + 4, nestingLevelOffsetString, bw);

                                                for (Iterator itM = programUnit.getMethods().iterator(); itM.hasNext();) {
                                                        IMethod method = (IMethod) itM.next();
                                                        bw.write(offset.toString() + nestingLevelOffsetString + nestingLevelOffsetString +
nestingLevelOffsetString);
                                                        bw.write("<method name=\"" + XMLTools.escapeForXML(method.getName()) +
"\">\n"); //$NON-NLS-1$ //$NON-NLS-2$
                                                        AnalysisRecord analysisRecord = getAnalysisRecord(method);
                                                        if (analysisRecord != null)
                                                                toXML(analysisRecord, nestingLevel + 5, nestingLevelOffsetString,
```

```
bw);
                                        else
                                                bw.write("<no_analysis_available/>"); //$NON-NLS-1$

                                        bw.write(offset.toString() + nestingLevelOffsetString + nestingLevelOffsetString + 
nestingLevelOffsetString);
                                        bw.write("</method>\n"); //$NON-NLS-1$
                                }

                                bw.write(offset.toString() + nestingLevelOffsetString + nestingLevelOffsetString);
                                bw.write("</programUnit>\n"); //$NON-NLS-1$
                        }
                        bw.write(offset.toString() + nestingLevelOffsetString + nestingLevelOffsetString);
                        bw.write("</namespace>\n"); //$NON-NLS-1$
                }
                bw.write(offset.toString() + nestingLevelOffsetString);
                bw.write("</project>\n"); //$NON-NLS-1$
        }
        bw.write(offset.toString());
        bw.write("</globalAnalysis>\n"); //$NON-NLS-1$
        bw.flush();
    }

    public void toXML(AnalysisRecord analysisPackage, int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw)
                throws IOException {
        StringBuffer offset = new StringBuffer();
        for (int i = 0; i < nestingLevel; i++)
                offset.append(nestingLevelOffsetString);
        bw.write(offset.toString());
        bw.write("<analysisPackage>\n"); //$NON-NLS-1$

        for (Iterator iter = AnalysisRecord.getAnalysisPackageDescriptors().listIterator(); iter.hasNext();) {
                AnalysisPackageDescriptor analysisPackageDescriptor = (AnalysisPackageDescriptor) iter.next();
                toXML(analysisPackage.getAnalysisPackage(analysisPackageDescriptor), nestingLevel + 1, nestingLevelOffsetString, bw);
        }
        bw.write(offset.toString());
        bw.write("</analysisPackage>\n"); //$NON-NLS-1$
        bw.flush();
    }

    public void toXML(IAnalysisPackage analysisPackage, int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw)
                throws IOException {
        if (analysisPackage == null)
                return;
        StringBuffer offset = new StringBuffer();
        for (int i = 0; i < nestingLevel; i++)
                offset.append(nestingLevelOffsetString);
        bw.write(offset.toString());
        bw.write("<analysisBundle name=\"" + XMLTools.escapeForXML(analysisPackage.getAnalysisPackageDescriptor().getName()) + "\">\n"); //
$NON-NLS-1$ //$NON-NLS-2$
        List descriptors = analysisPackage.getAnalysisDescrptors();
        for (Iterator iter = descriptors.iterator(); iter.hasNext();) {
                AnalysisDescriptor descriptor = (AnalysisDescriptor) iter.next();

                bw.write(offset.toString() + nestingLevelOffsetString);
                bw.write("<analysisItem name=\"" + XMLTools.escapeForXML(descriptor.getName()) + "\""); //$NON-NLS-1$ //$NON-NLS-2$
                if ((descriptor.getDescription() != null) && (!descriptor.getDescription().equals(""))) //$NON-NLS-1$
                        bw.write("description=\"" + XMLTools.escapeForXML(descriptor.getDescription()) + "\""); //$NON-NLS-1$ //
$NON-NLS-2$
                bw.write(">"); //$NON-NLS-1$
                Object analysisResult = analysisPackage.getAnalysisResult(descriptor);
                if (analysisResult != null)
                        bw.write(XMLTools.escapeForXML(analysisResult.toString()));
                else
                        bw.write(XMLTools.escapeForXML("<NA>"));
                bw.write("</analysisItem>\n"); //$NON-NLS-1$
        }

        bw.write(offset.toString());
        bw.write("</analysisBundle>\n"); //$NON-NLS-1$
        bw.flush();
    }
}


package org.bog.lachesis.analysis.tools;

import java.util.HashMap;
import java.util.Iterator;

public class SubIterator implements Iterator {
    private Iterator iterator;
    private Iterator subIterator;


    public SubIterator(Iterator iterator) {
        this.iterator = iterator;
    }

    public void remove() {
```

```
		}

		public boolean hasNext() {
			if ((subIterator != null) && subIterator.hasNext())
				return true;
			return iterator.hasNext();
		}

		public Object next() {
			if (subIterator != null) {
				if (subIterator.hasNext())
					return subIterator.next();
				else
					subIterator = null;
			}
			Object result = iterator.next();
			if (result instanceof HashMap) {
				HashMap subMap = (HashMap) result;
				subIterator = subMap.values().iterator();
				return subIterator.next();
			}
			return result;
		}

		public boolean isSubIteratorActive() {
			return (subIterator != null);
		}
}
```

Пакет org.bog.lachesis.metamodel

```
package org.bog.lachesis.metamodel;

public interface IAttribute extends INamed {

    /**
     * @return Returns the variable.
     */
    public abstract IVariable getVariable();

    /**
     * @param variable The variable to set.
     */
    public abstract void setVariable(IVariable variable);

}
```

```
package org.bog.lachesis.metamodel;

import java.util.List;

import org.bog.lachesis.ItemCounter;

public interface IBlock  extends ISerializable2XML {

    public abstract void clearCode();
    public abstract List getCode();
    public abstract List getAllCode(Class elementsOfType);
    public abstract void addExpression(IExpression value);
    public abstract List getExpressions();
    public abstract void addMethodCall(IMethodCall methodCall);
    public abstract List getMethodCalls();
    public abstract void addAttributeAccess(IReference attributeAccess);
    public abstract List getAttributeAccesses();
    public abstract List getVariables();
    public abstract void addVariable(IVariable value);
    public abstract void addBlock(IBlock value);
    public abstract List getBlocks();
    public abstract void setEnclosingBlock(IBlock block);
    public abstract IBlock getEnclosingBlock();

    /**
     * @return Returns the blockType.
     */
    public abstract IBlockType getBlockType();

    /**
     * @param blockType
     *            The blockType to set.
     */
    public abstract void setBlockType(IBlockType blockType);

    /**
     * @return Returns the branches.
     */
    public abstract ItemCounter getBranches();

    /**
     * @return Returns the operands.
     */
```

```
    public abstract ItemCounter getOperands();

    /**
     * @return Returns the operators.
     */
    public abstract ItemCounter getOperators();

    /**
     * @return Returns the linesOfCode.
     */
    public abstract int getLinesOfCode();
}

package org.bog.lachesis.metamodel;

import java.util.Hashtable;


public interface IClass extends IProgramUnit, ISerializable2XML {


    public abstract void clearSuperClasses();

    public abstract Hashtable getSuperClasses();

    public abstract void addSuperClass(String name, IClass value);

    public abstract void clearSuperInterfaces();

    public abstract Hashtable getSuperInterfaces();

    public abstract void addSuperInterface(String name, IClass value);

}

package org.bog.lachesis.metamodel;

import java.util.List;

public interface IExpression extends ISerializable2XML {
                    public final static int TYPE_SOURCE_TYPE_CAST = 0;
                    public final static int TYPE_SOURCE_METHOD_CALL = 1;
                    public final static int TYPE_SOURCE_VARIABLE_ACCESS = 2;

                    public abstract int getTypeSource();

    public abstract void clearItems();

    public abstract List getItems();

    public abstract void addItem(Object value);

    public abstract IType getType();

                    public abstract IMethodCall getMethodCall();

                    public abstract IReference getReference();

}

package org.bog.lachesis.metamodel;

public interface IImport extends INamed, ISerializable2XML {
}

package org.bog.lachesis.metamodel;

import java.util.Hashtable;

public interface IImports extends ISerializable2XML {
    public Hashtable getImports();

    public IImport addImport(IImport _import);
}


package org.bog.lachesis.metamodel;

import java.util.Hashtable;


public interface IInterface extends IProgramUnit, ISerializable2XML {


    public abstract void clearSuperInterfaces();

    public abstract Hashtable getSuperInterfaces();

    public abstract void addSuperInterface(String name, IInterface value);
```

```
}

package org.bog.lachesis.metamodel;


public interface ILinesOfCode {
    public int getFirstLineNumber();
    public int getLastLineNumber();
}

package org.bog.lachesis.metamodel;

public interface ILocation extends INamed {

}


package org.bog.lachesis.metamodel;

import java.util.List;



public interface IMethod extends IUniquelyIdentifiable,INamed, ILinesOfCode, ISerializable2XML {
                    public abstract IProgramUnit getEnclosingProgramUnit();
                    public abstract void setEnclosingProgramUnit(IProgramUnit programUnit);

    public abstract void clearParameters();
    public abstract List getParameters();
    public abstract void addParameter(IParameter value);

    public abstract int compliesWith(String methodName, List parameterTypes);

    /**
     * @return Returns the returningType.
     */
    public abstract IType getReturningType();

    /**
     * @param returningType The returningType to set.
     */
    public abstract void setReturningType(IType returningType);

    /**
     * @return Returns the block.
     */
    public abstract IBlock getBlock();

    /**
     * @param block The block to set.
     */
    public abstract void setBlock(IBlock block);

    /**
     * @return Returns the modified.
     */
    public abstract IModified getModified();

    /**
     * @param modified The modified to set.
     */
    public abstract void setModified(IModified modified);

}

package org.bog.lachesis.metamodel;


public interface IMethodCall extends INamed,ISerializable2XML {
                    /**
                     * @return the target method located during type resolving
                     */
                    public IMethod getTargetMethod();
                    /**
                     * @return true if the method call was processed for type resolving successfully
                     */
                    public boolean isResolved();
                    /**
                     * @return true if the method call was processed for type resolving
                     */
                    public boolean isProcessed();
}


package org.bog.lachesis.metamodel;

import java.util.Hashtable;

public interface IModel extends INamed, ISerializable2XML {
    public Hashtable getProjects();
```

```
    public IProject addProject(IProject project);
}

package org.bog.lachesis.metamodel;

import java.util.Hashtable;


public interface IModified extends ISerializable2XML {

    public abstract void clearModifiers();

    public abstract Hashtable getModifiers();

    public abstract void addModifier(IModifier value);

}

package org.bog.lachesis.metamodel;


public interface IModifier extends INamed, ISerializable2XML {
}


package org.bog.lachesis.metamodel;

public interface INamed extends ISerializable2XML {
    /**
     * @return Returns the name.
     */
    public abstract String getName();

    /**
     * Sets the name and removes all the spaces in it.
     * @param name The name to set.
     */
    public abstract void setName(String name);

    /**
     * Sets the name directly, without any modification.
     * @param name The name to set.
     */
    public void setRawName(String name);

}

package org.bog.lachesis.metamodel;

import java.util.Hashtable;

public interface INamespace extends IUniquelyIdentifiable, INamed, ISerializable2XML {

                public abstract void clearProgramUnits();

                public abstract Hashtable getProgramUnits();

                public abstract void addProgramUnit(IProgramUnit value);

                public abstract void addProgramUnit(IClass value);

                public abstract void addProgramUnit(IInterface value);

                public abstract IProject getEnclosingProject();

                public abstract void setEnclosingProject(IProject project);
}


package org.bog.lachesis.metamodel;

public interface IParameter extends IVariable, ISerializable2XML {
}

package org.bog.lachesis.metamodel;

import java.util.Hashtable;
import java.util.List;


public interface IProgramUnit extends IUniquelyIdentifiable,INamed, IResourceLocateable, ILinesOfCode, ISerializable2XML {
                public boolean isAbstract();

    public void setImports(IImports imports);

    public IImports getImports();

    public abstract void clearMethods();

    public abstract List getMethods();
```

```
    public abstract IMethod addMethod(IMethod value);

    public abstract void clearAttributes();

    public abstract Hashtable getAttributes();

    public abstract void addAttribute(IAttribute value);

    public abstract String getFullyQualifiedName();

    public abstract INamespace getEnclosingNamespace();

    public abstract void setEnclosingNamespace(INamespace namespace);

    public abstract IProgramUnit getEnclosingProgramUnit();

    public abstract void setEnclosingProgramUnit(IProgramUnit programUnit);

    public abstract List getChildProgramUnits();

    public abstract boolean equals(IProgramUnit programUnit);

    public abstract boolean isSuper(IProgramUnit programUnit);

    /**
     * @return Returns the modified.
     */
    public abstract IModified getModified();

    /**
     * @param modified The modified to set.
     */
    public abstract void setModified(IModified modified);

}



package org.bog.lachesis.metamodel;

import java.util.Hashtable;


public interface IProject extends IUniquelyIdentifiable,INamed, ISerializable2XML, IResourceLocateable {
    public abstract void clearNamespaces();
    public abstract Hashtable getNamespaces();
    public abstract INamespace addNamespace(INamespace value);
    public abstract IModel getEnclosingModel();
    public abstract void setEnclosingModel(IModel model);
    public abstract boolean isIncludedInReport();
    public abstract void setIncludedInReport(boolean includedInReport);
}


package org.bog.lachesis.metamodel;

public interface IReference extends INamed, ISerializable2XML {
                    public abstract IAttribute  getTargetAttribute();
                    public abstract IVariable  getTargetVariable();
}


package org.bog.lachesis.metamodel;

public interface IResourceLocateable {
    public ILocation getAbsoluteLocation();
    public void setAbsoluteLocation(ILocation location);
}

package org.bog.lachesis.metamodel;

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.Serializable;

public interface ISerializable2XML extends Serializable {
                    public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException;
}


package org.bog.lachesis.metamodel;


public interface ISourceCodeLineNumberLocateable {
    public int getLineNumber();
}


package org.bog.lachesis.metamodel;
```

```
public interface IType extends INamed, ISerializable2XML {
		public IProgramUnit getProgramUnit();
		public void setProgramUnit(IProgramUnit programUnit);
		public void setArray(boolean array);
		public boolean isArray();
}

package org.bog.lachesis.metamodel;

public interface IUniquelyIdentifiable {
		public Object getUniqueKey();
}


package org.bog.lachesis.metamodel;

public interface IVariable extends INamed, ISerializable2XML {

		public abstract IModified getModified();
		public abstract void setModified(IModified modified);

		public abstract boolean isDefinedLocally();

		public abstract void setDefinedLocally(boolean definedLocally);

   /**
    * @return Returns the type.
    */
   public abstract IType getType();

   /**
    * @param type
    *            The type to set.
    */
   public abstract void setType(IType type);

   /**
    * @return Returns the initValue.
    */
   public abstract Object getInitValue();

   /**
    * @param initValue The initValue to set.
    */
   public abstract void setInitValue(Object initValue);
}
```

Пакет org.bog.lachesis.metamodel.impl

```
package org.bog.lachesis.metamodel.impl;


import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Iterator;

import org.bog.lachesis.metamodel.IAttribute;
import org.bog.lachesis.metamodel.IModifier;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.metamodel.IVariable;
import org.bog.lachesis.reports.XMLTools;

public abstract class AbstractAttribute extends AbstractNamed implements ISerializable2XML, IAttribute {
		/**
		 * @param name
		 */
		private IVariable variable;


		public AbstractAttribute(String name) {
				super(name);
		}


		/**
		 * @return Returns the variable.
		 */
		public IVariable getVariable() {
				return variable;
		}

		/**
		 * @param variable
		 *            The variable to set.
		 */
		public void setVariable(IVariable variable) {
```

```
                        this.variable = variable;
        }

        public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                        StringBuffer offset = new StringBuffer();
                        for (int i = 0; i < nestingLevel; i++)
                                offset.append(nestingLevelOffsetString);

                        bw.write(offset.toString());
                        bw.write("<attribute name=\""); //$NON-NLS-1$
                        bw.write(XMLTools.escapeForXML(getName()));
                        bw.write("\">\n"); //$NON-NLS-1$

                        Iterator it = variable.getModified().getModifiers().values().iterator();
                        while (it.hasNext())
                                ((IModifier) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
                        if (variable != null)
                                variable.toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
                        bw.write(offset.toString());
                        bw.write("</attribute>\n"); //$NON-NLS-1$
                        bw.flush();
        }
}


package org.bog.lachesis.metamodel.impl;


import java.io.BufferedWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.ItemCounter;
import org.bog.lachesis.metamodel.IReference;
import org.bog.lachesis.metamodel.IBlock;
import org.bog.lachesis.metamodel.IBlockType;
import org.bog.lachesis.metamodel.IExpression;
import org.bog.lachesis.metamodel.IMethodCall;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.metamodel.IVariable;

public abstract class AbstractBlock implements ISerializable2XML, IBlock {
        private IBlockType blockType;

        private List code = new ArrayList();

        private List variables = new ArrayList();

        private List methodCalls = new ArrayList();

        private List attributeAccesses = new ArrayList();

        private List expressions = new ArrayList();

        private List blocks = new ArrayList();

        protected ItemCounter operators = new ItemCounter();

        protected ItemCounter operands = new ItemCounter();

        protected ItemCounter branches = new ItemCounter();

        protected int linesOfCode;

        protected IBlock enclosingBlock;

        public void clearCode() {
                        code.clear();
                        variables.clear();
                        methodCalls.clear();
                        attributeAccesses.clear();
                        expressions.clear();
                        blocks.clear();
        }

        public List getCode() {
                        return code;
        }

        public IBlock getEnclosingBlock() {
                        return enclosingBlock;
        }

        public void setEnclosingBlock(IBlock block) {
                        enclosingBlock = block;
        }

        public void addExpression(IExpression expression) {
```

```
		code.add(expression);
		int linesOfCode = 0;
		expressions.add(expression);
}

public List getExpressions() {
		return expressions;
}

public void addVariable(IVariable variable) {
		code.add(variable);
		variables.add(variable);
}

public List getVariables() {
		return variables;
}

public void addBlock(IBlock block) {
		code.add(block);
		blocks.add(block);
		block.setEnclosingBlock(this);
}

public List getBlocks() {
		return blocks;
}

public void addMethodCall(IMethodCall methodCall) {
		code.add(methodCall);
		methodCalls.add(methodCall);
}

public List getMethodCalls() {
		return methodCalls;
}

public void addAttributeAccess(IReference attributeAccess) {
		code.add(attributeAccess);
		attributeAccesses.add(attributeAccess);
}

public List getAttributeAccesses() {
		return attributeAccesses;
}

/**
 * @return Returns the blockType.
 */
public IBlockType getBlockType() {
		return blockType;
}

/**
 * @param blockType
 *            The blockType to set.
 */
public void setBlockType(IBlockType blockType) {
		this.blockType = blockType;
}

/**
 * @return Returns the branches.
 */
public ItemCounter getBranches() {
		return branches;
}

/**
 * @return Returns the operands.
 */
public ItemCounter getOperands() {
		return operands;
}

/**
 * @return Returns the operators.
 */
public ItemCounter getOperators() {
		return operators;
}

/**
 * @return Returns the linesOfCode.
 */
public int getLinesOfCode() {
		return linesOfCode;
}

public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
```

```
                        StringBuffer offset = new StringBuffer();

                        for (int i = 0; i < nestingLevel; i++)
                                offset.append(nestingLevelOffsetString);
                        bw.write(offset.toString());
                        bw.write("<block branches=\""); //$NON-NLS-1$
                        bw.write(String.valueOf(getBranches().getUsedItemsCount()));
                        bw.write(" of "); //$NON-NLS-1$
                        bw.write(String.valueOf(getBranches().getUniqueItemsCount()));
                        bw.write("\" operators=\""); //$NON-NLS-1$
                        bw.write(String.valueOf(getOperators().getUsedItemsCount()));
                        bw.write(" of "); //$NON-NLS-1$
                        bw.write(String.valueOf(getOperators().getUniqueItemsCount()));
                        bw.write("\" operands=\""); //$NON-NLS-1$
                        bw.write(String.valueOf(getOperands().getUsedItemsCount()));
                        bw.write(" of "); //$NON-NLS-1$
                        bw.write(String.valueOf(getOperands().getUniqueItemsCount()));
                        bw.write("\" loc=\""); //$NON-NLS-1$
                        bw.write(String.valueOf(getLinesOfCode()));
                        bw.write("\">\n"); //$NON-NLS-1$

                        List snippets = getCode();
                        if (snippets.size() > 0) {
                                bw.write(offset.toString() + nestingLevelOffsetString);
                                bw.write("<code_snippets>\n"); //$NON-NLS-1$
                                for (int i = 0; i < snippets.size(); i++) {
                                        ((ISerializable2XML) snippets.get(i)).toXML(nestingLevel + 2, nestingLevelOffsetString, bw);
                                }
                                bw.write(offset.toString() + nestingLevelOffsetString);
                                bw.write("</code_snippets>\n"); //$NON-NLS-1$
                        }

                        /*
                         * for (int i = 0; i < code.size(); i++) { ((ISerializable2XML)
                         * code.get(i)).toXML(nestingLevel + 1, nestingLevelOffsetString, bw); }
                         */
                        bw.write(offset.toString());

                        bw.write("<block/>\n"); //$NON-NLS-1$

                        bw.flush();
                }

                /**
                 * All code snippets of the given type from the block and recursively all sub-block are returned.
                 * */
                public List getAllCode(Class elementsOfType) {
                        List result = new ArrayList();
                        if (elementsOfType.equals(IMethodCall.class))
                                result.addAll(methodCalls);
                        if (elementsOfType.equals(IReference.class))
                                result.addAll(attributeAccesses);
                        if (elementsOfType.equals(IVariable.class))
                                result.addAll(variables);
                        if (elementsOfType.equals(IExpression.class))
                                result.addAll(expressions);
                        for (Iterator iter = blocks.iterator(); iter.hasNext();) {
                                IBlock block = (IBlock) iter.next();
                                result.addAll(block.getAllCode(elementsOfType));
                        }
                        return result;
                }
}


package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.IAttribute;
import org.bog.lachesis.metamodel.IClass;
import org.bog.lachesis.metamodel.IImports;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IModified;
import org.bog.lachesis.metamodel.IModifier;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.reports.XMLTools;
import org.bog.lachesis.tools.Lachesis;

public abstract class AbstractClass extends AbstractProgramUnit implements ISerializable2XML, IClass {

                private static Logger log = Logger.getLogger(Lachesis.class);

                private Hashtable superClasses = new Hashtable();
```

```
private Hashtable superInterfaces = new Hashtable();

/**
 * @param name
 */
public AbstractClass(String name) {
		super(name);
}

public void clearSuperClasses() {
		superClasses.clear();
}

public Hashtable getSuperClasses() {
		return superClasses;
}

public void addSuperClass(String name, IClass value) {
		superClasses.put(name.replaceAll(" ", ""), value); //$NON-NLS-1$ //$NON-NLS-2$
}

public void clearSuperInterfaces() {
		superInterfaces.clear();
}

public Hashtable getSuperInterfaces() {
		return superInterfaces;
}

public void addSuperInterface(String name, IClass value) {
		superInterfaces.put(name.replaceAll(" ", ""), value); //$NON-NLS-1$ //$NON-NLS-2$
}

public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
		log.debug("Serializing Class " + getName()); //$NON-NLS-1$
		StringBuffer offset = new StringBuffer();
		for (int i = 0; i < nestingLevel; i++)
				offset.append(nestingLevelOffsetString);
		String initOffset = offset.toString();
		String subOffset = offset.append(nestingLevelOffsetString).toString();
		bw.write(initOffset);
		bw.write("<class name=\""); //$NON-NLS-1$
		bw.write(XMLTools.escapeForXML(getName()));
		bw.write("\" >\n"); //$NON-NLS-1$

		imports.toXML(nestingLevel + 1, nestingLevelOffsetString, bw);

		Iterator it = null;
		if ((getModified() != null) && (getModified().getModifiers() != null)) {
				it = getModified().getModifiers().values().iterator();
				while (it.hasNext())
						((IModifier) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
		}
		it = superClasses.keySet().iterator();
		while (it.hasNext()) {
				bw.write(subOffset);
				bw.write("<extends>"); //$NON-NLS-1$
				bw.write(XMLTools.escapeForXML((String) it.next()));
				bw.write("</extends>\n"); //$NON-NLS-1$
		}

		it = superInterfaces.keySet().iterator();
		while (it.hasNext()) {
				bw.write(subOffset);
				bw.write("<implements>"); //$NON-NLS-1$
				bw.write(XMLTools.escapeForXML((String) it.next()));
				bw.write("</implements>\n"); //$NON-NLS-1$
		}

		it = attributes.values().iterator();
		while (it.hasNext())
				((IAttribute) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);

		it = methods.iterator();
		while (it.hasNext())
				((IMethod) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);

		bw.write(initOffset);
		bw.write("</class>\n"); //$NON-NLS-1$
		bw.flush();
}

/**
 * @return Returns the modified.
 */
public IModified getModified() {
		return modified;
}

/**
```

```
		 * @param modified
		 *            The modified to set.
		 */
		public void setModified(IModified modified) {
			this.modified = modified;
		}

		/**
		 * @return Returns the imports.
		 */
		public IImports getImports() {
			return imports;
		}

		/**
		 * @param imports
		 *            The imports to set.
		 */
		public void setImports(IImports imports) {
			this.imports = imports;
		}

		public boolean isSuper(IProgramUnit programUnit) {
			if (programUnit instanceof IClass) {
				for (Iterator iter = getSuperClasses().values().iterator(); iter.hasNext();) {
					IProgramUnit superProgramUnit = (IProgramUnit) iter.next();
					// we need instance equality here
					if (programUnit == superProgramUnit)
						return true;
					if ((this !=superProgramUnit) && superProgramUnit.isSuper(programUnit))
						return true;
				}
			} else {
				for (Iterator iter = getSuperInterfaces().values().iterator(); iter.hasNext();) {
					IProgramUnit superProgramUnit = (IProgramUnit) iter.next();
					// we need instance equality here
					if (programUnit == superProgramUnit)
						return true;
					if ((this != superProgramUnit) && superProgramUnit.isSuper(programUnit))
						return true;
				}
			}

			return false;
		}

}


package org.bog.lachesis.metamodel.impl;


import java.io.BufferedWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.bog.lachesis.metamodel.IReference;
import org.bog.lachesis.metamodel.IExpression;
import org.bog.lachesis.metamodel.IMethodCall;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.metamodel.IType;
import org.bog.lachesis.reports.XMLTools;


public abstract class AbstractExpression implements ISerializable2XML, IExpression  {

		private List items = new ArrayList();
		private IType type;
		private IMethodCall methodCall;
		private IReference reference;

		private int typeSource;

		public void clearItems() {
			items.clear();
		}
		public List getItems()
		{
			return items;
		}
		public void addItem(Object value)
		{
			items.add(value);
		}
		/*
		 * (non-Javadoc)
		 *
		 * @see bog.lachesis.model.independent.ISerializable2XML#toXML(int,
		 *      java.lang.String)
```

```
		 */
		public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
			StringBuffer offset = new StringBuffer();
			for (int i = 0; i < nestingLevel; i++)
				offset.append(nestingLevelOffsetString);
			bw.write(offset.toString());
			bw.write("<expression>"); //$NON-NLS-1$
			for (int i = 0; i < items.size(); i++) {
				bw.write("["); //$NON-NLS-1$
				bw.write(XMLTools.escapeForXML(items.get(i).toString()));
				bw.write("] "); //$NON-NLS-1$
			}
			bw.write("</expression>/n"); //$NON-NLS-1$
			bw.flush();
		}
		public IType getType() {
			return type;
		}
		public void setType(IType type) {
			this.type = type;
		}
		public IMethodCall getMethodCall() {
			return methodCall;
		}
		public void setMethodCall(IMethodCall methodCall) {
			this.methodCall = methodCall;
		}
		public IReference getReference() {
			return reference;
		}
		public void setReference(IReference reference) {
			this.reference = reference;
		}
		public int getTypeSource() {
			return typeSource;
		}
		public void setTypeSource(int typeSource) {
			this.typeSource = typeSource;
		}
}


package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;

import org.bog.lachesis.metamodel.IImport;
import org.bog.lachesis.reports.XMLTools;


public abstract class AbstractImport extends AbstractNamed implements IImport{
   /**
    * @param name
    */
   public AbstractImport(String name) {
      super(name);
   }
		public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
			StringBuffer offset = new StringBuffer();
			for (int i = 0; i < nestingLevel; i++)
				offset.append(nestingLevelOffsetString);
			bw.write(offset.toString());
			bw.write("<import>");
			bw.write(XMLTools.escapeForXML(getName()));
			bw.write("</import>\n");
			bw.flush();
		}

}

package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;

import org.bog.lachesis.metamodel.IImport;
import org.bog.lachesis.metamodel.IImports;



public abstract class AbstractImports implements IImports {

		/* (non-Javadoc)
		 * @see org.bog.lachesis.metamodel.independent.ISerializable2XML#toXML(int, java.lang.String, java.io.BufferedWriter)
		 */
		public void toXML(int nestingLevel, String nestingLevelOffsetString,
				BufferedWriter bw) throws IOException {
```

```
                                StringBuffer offset = new StringBuffer();
                                for (int i = 0; i < nestingLevel; i++)
                                        offset.append(nestingLevelOffsetString);
                                bw.write(offset.toString());
                                bw.write("<imports>\n");
                                Iterator it = imports.values().iterator();
                                while (it.hasNext())
                                        ((IImport) it.next()).toXML(nestingLevel + 1,
                                                        nestingLevelOffsetString, bw);
                                bw.write(offset.toString());
                                bw.write("</imports>\n");
                                bw.flush();
                }
        Hashtable imports = new Hashtable();
        /* (non-Javadoc)
         * @see bog.lachesis.metamodel.independent.IImports#getImports()
         */
        public Hashtable getImports() {
                return imports;
        }

        /* (non-Javadoc)
         * @see bog.lachesis.metamodel.independent.IImports#addImport(bog.lachesis.metamodel.independent.IImport)
         */
        public IImport addImport(IImport _import) {
                return (IImport)imports.put(_import.getName(), _import);
        }

}


package org.bog.lachesis.metamodel.impl;


import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.IAttribute;
import org.bog.lachesis.metamodel.IInterface;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IModifier;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.ISerializable2XML;

public abstract class AbstractInterface extends AbstractProgramUnit implements ISerializable2XML, IInterface {
                private static Logger log = Logger.getLogger(AbstractInterface.class);

                private Hashtable superInterfaces = new Hashtable();

                public void clearSuperInterfaces() {
                                superInterfaces.clear();
                }

                public Hashtable getSuperInterfaces() {
                                return superInterfaces;
                }

                public void addSuperInterface(String name, IInterface value) {
                                superInterfaces.put(name.replaceAll(" ", ""), value);
                }

                /**
                 * @param name
                 */
                public AbstractInterface(String name) {
                                super(name);
                }

                public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                                log.debug("Serializing Interface " + getName());
                                StringBuffer offset = new StringBuffer();
                                for (int i = 0; i < nestingLevel; i++)
                                                offset.append(nestingLevelOffsetString);
                                bw.write(offset.toString());
                                bw.write("<interface>\n");
                                super.toXML(nestingLevel + 1, nestingLevelOffsetString, bw);

                                if ((getModified() != null) && (getModified().getModifiers() != null)) {
                                                Iterator it = getModified().getModifiers().values().iterator();
                                                while (it.hasNext())
                                                                ((IModifier) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
                                }
                                Iterator it = superInterfaces.keySet().iterator();
                                while (it.hasNext()) {
                                                bw.write(offset.toString());
                                                bw.write("<extends>");
                                                bw.write((String) it.next());
```

```
                                        bw.write("</extends>\n");
                                }
                                it = methods.iterator();
                                while (it.hasNext())
                                        ((IMethod) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);

                                it = attributes.values().iterator();
                                while (it.hasNext())
                                        ((IAttribute) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);

                                bw.write(offset.toString());
                                bw.write("</interface>\n");
                                bw.flush();
                }

                public boolean isSuper(IProgramUnit programUnit) {
                                for (Iterator iter = getSuperInterfaces().values().iterator(); iter.hasNext();) {
                                        IProgramUnit superProgramUnit = null;
                                                superProgramUnit = (IProgramUnit) iter.next();
                                        // we need instance equality here
                                        if (programUnit == superProgramUnit)
                                                return true;
                                        if (superProgramUnit.isSuper(programUnit))
                                                return true;
                                }
                                return false;
                }

}


package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.IBlock;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IModified;
import org.bog.lachesis.metamodel.IModifier;
import org.bog.lachesis.metamodel.IParameter;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.metamodel.IType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor;
import org.bog.lachesis.reports.XMLTools;

public abstract class AbstractMethod extends AbstractUniquelyIdentified implements ISerializable2XML, IMethod {
                private static Logger log = Logger.getLogger(ModelPreProcessor.class);

                protected IProgramUnit enclosingProgramUnit;

                protected List parameters = new ArrayList();

                protected IModified modified;

                protected IBlock block;

                protected IType returningType;

                public void clearParameters() {
                                parameters.clear();
                }

                public List getParameters() {
                                return parameters;
                }

                public void addParameter(IParameter value) {
                                parameters.add(value);
                }

                /**
                 * @param name
                 */
                public AbstractMethod(String name) {
                                super(name);
                }

                /**
                 * @return Returns the returningType.
                 */
                public IType getReturningType() {
                                return returningType;
                }
```

```
		/**
		 * @param returningType
		 *            The returningType to set.
		 */
		public void setReturningType(IType returningType) {
			this.returningType = returningType;
		}

		/**
		 * @return Returns the block.
		 */
		public IBlock getBlock() {
			return block;
		}

		/**
		 * @param block
		 *            The block to set.
		 */
		public void setBlock(IBlock block) {
			this.block = block;
		}

		public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
			StringBuffer offset = new StringBuffer();
			for (int i = 0; i < nestingLevel; i++)
				offset.append(nestingLevelOffsetString);
			bw.write(offset.toString());
			bw.write("<method name=\"");
			bw.write(XMLTools.escapeForXML(getName()));
			bw.write("\">\n");
			Iterator it = null;
			if ((getModified() != null) && (getModified().getModifiers() != null)) {
				it = getModified().getModifiers().values().iterator();
				while (it.hasNext())
					((IModifier) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
			}
			it = getParameters().iterator();
			while (it.hasNext())
				((IParameter) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
			if (getName().equals("deleteMeLater"))
				System.out.println("debug here");
			if (block != null)
				block.toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
			bw.write(offset.toString());
			bw.write("</method>\n");
			bw.flush();
		}

		/**
		 * @return Returns the modified.
		 */
		public IModified getModified() {
			return modified;
		}

		/**
		 * @param modified
		 *            The modified to set.
		 */
		public void setModified(IModified modified) {
			this.modified = modified;
		}

		/**
		 * @return Returns the enclosingProgramUnit.
		 */
		public IProgramUnit getEnclosingProgramUnit() {
			return enclosingProgramUnit;
		}

		/**
		 * @param enclosingProgramUnit
		 *            The enclosingProgramUnit to set.
		 */
		public void setEnclosingProgramUnit(IProgramUnit enclosingProgramUnit) {
			this.enclosingProgramUnit = enclosingProgramUnit;
			setUniqueKey(getName() + "@" + enclosingProgramUnit.getUniqueKey());
		}

		public String toString() {
			return getName() + "_" + getParameters().toString();
		}
}

package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;
```

```
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IMethodCall;
import org.bog.lachesis.reports.XMLTools;

public abstract class AbstractMethodCall extends AbstractNamed implements IMethodCall {

		public AbstractMethodCall(String name) {
			super(name);
		}

		private IMethod targetMethod;
		private boolean processed = false;

		public boolean isResolved() {
			return getTargetMethod() != null;
		}
		public boolean isProcessed() {
			return processed;
		}
		public void setProcessed(boolean processed) {
			this.processed = processed;
		}
		public IMethod getTargetMethod() {
			return targetMethod;
		}

		public void setTargetMethod(IMethod method) {
			this.targetMethod = method;
		}

		public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
			StringBuffer offset = new StringBuffer();

			for (int i = 0; i < nestingLevel; i++)
				offset.append(nestingLevelOffsetString);
			bw.write(offset.toString());
			bw.write("<method_call content=\"");
			bw.write(XMLTools.escapeForXML(getName()));
			bw.write("\"");

			bw.write(" parsed=\"");
			bw.write(XMLTools.escapeForXML(toString()));
			bw.write("\"");

			IMethod method = getTargetMethod();
			if (method != null) {
				bw.write(" resolved=\"");
				bw.write(XMLTools.escapeForXML(method.getEnclosingProgramUnit().getName()+"."+method.getName()));
				bw.write("\"");
			}
			bw.write(" />\n");
		}
}

package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;

import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.reports.XMLTools;

public class AbstractModel extends AbstractNamed implements IModel {
   /**
    * @param name
    */
   public AbstractModel(String name) {
      super(name);
   }

   Hashtable projects = new Hashtable();

   /*
    * (non-Javadoc)
    *
    * @see bog.lachesis.metamodel.independent.IModel#getProjects()
    */
   public Hashtable getProjects() {
      return projects;
   }


   /*
    * (non-Javadoc)
    *
    * @see bog.lachesis.metamodel.independent.INamed#toXML(int,
    *      java.lang.String)
```

```
     */
    public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
        StringBuffer offset = new StringBuffer();
        for (int i = 0; i < nestingLevel; i++)
            offset.append(nestingLevelOffsetString);
                                    bw.write("<?xml version=\"1.0\" encoding=\""+XMLTools.DEFAULT_ENCODING+"\"?>\n");
        bw.write(offset.toString());
        bw.write("<project>\n"); //$NON-NLS-1$
        super.toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
        Iterator it = projects.values().iterator();
        while (it.hasNext())
            ((AbstractProject) it.next()).toXML(nestingLevel + 1,
                    nestingLevelOffsetString, bw);
        bw.write(offset.toString());
        bw.write("</project>\n"); //$NON-NLS-1$
                                    bw.flush();
    }



    /* (non-Javadoc)
     * @see bog.lachesis.metamodel.independent.IModel#addProject(bog.lachesis.metamodel.independent.IProject)
     */
    public IProject addProject(IProject project) {
        projects.put(project.getName(),project);
        project.setEnclosingModel(this);
        return project;
    }

}


package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;

import org.bog.lachesis.metamodel.IModified;
import org.bog.lachesis.metamodel.IModifier;
import org.bog.lachesis.metamodel.ISerializable2XML;


public abstract class AbstractModified implements ISerializable2XML, IModified {
                    private Hashtable modifiers = new Hashtable();

                    public void clearModifiers() {
                                    modifiers.clear();
                    }
                    public Hashtable getModifiers()
                    {
                                    return modifiers;
                    }
                    public void addModifier(IModifier value)
                    {
                                    modifiers.put(value.getName(),value);
                    }
                    public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                                    StringBuffer offset = new StringBuffer();
        for (int i = 0; i < nestingLevel; i++)
            offset.append(nestingLevelOffsetString);
        if (modifiers.isEmpty())
            return;
        bw.write(offset.toString());
        bw.write("<modifiers>\n"); //$NON-NLS-1$

        Iterator it = modifiers.values().iterator();
        while (it.hasNext())
            ((IModifier) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
        bw.write(offset.toString());
        bw.write("</modifiers>\n");                                 //$NON-NLS-1$
                                    bw.flush();
                    }

}


package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;

import org.bog.lachesis.metamodel.IModifier;
import org.bog.lachesis.metamodel.INamed;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.reports.XMLTools;


public abstract class AbstractModifier extends AbstractNamed implements ISerializable2XML, IModifier, INamed {
```

```
		/**
		 * @param name
		 */
		public AbstractModifier(String name) {
			super(name);
		}
		/* (non-Javadoc)
		 * @see bog.lachesis.model.independent.ISerializable2XML#toXML(int, java.lang.String)
		 */
		public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
			StringBuffer offset = new StringBuffer();
			for (int i = 0; i < nestingLevel; i++)
				offset.append(nestingLevelOffsetString);
			bw.write(offset.toString());
			bw.write("<modifier>");
			bw.write(XMLTools.escapeForXML(getName()));
			bw.write("</modifier>\n");
			bw.flush();
		}
}


package org.bog.lachesis.metamodel.impl;


import java.io.BufferedWriter;
import java.io.IOException;

import org.bog.lachesis.metamodel.INamed;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.reports.XMLTools;

public abstract class AbstractNamed implements ISerializable2XML, INamed {
		private String name;

		/**
		 * @return Returns the name.
		 */
		public String getName() {
			return name;
		}

		/**
		 * Sets the name and removes all spaces from it.
		 * @param name
		 *		The name to set.
		 */
		public void setName(String name) {
			if (name != null)
				name = name.replaceAll(" ", "");
			this.name = name;
		}

		/**
		 * Sets the name directly without any modifications.
		 * @param name
		 *		The name to set.
		 */
		public void setRawName(String name) {
			this.name = name;
		}

		/**
		 * @param name
		 */
		public AbstractNamed(String name) {
			super();
			setName(name);
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see bog.lachesis.model.independent.ISerializable2XML#toXML(int,
		 *	java.lang.String)
		 */
		public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
			StringBuffer offset;
			offset = new StringBuffer();
			for (int i = 0; i < nestingLevel; i++)
				offset.append(nestingLevelOffsetString);
			bw.write(offset.toString());
			bw.write("<name>");
			bw.write(XMLTools.escapeForXML(name));
			bw.write("</name>\n");
			bw.flush();
		}

		public String toString() {
```

```
                return name;
        }
}


package org.bog.lachesis.metamodel.impl;


import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;

import org.bog.lachesis.metamodel.IClass;
import org.bog.lachesis.metamodel.IInterface;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.metamodel.ISerializable2XML;


public abstract class AbstractNamespace extends AbstractUniquelyIdentified implements ISerializable2XML, INamespace {
        private Hashtable elements = new Hashtable();
        private IProject enclosingProject;

        public void clearProgramUnits() {
                elements.clear();
        }
        public Hashtable getProgramUnits() {
                return elements;
        }
   public void addProgramUnit(IProgramUnit programUnit) {
      elements.put(programUnit.getName(), programUnit);
      programUnit.setEnclosingNamespace(this);
   }
   public void addProgramUnit(IClass _class) {
      elements.put(_class.getName(), _class);
      _class.setEnclosingNamespace(this);
   }
   public void addProgramUnit(IInterface _interface) {
      elements.put(_interface.getName(), _interface);
      _interface.setEnclosingNamespace(this);
   }
        /**
         * @param name
         */
        public AbstractNamespace(String name) {
                super(name);
        }
        public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                StringBuffer offset = new StringBuffer();
                for (int i = 0; i < nestingLevel; i++)
                        offset.append(nestingLevelOffsetString);
                bw.write(offset.toString());
                bw.write("<package>\n");
                super.toXML(nestingLevel + 1, nestingLevelOffsetString,bw);
                Iterator it = elements.values().iterator();
                while (it.hasNext())
                        ((IProgramUnit) it.next()).toXML(nestingLevel + 1,
                                        nestingLevelOffsetString, bw);
                bw.write(offset.toString());
                bw.write("</package>\n");
                bw.flush();
        }
        /**
         * @return Returns the ecnlosingProject.
         */
        public IProject getEnclosingProject() {
                return enclosingProject;
        }
        /**
         * @param ecnlosingProject The ecnlosingProject to set.
         */
        public void setEnclosingProject(IProject enclosingProject) {
                this.enclosingProject = enclosingProject;
                setUniqueKey(getName()+"@"+enclosingProject.getUniqueKey());
        }

}


package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;

import org.bog.lachesis.metamodel.IParameter;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.reports.XMLTools;
```

```
public abstract class AbstractParameter extends AbstractVariable implements ISerializable2XML, IParameter
{

   public AbstractParameter(String name) {
                                super(name);
                }

                public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException
   {
      StringBuffer offset = new StringBuffer();
      for (int i = 0; i < nestingLevel; i++)
         offset.append(nestingLevelOffsetString);
      bw.write(offset.toString());
      bw.write("<parameter");
      bw.write(" name=\"");
      bw.write(XMLTools.escapeForXML(getName()));
      bw.write("\"");
      bw.write(" type=\"");
      bw.write(XMLTools.escapeForXML(getType().getName()));
      bw.write("\" />\n");
                                bw.flush();
   }
}

package org.bog.lachesis.metamodel.impl;

import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.IAttribute;
import org.bog.lachesis.metamodel.IImports;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IModified;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IProgramUnit;

public abstract class AbstractProgramUnit extends AbstractUniquelyIdentified implements IProgramUnit {
                private static Logger log = Logger.getLogger(AbstractProgramUnit.class);

                protected List methods = new ArrayList();

                protected Hashtable attributes = new Hashtable();

                protected IModified modified;

                protected INamespace enclosingNamespace;

                protected IProgramUnit enclosingProgramUnit;

                protected List childProgramUnits;

                protected IImports imports;

                public AbstractProgramUnit(String name) {
                                super(name);
                }

                public void clearMethods() {
                                methods.clear();
                }

                public List getMethods() {
                                return methods;
                }

                public IMethod addMethod(IMethod method) {
                                methods.add(method);
                                method.setEnclosingProgramUnit(this);
                                return method;
                }

                public void clearAttributes() {
                                attributes.clear();
                }

                public Hashtable getAttributes() {
                                return attributes;
                }

                public void addAttribute(IAttribute attribute) {
                                attributes.put(attribute.getName(), attribute);
                }

                public String getFullyQualifiedName() {
                                try {
                                                return getEnclosingNamespace().getName() + "." + getName();
```

```
                        } catch (NullPointerException e) {
//                                      System.out.println("Name: " + getName());
                                        e.printStackTrace();
                        }
                        return null;
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.metamodel.independent.IProgramUnit#getEnclosingNamespace()
         */
        public INamespace getEnclosingNamespace() {
                        return enclosingNamespace;
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.metamodel.independent.IProgramUnit#setEnclosingNamespace(org.bog.lachesis.metamodel.independent.INamespace)
         */
        public void setEnclosingNamespace(INamespace namespace) {
                        enclosingNamespace = namespace;
                        setUniqueKey(getName() + "@" + namespace.getUniqueKey());
        }

        /**
         * @return Returns the modified.
         */
        public IModified getModified() {
                        return modified;
        }

        /**
         * @param modified
         *            The modified to set.
         */
        public void setModified(IModified modified) {
                        this.modified = modified;
        }

        /**
         * @return Returns the imports.
         */
        public IImports getImports() {
                        return imports;
        }

        /**
         * @param imports
         *            The imports to set.
         */
        public void setImports(IImports imports) {
                        this.imports = imports;
        }

        /**
         * @param attributes
         *            The attributes to set.
         */
        public void setAttributes(Hashtable attributes) {
                        this.attributes = attributes;
        }

        /**
         * @param methods
         *            The methods to set.
         */
        public void setMethods(List methods) {
                        this.methods = methods;
        }

        /**
         * returns true if the method IProgramUnit.getFullyQualifiedName() for both
         * classes is the same
         */
        public boolean equals(IProgramUnit programUnit) {
                        try {
                                        return programUnit.getFullyQualifiedName().equals(getFullyQualifiedName());
                        } catch (Exception e) {
                                        String message = null;
                                        if (programUnit != null)

                                                        message = "Could not compare ProgramUnits: '" + programUnit.getEnclosingNamespace() + "." +
programUnit.getName()
                                                                        + "' and '" + getEnclosingNamespace() + "." + getName() + "'";
                                        else
                                                        message = "Could not compare null ProgramUnit";
                                        //System.out.println(message);
                                        log.error(message);
```

```
                                        e.printStackTrace();
                        }
                        return false;
        }

        public IProgramUnit getEnclosingProgramUnit() {
                        return enclosingProgramUnit;
        }

        public void setEnclosingProgramUnit(IProgramUnit programUnit) {
                        this.enclosingProgramUnit = programUnit;

        }

        public List getChildProgramUnits() {
                        return childProgramUnits;
        }

}

package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Iterator;

import org.bog.lachesis.metamodel.ILocation;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.metamodel.ISerializable2XML;

public abstract class AbstractProject extends AbstractUniquelyIdentified implements ISerializable2XML, IProject {
        private Hashtable packages = new Hashtable();

        private ILocation absoluteLocation;

        private IModel enclosingModel;

        private boolean includedInReport;

        public void clearNamespaces() {
                        packages.clear();
        }

        public Hashtable getNamespaces() {
                        return packages;
        }

        public INamespace addNamespace(INamespace namespace) {
                        if (packages.containsKey(namespace.getName()))
                                        return (INamespace) packages.get(namespace.getName());
                        packages.put(namespace.getName(), namespace);
                        namespace.setEnclosingProject(this);
                        return namespace;
        }

        public void toXML(String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                        toXML(0, nestingLevelOffsetString, bw);
        }

        /*
         * (non-Javadoc)
         *
         * @see bog.lachesis.model.independent.ISerializable2XML#toXML(int,
         *      java.lang.String)
         */
        public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                        if (!includedInReport)
                                        return;
                        StringBuffer offset = new StringBuffer();
                        for (int i = 0; i < nestingLevel; i++)
                                        offset.append(nestingLevelOffsetString);
                        bw.write(offset.toString());
                        bw.write("<project>\n"); //$NON-NLS-1$
                        super.toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
                        Iterator it = packages.values().iterator();
                        while (it.hasNext())
                                        ((AbstractNamespace) it.next()).toXML(nestingLevel + 1, nestingLevelOffsetString, bw);
                        bw.write(offset.toString());
                        bw.write("</project>\n"); //$NON-NLS-1$
                        bw.flush();
        }

        /**
         * @param name
         */
        public AbstractProject(String name, ILocation location) {
                        super(name);
```

```
                        absoluteLocation = location;
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.metamodel.independent.IResourceLocateable#getAbsoluteLocation()
         */
        public ILocation getAbsoluteLocation() {
                        return absoluteLocation;
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.metamodel.independent.IResourceLocateable#setAbsoluteLocation()
         */
        public void setAbsoluteLocation(ILocation location) {
                        absoluteLocation = location;
        }

        /**
         * @return Returns the enclosingModel.
         */
        public IModel getEnclosingModel() {
                        return enclosingModel;
        }

        /**
         * @param enclosingModel
         *            The enclosingModel to set.
         */
        public void setEnclosingModel(IModel enclosingModel) {
                        this.enclosingModel = enclosingModel;
                        setUniqueKey(getName() + "@" + enclosingModel.getName()); //$NON-NLS-1$
        }

        public boolean isIncludedInReport() {
                        return includedInReport;
        }

        public void setIncludedInReport(boolean includedInReport) {
                        this.includedInReport = includedInReport;
        }
}

package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;

import org.bog.lachesis.metamodel.IAttribute;
import org.bog.lachesis.metamodel.IReference;
import org.bog.lachesis.metamodel.IVariable;
import org.bog.lachesis.reports.XMLTools;

public abstract class AbstractReference extends AbstractNamed implements IReference {

        public AbstractReference(String name) {
                        super(name);
        }

        public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                        StringBuffer offset = new StringBuffer();

                        for (int i = 0; i < nestingLevel; i++)
                                        offset.append(nestingLevelOffsetString);
                        bw.write(offset.toString());
                        if (getTargetVariable() != null) {
                                        if (getTargetVariable().isDefinedLocally())
                                                        bw.write("<local_variable_access name=\""); //$NON-NLS-1$
                                        else
                                                        bw.write("<member_variable_access name=\""); //$NON-NLS-1$
                        } else
                                        bw.write("<unknown_variable_access name=\""); //$NON-NLS-1$
                        bw.write(XMLTools.escapeForXML(getName()));
                        bw.write("\""); //$NON-NLS-1$
                        bw.write(" toString=\""); //$NON-NLS-1$
                        bw.write(XMLTools.escapeForXML(toString()));
                        bw.write("\""); //$NON-NLS-1$

                        IAttribute attribute = getTargetAttribute();
                        if (attribute != null) {
                                        if (attribute.getVariable().getType().getProgramUnit() != null) {
                                                        bw.write(" attr_type_resolved=\""); //$NON-NLS-1$
                                                        bw.write(XMLTools.escapeForXML(attribute.getVariable().getType().getProgramUnit().getFullyQualifiedName()));
                                        } else {
                                                        bw.write(" attr_type_not_resolved=\""); //$NON-NLS-1$
                                                        bw.write(XMLTools.escapeForXML(attribute.getVariable().getType().getName()));
                                        }
```

```
                                        bw.write("\" "); //$NON-NLS-1$
                                }
                                IVariable variable = getTargetVariable();
                                if (variable != null) {
                                        if (variable.getType().isArray()) {
                                                bw.write(" array=\"true\" "); //$NON-NLS-1$
                                        }

                                        if (variable.getType().getProgramUnit() != null) {
                                                bw.write(" var_type_resolved=\""); //$NON-NLS-1$
                                                bw.write(XMLTools.escapeForXML(variable.getType().getProgramUnit().getFullyQualifiedName()));
                                        } else {
                                                bw.write(" var_type_not_resolved=\""); //$NON-NLS-1$
                                                bw.write(XMLTools.escapeForXML(variable.getType().getName()));
                                        }
                                        bw.write("\" "); //$NON-NLS-1$
                                }

                                bw.write(" />\n"); //$NON-NLS-1$
                        }

}


package org.bog.lachesis.metamodel.impl;

import java.io.BufferedWriter;
import java.io.IOException;

import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.metamodel.IType;
import org.bog.lachesis.reports.XMLTools;

public abstract class AbstractType extends AbstractNamed implements ISerializable2XML, IType {
                protected IProgramUnit programUnit;

                protected boolean array = false;

                /**
                 * @param name
                 */
                public AbstractType(String name) {
                        super(name);
                }

                public IProgramUnit getProgramUnit() {
                        return programUnit;
                }

                public boolean isArray() {
                        return array;
                }

                public void setArray(boolean array) {
                        this.array = array;
                }

                public void setProgramUnit(IProgramUnit programUnit) {
                        this.programUnit = programUnit;
                }

                public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
                        StringBuffer offset = new StringBuffer();
                        for (int i = 0; i < nestingLevel; i++)
                                offset.append(nestingLevelOffsetString);
                        bw.write(offset.toString());
                        bw.write("<type>"); //$NON-NLS-1$
                        bw.write(XMLTools.escapeForXML(getName()));
                        bw.write("</type>\n"); //$NON-NLS-1$
                        bw.flush();
                }
}


package org.bog.lachesis.metamodel.impl;

import org.bog.lachesis.metamodel.IUniquelyIdentifiable;

public abstract class AbstractUniquelyIdentified extends AbstractNamed implements IUniquelyIdentifiable {
                public AbstractUniquelyIdentified(String name) {
                        super(name);
                }
                private String uniqueKey;

                public Object getUniqueKey() {
                        return uniqueKey;
                }
                public void setUniqueKey(String key) {
                        uniqueKey = key;
```

```
		}
}

package org.bog.lachesis.metamodel.impl;


import java.io.BufferedWriter;
import java.io.IOException;

import org.bog.lachesis.metamodel.IModified;
import org.bog.lachesis.metamodel.ISerializable2XML;
import org.bog.lachesis.metamodel.IType;
import org.bog.lachesis.metamodel.IVariable;
import org.bog.lachesis.reports.XMLTools;

public abstract class AbstractVariable extends AbstractNamed implements ISerializable2XML, IVariable {
		private boolean definedLocally = true;

		private IModified modified;

		private IType type;

		private Object initValue;

		/**
		 * @param name
		 */
		public AbstractVariable(String name) {
			super(name);
		}

		/**
		 * @return Returns the type.
		 */
		public IType getType() {
			return type;
		}

		/**
		 * @param type
		 *            The type to set.
		 */
		public void setType(IType type) {
			this.type = type;
		}

		public void toXML(int nestingLevel, String nestingLevelOffsetString, BufferedWriter bw) throws IOException {
			StringBuffer offset = new StringBuffer();
			for (int i = 0; i < nestingLevel; i++)
				offset.append(nestingLevelOffsetString);
			bw.write(offset.toString());
			bw.write("<variable");
			bw.write(" name=\"");
			bw.write(XMLTools.escapeForXML(getName()));
			bw.write("\"");
			if ((initValue != null) && (!initValue.equals(""))) {
				bw.write(" value=\"");
				bw.write(XMLTools.escapeForXML(initValue.toString()));
				bw.write("\" ");
			}
			if (type.isArray())
				bw.write(" array=\"true\" ");

			bw.write(" type=\"");
			bw.write(XMLTools.escapeForXML(type.getName()));
			bw.write("\" ");
			if (type.getProgramUnit() != null) {
				bw.write(" resolved_type=\"");
				bw.write(XMLTools.escapeForXML(type.getProgramUnit().getFullyQualifiedName()));
				bw.write("\" ");
			}
			bw.write("/>\n");
			bw.flush();
		}

		/**
		 * @return Returns the initValue.
		 */
		public Object getInitValue() {
			return initValue;
		}

		/**
		 * @param initValue
		 *            The initValue to set.
		 */
		public void setInitValue(Object initValue) {
			this.initValue = initValue;
		}
```

```
		public boolean isDefinedLocally() {
			return definedLocally;
		}

		public void setDefinedLocally(boolean definedLocally) {
			this.definedLocally = definedLocally;
		}

		public IModified getModified() {
			return modified;
		}

		public void setModified(IModified modified) {
			this.modified = modified;
		}
}
```

Пакет org.bog.lachesis.model

```
package org.bog.lachesis.model;

import org.bog.lachesis.analysis.AnalysisException;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.metamodel.IModel;



public interface IModelReader {
    public IModel readModel(IModelSource modelSource, IAnalysisMonitor parseMonitor) throws InterruptedException, AnalysisException;
}
```

```
package org.bog.lachesis.model;

import java.util.List;

public interface IModelSource {
    public String getName();
    public void setName(String name);
    public void includeAnalysisResource(Object resource);
    public void includeSupplementResource(Object resource);
    public List getAnalysisResources();
    public List getSupplementResources();
}
```

Пакет org.bog.lachesis.model.readers

```
package org.bog.lachesis.model.readers;

import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.jar.JarFile;

import org.apache.log4j.Logger;
import org.bog.lachesis.Messages;
import org.bog.lachesis.analysis.AnalysisException;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.model.IModelReader;
import org.bog.lachesis.model.IModelSource;
import org.bog.lachesis.profiler.Profiler;
import org.bog.lachesis.recognizers.binarycode.javabytecode.ASMModelExtractor;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.JavaProjectLocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Model;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Project;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.parser.ModelSourceReader;

public class JavaModelReader implements IModelReader {

		private final static Logger log = Logger.getLogger(JavaModelReader.class);

		private final static String PROFLIED_TASK_NAME="Model Read"; //$NON-NLS-1$

		//private ClassPath classPath;

		private int countFilesByType(List resources) {
			int result = 0;
			for (int i = 0; i < resources.size(); i++) {
				File file = (File) resources.get(i);
				if (file.getName().endsWith(".java")) //$NON-NLS-1$
					result++;
				if (file.getName().endsWith(".jar")) { //$NON-NLS-1$
					JarFile jarFile;
					try {
						jarFile = new JarFile(file);
```

```
                                                result += jarFile.size();
                                        } catch (IOException e) {
                                                e.printStackTrace();
                                        }
                                }
                        }
                        return result;
        }

        public static String composeClassPath(List resources) {
                        StringBuffer classPathStringBuffer = new StringBuffer();
                        for (int i = 0; i < resources.size(); i++) {
                                File file = (File) resources.get(i);
                                if (file.getName().endsWith(".jar")) { //$NON-NLS-1$
                                        log.debug("Adding to CLASSPATH: " + file.getAbsolutePath()); //$NON-NLS-1$
                                        classPathStringBuffer.append(file.getAbsoluteFile()).append(System.getProperty("path.separator")); //$NON-NLS-1$
                                }
                        }
                        return classPathStringBuffer.toString();
        }

        /*
         * (non-Javadoc)
         *
         * @see bog.lachesis.model.IModelReader#readModel(bog.lachesis.model.IModelSource)
         */
        public IModel readModel(IModelSource modelSource, IAnalysisMonitor analysisMonitor) throws AnalysisException, InterruptedException {
                        IModel model = new Model(Messages.getString("JavaModelReader.6")); //$NON-NLS-1$
                        IProject sourceCodeProject = new Project(modelSource.getName(), new JavaProjectLocation(modelSource.getName()));
                        sourceCodeProject.setIncludedInReport(true);
                        model.addProject(sourceCodeProject);

                        List analysisResources = modelSource.getAnalysisResources();
                        List supplementResources = modelSource.getSupplementResources();


                        analysisMonitor.setBeginTask(Messages.getString("JavaModelReader.9"), countFilesByType(analysisResources)); //$NON-NLS-1$

                        Profiler.startProfile(PROFLIED_TASK_NAME);
                        for (int i = 0; i < analysisResources.size(); i++) {
                                File resource = (File) analysisResources.get(i);
                                if (resource.getName().endsWith(".java")) //$NON-NLS-1$
                                        analyzeJavaFile(resource, sourceCodeProject, model, analysisMonitor);
                                else if (resource.getName().endsWith(".jar")) //$NON-NLS-1$
                                        analyzeJarFile(resource, model, analysisMonitor, false);
                        }

                        for (int i = 0; i < supplementResources.size(); i++) {
                                File resource = (File) supplementResources.get(i);
                                if (resource.getName().endsWith(".jar")) //$NON-NLS-1$
                                        analyzeJarFile(resource, model, analysisMonitor, true);
                        }
                        Profiler.stopProfile(PROFLIED_TASK_NAME);

                        log.debug("Profiled: "+ Profiler.getStatus().toString()); //$NON-NLS-1$

                        return model;
        }

        private void analyzeJarFile(File resource, IModel model, IAnalysisMonitor analysisMonitor, boolean isSupplement) throws AnalysisException,
                                InterruptedException {
                        try {
                                log.debug("Analyzing jar file: " + resource.getName()); //$NON-NLS-1$
                                ASMModelExtractor bcme = new ASMModelExtractor();
                                //bcme.setClassPath(classPath);
                                bcme.analyze(model, resource, analysisMonitor, isSupplement);
                        } catch (InterruptedException e) {
                                throw e;
                        } catch (Exception e) {
                                // log.error("Failed to parse: "+resource.getName());
                                if (resource != null)
                                        log.error("analyzeResource() failed for: " + resource.getName()); //$NON-NLS-1$
                                else
                                        log.error("analyzeResource() failed: null file"); //$NON-NLS-1$
                                // e.printStackTrace();
                        }
        }

        private void analyzeJavaFile(File resource, IProject project, IModel model, IAnalysisMonitor analysisMonitor)
                                throws AnalysisException, InterruptedException {
                        try {
                                log.debug("Parsing java file: " + resource.getName()); //$NON-NLS-1$
                                ModelSourceReader modelSourceReader = new ModelSourceReader();
                                analysisMonitor.processingResource(resource.getName(), 1);
                                modelSourceReader.parse(model, project, resource, analysisMonitor);
                                if (analysisMonitor.isCanceled())
                                        throw new InterruptedException("canceled"); //$NON-NLS-1$

                                /*
                                 * ANTLR -> Parser parser = new Parser(); LinguisticProcessor
```

```
                                        * abstractLinguisticProcessor; abstractLinguisticProcessor =
                                        * parser.parseFile(resource, new AntlrParser(), new
                                        * LinguisticProcessor(project));
                                        * parseMonitor.processingResource(resource.getName(), 1);
                                        * abstractLinguisticProcessor.processTree(model, parseMonitor);
                                        */

                                } catch (InterruptedException e) {
                                        throw e;
                                } catch (Exception e) {
                                        // log.error("Failed to parse: "+resource.getName());
                                        if (resource != null)
                                                log.error("analyzeJavaFile() failed for: " + resource.getName()); //$NON-NLS-1$
                                        else
                                                log.error("analyzeJavaFile() failed: null file"); //$NON-NLS-1$
                                        e.printStackTrace();
                                }
                        }
}
```

Пакет org.bog.lachesis.model.sources

```
package org.bog.lachesis.model.sources;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import org.bog.lachesis.model.IModelSource;


public class FileSystemModelSource implements IModelSource {
                private List analysisResources = new ArrayList();

                private List supplementResources = new ArrayList();

                private String name;

                public FileSystemModelSource() {
                }

                public FileSystemModelSource(File analysisResourcesRoot) {
                        this(analysisResourcesRoot, null);
                }
                public FileSystemModelSource(File analysisResourcesRoot, File supplementResourcesRoot) {
                        name = analysisResourcesRoot.getName();

                        recurseInDirFrom(analysisResourcesRoot.getAbsolutePath(), analysisResources);
                        if (supplementResourcesRoot != null)
                                recurseInDirFrom(supplementResourcesRoot.getAbsolutePath(), supplementResources);
                }

                public void includeAnalysisResource(File source) {
                        recurseInDirFrom(source.getAbsolutePath(), analysisResources);
                }

                public void includeSupplementResource(File source) {
                        recurseInDirFrom(source.getAbsolutePath(), supplementResources);
                }

                public void includeAnalysisResource(Object resource) {
                        includeAnalysisResource((File) resource);
                }

                public void includeSupplementResource(Object resource) {
                        includeSupplementResource((File) resource);
                }

                private void recurseInDirFrom(String dirItem, List files) {
                        File file = new File(dirItem);
                        String list[];
                        if (file.isDirectory()) {
                                list = file.list();
                                for (int i = 0; i < list.length; i++)
                                        recurseInDirFrom(dirItem + File.separatorChar + list[i], files);
                        } else {
                                files.add(file);
                        }
                }


                /*
                 * (non-Javadoc)
                 *
                 * @see bog.lachesis.model.IModelSource#getName()
                 */
                public String getName() {
                        return name;
```

```
	}

	public List getAnalysisResources() {
		return analysisResources;
	}

	public List getSupplementResources() {
		return supplementResources;
	}

	/**
	 * @param name The name to set.
	 */
	public void setName(String name) {
		this.name = name;
	}

}
```

Пакет org.bog.lachesis.profiler

```
package org.bog.lachesis.profiler;

import java.util.HashMap;
import java.util.Iterator;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.Status;

public class Profiler {
	private static Logger log = Logger.getLogger(Profiler.class);


	private static HashMap profiles = new HashMap();

	private static long[] getTimeRecord(String task) {
		long[] timeRecord = (long[]) profiles.get(task);
		if (timeRecord == null) {
			timeRecord = new long[2];
			timeRecord[0] = 0;
			timeRecord[1] = 0;
			profiles.put(task, timeRecord);
		}
		return timeRecord;
	}

	public static void startProfile(String task) {
		long[] timeRecord = getTimeRecord(task);
		timeRecord[0] = System.currentTimeMillis();
	}

	public static void stopProfile(String task) {
		long currentTime = System.currentTimeMillis();
		long[] timeRecord = getTimeRecord(task);
		timeRecord[1] += currentTime - timeRecord[0];
	}
	public static long getProfile(String task) {
		return getTimeRecord(task)[1];
	}
	public static Status getStatus() {
		Status status =  new Status();
		for (Iterator iter = profiles.keySet().iterator(); iter.hasNext();) {
			String profiledTask = (String)iter.next();
			status.addInfo(profiledTask+": "+getProfile(profiledTask)+" ms");
		}
		return status;
	}
	public static void logStatus() {
		log.debug(getStatus().toString());
	}
}
```

Пакет org.bog.lachesis.recognizers.binarycode

```
package org.bog.lachesis.recognizers.binarycode;

import java.io.File;
import java.lang.reflect.InvocationTargetException;

import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IProject;


public abstract class AbstractModelExtractor {
	public abstract void analyze(IModel model, File resource, IAnalysisMonitor analysisMonitor) throws InterruptedException,
			InvocationTargetException;
```

```
		public abstract IProgramUnit analyzeProgramUnit(String fullyQualifiedName,IProject project, File resource, IAnalysisMonitor analysisMonitor)
				throws InterruptedException, InvocationTargetException, Exception;
}
```

Пакет org.bog.lachesis.recognizers.binarycode.javabytecode

```
package org.bog.lachesis.recognizers.binarycode.javabytecode;

import java.io.File;
import java.io.InputStream;
import java.lang.reflect.InvocationTargetException;
import java.util.Enumeration;
import java.util.List;
import java.util.jar.JarFile;
import java.util.zip.ZipEntry;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.metamodel.IClass;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IParameter;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.metamodel.IType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Attribute;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Block;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Class;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Import;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Imports;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.JavaFileLocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.JavaProjectLocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Modified;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Modifier;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Package;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Parameter;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Project;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Variable;
import org.objectweb.asm.ClassReader;
import org.objectweb.asm.Opcodes;
import org.objectweb.asm.Type;
import org.objectweb.asm.tree.AbstractInsnNode;
import org.objectweb.asm.tree.ClassNode;
import org.objectweb.asm.tree.FieldInsnNode;
import org.objectweb.asm.tree.FieldNode;
import org.objectweb.asm.tree.JumpInsnNode;
import org.objectweb.asm.tree.MethodInsnNode;
import org.objectweb.asm.tree.MethodNode;
import org.objectweb.asm.tree.TableSwitchInsnNode;

public class ASMModelExtractor implements Opcodes {
		private Logger log = Logger.getLogger(ASMModelExtractor.class);


		private final static String IF = "IF";


		public void analyze(IModel model, File resource, IAnalysisMonitor analysisMonitor, boolean isSupplement) throws InterruptedException,
				InvocationTargetException {
			IClass result = null;
			try {
				JarFile jarFile = new JarFile(resource);
				Enumeration enumeration = jarFile.entries();
				log.debug("Zip file: " + jarFile.getName());

				// Repository repository = prepareRepository();

				Project project = new Project(jarFile.getName().substring(jarFile.getName().lastIndexOf('\\') + 1), new JavaProjectLocation(
						resource));
				project.setIncludedInReport(!isSupplement);
				model.addProject(project);
				if (!isSupplement) {
					INamespace currentNamespace = new Package("default");

					while (enumeration.hasMoreElements()) {
						ZipEntry zipEntry = (ZipEntry) enumeration.nextElement();
						analysisMonitor.processingResource(zipEntry.getName(), 1);
						if (zipEntry.getName().endsWith(".class")) {
							InputStream inputStream = jarFile.getInputStream(zipEntry);
							analyzeProgramUnit(zipEntry.getName().substring(0, zipEntry.getName().lastIndexOf
("")), project, inputStream,
									analysisMonitor);
						}
					}
				}
			} catch (InterruptedException e) {
				throw e;
			} catch (Throwable e) {
				e.printStackTrace();
				throw new InvocationTargetException(e);
```

```
		}
}

/**
 * @param classPath
 *            The classPath to set.
 */
private Modified processModifiers(int accessFlags) {

		Modified modified = new Modified();

		if ((accessFlags & ACC_PRIVATE) != 0)
			modified.addModifier(Modifier.PRIVATE);
		else if ((accessFlags & ACC_PUBLIC) != 0)
			modified.addModifier(Modifier.PUBLIC);

		else if ((accessFlags & ACC_PROTECTED) != 0)
			modified.addModifier(Modifier.PROTECTED);

		if ((accessFlags & ACC_ABSTRACT) != 0)
			modified.addModifier(Modifier.ABSTRACT);
		if ((accessFlags & ACC_FINAL) != 0)
			modified.addModifier(Modifier.FINAL);
		if ((accessFlags & ACC_NATIVE) != 0)
			modified.addModifier(Modifier.NATIVE);
		if ((accessFlags & ACC_STATIC) != 0)
			modified.addModifier(Modifier.STATIC);
		if ((accessFlags & ACC_STRICT) != 0)
			modified.addModifier(Modifier.STRICTFP);
		if ((accessFlags & ACC_SYNCHRONIZED) != 0)
			modified.addModifier(Modifier.SYNCHRONIZED);
		if ((accessFlags & ACC_TRANSIENT) != 0)
			modified.addModifier(Modifier.TRANSIENT);
		if ((accessFlags & ACC_VOLATILE) != 0)
			modified.addModifier(Modifier.VOLATILE);

		return modified;
}

public IProgramUnit analyzeProgramUnit(String fullyQualifiedName, IProject project, JarFile jarFile, IAnalysisMonitor analysisMonitor)
			throws InterruptedException, InvocationTargetException, Exception {
		ZipEntry zipEntry = jarFile.getEntry(fullyQualifiedName.replaceAll("\\.", "/") + ".class");
		if (zipEntry != null) {
			InputStream inputStream = jarFile.getInputStream(zipEntry);
			return analyzeProgramUnit(fullyQualifiedName, project, inputStream, analysisMonitor);
		}
		return null;
}

public IProgramUnit analyzeProgramUnit(String fullyQualifiedName, IProject project, InputStream inputStream,
			IAnalysisMonitor analysisMonitor) throws InterruptedException, InvocationTargetException, Exception {
		log.debug("Extracting program unit: " + fullyQualifiedName);

		if (analysisMonitor.isCanceled())
			throw new InterruptedException("canceled");

		// Repository repository = prepareRepository();

		IProgramUnit result = null;

		// JavaClass jclass = repository.loadClass(fullyQualifiedName);
		ClassReader cr = new ClassReader(inputStream);
		ClassNode classNode = new ClassNode();
		cr.accept(classNode, true);

		// if (jclass != null) {

		String className = classNode.name.substring(classNode.name.lastIndexOf("/") + 1);
		String packageName = classNode.name.substring(0, classNode.name.lastIndexOf("/")).replaceAll("/", ".");

		// if (cn.access)
		result = new Class(className, new JavaFileLocation(fullyQualifiedName + ".class"));
		/*
		 * else if (jclass.isInterface()) result = new Interface(className, new
		 * JavaFileLocation(fullyQualifiedName + ".class")); else throw new
		 * Exception("Unknown entry: " + className);
		 */

		INamespace currentNamespace = (INamespace) project.getNamespaces().get(packageName);
		if (currentNamespace == null) {
			currentNamespace = new Package(packageName);
			log.debug("Extracted Package: " + packageName);
			project.addNamespace(currentNamespace);
		}
		currentNamespace.addProgramUnit(result);

		result.setImports(new Imports());

		log.debug("-------------------------------------------------------");
		log.debug("Entry: " + className);
```

```
                log.debug("-------------------------------------------------");

                if (classNode.superName != null)
                        // if (cn.isClass())
                        ((IClass) result).addSuperClass(Type.getType("L" + classNode.superName + ";").getClassName(), (IClass) result);
                /*
                 * else if (cn.isInterface()) ((IInterface)
                 * result).addSuperInterface(cn.getSuperClass().getClassName(),
                 * (IInterface) result); else throw new Exception("Unknown entry: " +
                 * className);
                 */

                // analyze attributes
                List fields = classNode.fields;
                for (int i = 0; i < fields.size(); i++) {
                        FieldNode fieldNode = (FieldNode) fields.get(i);

                        Attribute attribute = new Attribute(fieldNode.name);
                        Variable variable = new Variable(fieldNode.name);
                        org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Type type = new
org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Type(
                                        Type.getType(fieldNode.desc).getClassName());
                        variable.setType(type);
                        variable.setModified(processModifiers(fieldNode.access));
                        attribute.setVariable(variable);
                        result.addAttribute(attribute);
                }

                // analyze methods
                List methods = classNode.methods;

                for (int i = 0; i < methods.size(); i++) {
                        if (analysisMonitor.isCanceled())
                                throw new InterruptedException("canceled");
                        MethodNode methodNode = (MethodNode) methods.get(i);

                        String methodName = methodNode.name;
                        if (methodName.equals("<clinit>"))
                                continue;
                        if (methodName.equals("<init>"))
                                methodName = className;
                        org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Method modelMethod = new
org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Method(
                                        methodName);
                        // meth.getModifiers() ------------------translate the type
                        IType type = new org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Type(Type.getReturnType(methodNode.desc)
                                        .getClassName());
                        modelMethod.setReturningType(type);

                        modelMethod.setModified(processModifiers(methodNode.access));

                        Type[] argumentTypes = Type.getArgumentTypes(methodNode.desc);
                        for (int aT = 0; aT < argumentTypes.length; aT++) {
                                IParameter parameter = new Parameter("arg" + aT);
                                parameter.setType(new org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Type(argumentTypes[aT].
getClassName()));
                                // parameter.setModified(processModifiers(meth)); cannot
                                // access argument modifiers right now
                                modelMethod.addParameter(parameter);
                        }

                        result.addMethod(modelMethod);
                        Block block = new Block();
                        modelMethod.setBlock(block);
                        // analyze code
                        List instructions = methodNode.instructions;
                        if (instructions != null) {
                                log.debug("----------> Method = " + methodNode.name);
                                for (int j = 0; j < instructions.size(); j++) {
                                        AbstractInsnNode insnNode = (AbstractInsnNode) instructions.get(j);
                                        if ((insnNode instanceof JumpInsnNode) || (insnNode instanceof TableSwitchInsnNode))
                                                block.getBranches().incItemUsage(IF);
                                        if (insnNode instanceof MethodInsnNode) {
                                                MethodInsnNode methodInsnNode = (MethodInsnNode) insnNode;

                                                Type[] types = Type.getArgumentTypes(methodInsnNode.desc);

                                                result.getImports().addImport(new Import(Type.getReturnType
(methodInsnNode.desc).getClassName()));
                                        } else if (insnNode instanceof FieldInsnNode) {
                                                FieldInsnNode fieldInsnNode = (FieldInsnNode) insnNode;
                                                result.getImports().addImport(new Import(Type.getType(fieldInsnNode.desc).
getClassName()));
                                        }
                                }
                        }
                }

                return result;
        }
```

```
}
```

Пакет org.bog.lachesis.recognizers.sourcecode.java.sablecc

```
package org.bog.lachesis.recognizers.sourcecode.java.sablecc;

import org.bog.lachesis.metamodel.IMethod;

public class MethodResolveResult {
	IMethod method;

	int compatibilityDistance = -1;

	public int getCompatibilityDistance() {
		return compatibilityDistance;
	}

	public IMethod getMethod() {
		return method;
	}

	/*************************************************************************
	 * Updates the state of the instance if the <b>newCompatibilityDistance</b>
	 * is shorter than <b>compatibilityDistance</b> and returns true if we have
	 * equality (zero distance).
	 *
	 * @param newCompatibilityDistance
	 * @param newMethod
	 * @return
	 */
	public boolean approximate(int newCompatibilityDistance, IMethod newMethod) {
		if (newCompatibilityDistance != -1)
			if ((compatibilityDistance == -1) || (newCompatibilityDistance < compatibilityDistance)) {

				method = newMethod;
				compatibilityDistance = newCompatibilityDistance;
				// check if we have equality
				if (newCompatibilityDistance == 0)
					return true;
			}
		return false;
	}

	/*************************************************************************
	 * @see method <b>approximate(int newCompatibilityDistance, IMethod
	 *      newMethod)</b>
	 */

	public boolean approximate(MethodResolveResult resolveResult) {
		return approximate(resolveResult.getCompatibilityDistance(), resolveResult.getMethod());
	}

	public boolean matchFound() {
		return compatibilityDistance >= 0;
	}

	public boolean fullMatchFound() {
		return compatibilityDistance == 0;
	}
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;

import org.apache.log4j.Logger;
import org.bog.lachesis.Messages;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.analysis.Status;
import org.bog.lachesis.analysis.tools.AnalysisHashMap;
import org.bog.lachesis.analysis.tools.SubIterator;
import org.bog.lachesis.metamodel.IAttribute;
import org.bog.lachesis.metamodel.IBlock;
import org.bog.lachesis.metamodel.IClass;
import org.bog.lachesis.metamodel.IImport;
import org.bog.lachesis.metamodel.IImports;
import org.bog.lachesis.metamodel.IInterface;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IMethodCall;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.INamed;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IParameter;
```

```
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.metamodel.IReference;
import org.bog.lachesis.metamodel.IType;
import org.bog.lachesis.metamodel.IVariable;
import org.bog.lachesis.profiler.Profiler;
import org.bog.lachesis.recognizers.binarycode.javabytecode.ASMModelExtractor;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Block;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Class;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Expression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.MethodCall;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Project;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Reference;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Type;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.Variable;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst.MethodCall.ReferenceAccess;
import org.bog.lachesis.tools.NullAnalysisMonitor;

public class ModelPreProcessor {
        private static Logger log = Logger.getLogger(ModelPreProcessor.class);

        private final static String PROFILED_TASK_TYPES = "Resolve Types"; //$NON-NLS-1$

        private final static String PROFILED_TASK_METHOD_CALLS = "Resolve Method Calls"; //$NON-NLS-1$

        private AnalysisHashMap programUnits;

        private AnalysisHashMap methods;

        private AnalysisHashMap namespaces;

        private HashMap unitsStatus;

        private HashMap typesResolveProcessingList;

        private HashMap supplementProgramUnits = new HashMap();

        private IModel model;

        private Status preProcessorStatus;

        private ASMModelExtractor modelExtractor;

        private IProgramUnit javaLangObject;

        public ModelPreProcessor(IModel model) {
                this.model = model;
        }

        /**
         * @return Returns the model.
         */
        public IModel getModel() {
                return model;
        }

        /**
         * @param model
         *            The model to set.
         */
        public void setModel(IModel model) {
                this.model = model;
        }

        private String singleQuoted(Object param) {
                if (param != null)
                        return "'" + param + "'";
                else
                        return "''";
        }

        public void initialize(IAnalysisMonitor analysisMonitor) throws Exception {

                programUnits = new AnalysisHashMap();
                namespaces = new AnalysisHashMap();
                methods = new AnalysisHashMap();
                unitsStatus = new HashMap();
                preProcessorStatus = new Status();
                typesResolveProcessingList = new HashMap();

                for (Iterator projectsIterator = model.getProjects().values().iterator(); projectsIterator.hasNext();) {
                        IProject project = (IProject) projectsIterator.next();
                        for (Iterator namespacesIterator = project.getNamespaces().values().iterator(); namespacesIterator.hasNext();) {
                                INamespace namespace = (INamespace) namespacesIterator.next();

                                namespaces.addResultForKey(namespace);
                                for (Iterator programUnitsIterator = namespace.getProgramUnits().values().iterator(); programUnitsIterator.hasNext
();) {
                                        IProgramUnit programUnit = (IProgramUnit) programUnitsIterator.next();
```

```
                                                programUnits.addResultForKey(programUnit);
                                                for (Iterator methodsIterator = programUnit.getMethods().iterator(); methodsIterator.hasNext();) {
                                                        IMethod method = (IMethod) methodsIterator.next();

                                                        methods.addResultForKey(method);
                                                }
                                        }
                                }
                        }

                        // String jdkJarFilePath = System.getProperty("java.home") +
                        // "/lib/rt.jar"; //$NON-NLS-1$ //$NON-NLS-2$
                        // jdkResource = new JarFile(jdkJarFilePath);
                        // jdkProject = new Project("JRE", new ProjectLocation(jdkJarFilePath));
                        // //$NON-NLS-1$
                        // ((Project) jdkProject).setIncludedInReport(false);
                        // model.addProject(jdkProject);
                        modelExtractor = new ASMModelExtractor();
                        // List resources = new ArrayList();
                        // resources.add(jdkResource);
                        // modelExtractor.setClassPath(new
                        // ClassPath(JavaModelReader.composeClassPath(resources)));

                        javaLangObject = getSupplementProgramUnit("java.lang.Object");
                        Class.CLASS_ARRAY.addSuperClass("Object", (IClass) javaLangObject);

                        analysisMonitor.setBeginTask(Messages.getString("ModelPreProcessor.5"), programUnits.values().size() * 2); //$NON-NLS-1$

                        Profiler.startProfile(PROFILED_TASK_TYPES);
                        // resolve types for super classes,local variables, parameters and
                        // members for each and every ProgramUnit

                        for (Iterator projectsIterator = model.getProjects().values().iterator(); projectsIterator.hasNext();) {
                                IProject project = (IProject) projectsIterator.next();
                                if (project.isIncludedInReport())
                                        for (Iterator namespacesIterator = project.getNamespaces().values().iterator(); namespacesIterator.hasNext();) {
                                                INamespace namespace = (INamespace) namespacesIterator.next();

                                                for (Iterator programUnitsIterator = namespace.getProgramUnits().values().iterator();
programUnitsIterator.hasNext();) {

                                                        if (analysisMonitor.isCanceled())
                                                                throw new InterruptedException("canceled"); //$NON-NLS-1$

                                                        IProgramUnit programUnit = (IProgramUnit) programUnitsIterator.next();

                                                        analysisMonitor.setSubTask(Messages.getString("ModelPreProcessor.0") +
programUnit.getName()); //$NON-NLS-1$
                                                        analysisMonitor.processingPart(1);

                                                        log.debug
("----------------------------------------------------------------------------------"); //$NON-NLS-1$
                                                        log.debug("Resolving types for " + singleQuoted(programUnit.getUniqueKey())); //
$NON-NLS-1$
                                                        log.debug
("----------------------------------------------------------------------------------"); //$NON-NLS-1$
                                                        resolveAllTypesInProgramUnit(programUnit);
                                                }
                                        }
                        }

                        Profiler.stopProfile(PROFILED_TASK_TYPES);
                        Profiler.startProfile(PROFILED_TASK_METHOD_CALLS);
                        // resolve method calls in every method in every Class
                        for (Iterator projectsIterator = model.getProjects().values().iterator(); projectsIterator.hasNext();) {
                                IProject project = (IProject) projectsIterator.next();
                                if (project.isIncludedInReport())
                                        for (Iterator namespacesIterator = project.getNamespaces().values().iterator(); namespacesIterator.hasNext();) {
                                                INamespace namespace = (INamespace) namespacesIterator.next();

                                                for (Iterator programUnitsIterator = namespace.getProgramUnits().values().iterator();
programUnitsIterator.hasNext();) {

                                                        if (analysisMonitor.isCanceled())
                                                                throw new InterruptedException("canceled"); //$NON-NLS-1$

                                                        IProgramUnit programUnit = (IProgramUnit) programUnitsIterator.next();

                                                        if (programUnit instanceof IClass) {
                                                                analysisMonitor.setSubTask(Messages.getString
("ModelPreProcessor.10") + programUnit.getName()); //$NON-NLS-1$
                                                                analysisMonitor.processingPart(1);
                                                                log.debug
("----------------------------------------------------------------------------------"); //$NON-NLS-1$
                                                                log.debug("Resolving method calls for " + singleQuoted
(programUnit.getFullyQualifiedName())); //$NON-NLS-1$
                                                                log.debug
("----------------------------------------------------------------------------------"); //$NON-NLS-1$
                                                                resolveAllMethodCallsInClass((IClass) programUnit);
                                                        }
```

```
                                                }
                                        }
                                }
                                Profiler.stopProfile(PROFILED_TASK_METHOD_CALLS);

                                // update status for unresolved method calls and variables
                                int modelUnResolvedMethods = 0;
                                int modelTotalMethodCalls = 0;
                                for (SubIterator iter = programUnits.valuesSubIterator(); iter.hasNext();) {
                                        IProgramUnit programUnit = (IProgramUnit) iter.next();
                                        if (programUnit instanceof IClass) {
                                                // REMOVE THIS CODE if
                                                // (programUnit.getName().equals("LoggerConfiguration"))
                                                // System.out.println("debug here");
                                                int unResolvedMethods = 0;
                                                int totalMethodCalls = 0;
                                                for (Iterator iterator = programUnit.getMethods().iterator(); iterator.hasNext();) {
                                                        IMethod method = (IMethod) iterator.next();
                                                        List allMethodCalls = method.getBlock().getAllCode(IMethodCall.class);
                                                        totalMethodCalls += allMethodCalls.size();
                                                        for (Iterator it = allMethodCalls.iterator(); it.hasNext();) {
                                                                IMethodCall methodCall = (IMethodCall) it.next();
                                                                if (methodCall.getTargetMethod() == null)
                                                                        unResolvedMethods++;
                                                        }
                                                }
                                                if (unResolvedMethods > 0)
                                                        getStatus(programUnit)
                                                                        .addWarn(
                                                                                        Messages.getString
("ModelPreProcessor.12") + unResolvedMethods + Messages.getString("ModelPreProcessor.13") + (totalMethodCalls - unResolvedMethods) //$NON-NLS-1$ //$NON-NLS-2$
                                                                                                        +
Messages.getString("ModelPreProcessor.14")
                                                                                                        +
singleQuoted(programUnit.getFullyQualifiedName())
                                                                                                        +
Messages.getString("ModelPreProcessor.15")); //$NON-NLS-1$
                                                        modelUnResolvedMethods += unResolvedMethods;
                                                        modelTotalMethodCalls += totalMethodCalls;
                                        }
                                        preProcessorStatus.addStatus(getStatus(programUnit));
                                }
                                int modelResolvedMethods = (modelTotalMethodCalls - modelUnResolvedMethods);
                                double resolveRate = 0;
                                if (modelResolvedMethods != 0)
                                        resolveRate = 100 * modelResolvedMethods / (double) modelTotalMethodCalls;
                                preProcessorStatus
                                                .addWarn(Messages.getString("ModelPreProcessor.12") + modelUnResolvedMethods + Messages.getString
("ModelPreProcessor.13") + (modelTotalMethodCalls - modelUnResolvedMethods) //$NON-NLS-1$ //$NON-NLS-2$
                                                                + Messages.getString("ModelPreProcessor.14") + ". The resolve rate is " +
resolveRate + " %"); //$NON-NLS-1$

                                typesResolveProcessingList.clear();
                                typesResolveProcessingList = null;
                }

                private IVariable resolveVariableReferenceInBlock(String referenceName, IBlock hostBlock) {
                        IBlock currentBlock = hostBlock;
                        int positionInBlock = currentBlock.getCode().size();
                        while (currentBlock != null) {
                                for (ListIterator iter = currentBlock.getCode().listIterator(positionInBlock); iter.hasPrevious();) {
                                        Object codeSnippet = iter.previous();
                                        if (codeSnippet instanceof IVariable) {
                                                IVariable variable = (IVariable) codeSnippet;
                                                if (variable.getName().equals(referenceName))
                                                        return variable;
                                        }
                                }
                                positionInBlock = ((Block) currentBlock).getIndexInEnclosingBlock();
                                currentBlock = currentBlock.getEnclosingBlock();
                        }
                        return null;
                }

                private IVariable resolveVariableReferenceInMembers(String variableName, IProgramUnit programUnit) {
                        for (Iterator iter = programUnit.getAttributes().values().iterator(); iter.hasNext();) {
                                IAttribute attribute = (IAttribute) iter.next();
                                if (attribute.getName().equals(variableName))
                                        return attribute.getVariable();
                        }
                        return null;
                }

                private IVariable resolveVariableReferenceInEnclosingProgramUnits(String referenceName, IProgramUnit hostProgramUnit) {
                        IVariable resolvedVariable = null;
                        IProgramUnit enclosingProgramUnit = hostProgramUnit.getEnclosingProgramUnit();
                        resolvedVariable = resolveVariableReferenceInMembers(referenceName, enclosingProgramUnit);
                        if (resolvedVariable != null)
                                return resolvedVariable;
                        if (enclosingProgramUnit.getEnclosingProgramUnit() != null)
```

```
                        return resolveVariableReferenceInEnclosingProgramUnits(referenceName, enclosingProgramUnit);
                return null;
        }

        private IVariable resolveVariableReferenceInProgramUnitHierarcy(String referenceName, IProgramUnit hostProgramUnit) {
                IVariable resolvedVariable = null;
                resolvedVariable = resolveVariableReferenceInMembers(referenceName, hostProgramUnit);
                if (resolvedVariable != null)
                        return resolvedVariable;
                // check enclosing units if the current one is an inner class
                if (hostProgramUnit.getEnclosingProgramUnit() != null) {
                        resolvedVariable = resolveVariableReferenceInEnclosingProgramUnits(referenceName, hostProgramUnit);
                        if (resolvedVariable != null)
                                return resolvedVariable;
                }
                if (hostProgramUnit instanceof IClass) {
                        for (Iterator iter = ((IClass) hostProgramUnit).getSuperClasses().values().iterator(); iter.hasNext();) {
                                IProgramUnit current = (IProgramUnit) iter.next();
                                if (current != hostProgramUnit)
                                        if ((resolvedVariable = resolveVariableReferenceInProgramUnitHierarcy(referenceName, current)) !=
null)
                                                return resolvedVariable;

                        }

                        for (Iterator iter = ((IClass) hostProgramUnit).getSuperInterfaces().values().iterator(); iter.hasNext();) {
                                IProgramUnit current = (IProgramUnit) iter.next();
                                if (current != hostProgramUnit)
                                        if ((resolvedVariable = resolveVariableReferenceInProgramUnitHierarcy(referenceName, current)) !=
null)
                                                return resolvedVariable;
                        }
                } else if (hostProgramUnit instanceof IInterface) {
                        for (Iterator iter = ((IInterface) hostProgramUnit).getSuperInterfaces().values().iterator(); iter.hasNext();) {
                                IProgramUnit current = (IProgramUnit) iter.next();
                                if (current != hostProgramUnit)
                                        if ((resolvedVariable = resolveVariableReferenceInProgramUnitHierarcy(referenceName, current)) !=
null)
                                                return resolvedVariable;
                        }
                }
                return null;
        }

        private IVariable detectVariable(String variableName, IProgramUnit hostProgramUnit, IMethod hostMethod, IBlock hostBlock)
                        throws Exception {
                IVariable resolvedVariable = null;
                // search for this variable name in the parameters of the method
                // if there is a parameter with mathching name then it is definitely our
                // variable
                // because the name of a parameter canot be redeclared in the method

                for (Iterator iter = hostMethod.getParameters().iterator(); iter.hasNext();) {
                        IParameter parameter = (IParameter) iter.next();
                        if (parameter.getName().equals(variableName))
                                resolvedVariable = (IVariable) parameter;
                }
                if (resolvedVariable != null)
                        log.debug(" Resolved variable type name in parameters " + singleQuoted(resolvedVariable.getName())); //$NON-NLS-1$

                // search for this variable in the block and in all enclosing blocks
                // upper blocks are searched only above the point a sub block was
                // defined

                if (resolvedVariable == null)
                        if ((resolvedVariable = resolveVariableReferenceInBlock(variableName, hostBlock)) != null)
                                log.debug(" Resolved variable type name in blocks " + singleQuoted(resolvedVariable.getName())); //$NON-NLS-1$

                // search the class members of the host class and then all its super
                // classes/interfaces
                if (resolvedVariable == null)
                        if ((resolvedVariable = resolveVariableReferenceInProgramUnitHierarcy(variableName, hostProgramUnit)) != null)
                                log.debug(" Resolved variable type name in class hierarchy " + singleQuoted(resolvedVariable.getName())); //$NON-
NLS-1$

                return resolvedVariable;

        }

        private IVariable resolveReference(IReference reference, IClass hostClass, IMethod hostMethod, IBlock hostBlock) throws Exception {

                IVariable resolvedVariable = null;
                IProgramUnit resolvedProgramUnit = null;
                IProgramUnit currentProgramUnit = hostClass;

                log.debug(" Resolving variable for name " + singleQuoted(reference.toString())); //$NON-NLS-1$

                StringBuffer typeName = new StringBuffer("");
                boolean variableFound = false;
```

```
for (Iterator iter = ((Reference) reference).getIdentifiers().iterator(); iter.hasNext();) {
	if (currentProgramUnit == null) {
		resolvedProgramUnit = null;
		break;
	}

	String segment = (String) iter.next();
	boolean indexed = false;
	// check if it is an index operation
	if (segment.indexOf("[") > -1)
		indexed = true;

	segment = extractTypeName(segment);
	log.debug("Assessing segment " + singleQuoted(segment)); //$NON-NLS-1$

	// check if the reference is the null reference
	if (segment.equals("null")) {
		resolvedProgramUnit = Class.CLASS_WRAPPER_NULL;
		break;
	}
	// if it is still about a local for the block access
	if (!variableFound)
		resolvedVariable = detectVariable(segment, currentProgramUnit, hostMethod, hostBlock);
	else
		resolvedVariable = resolveVariableReferenceInProgramUnitHierarcy(segment, currentProgramUnit);

	if (resolvedVariable != null) {
		variableFound = true;
		if (resolvedVariable.getType().isArray() && (!indexed)) {
			resolvedProgramUnit = Class.CLASS_ARRAY;
			currentProgramUnit = resolvedProgramUnit;
			log
.debug("Segment " + singleQuoted(segment) + " resolved as " +
singleQuoted(currentProgramUnit.getFullyQualifiedName())); //$NON-NLS-1$
		} else {
			resolvedProgramUnit = resolvedVariable.getType().getProgramUnit();
			currentProgramUnit = resolvedProgramUnit;
		}
		continue;

	}

	// check if the reference is a reference to this
	if (segment.equals("this")) {
		resolvedVariable = null;
		variableFound = true;
		resolvedProgramUnit = currentProgramUnit;
		log
.debug("Segment " + singleQuoted(segment) + " resolved as this reference for " +
singleQuoted(currentProgramUnit.getFullyQualifiedName())); //$NON-NLS-1$
	}
	// check if the reference is a reference to super
	if (segment.equals("super")) {
		resolvedVariable = null;
		variableFound = true;
		Iterator it = hostClass.getSuperClasses().values().iterator();
		if (it.hasNext()) {
			resolvedProgramUnit = (IProgramUnit) it.next();
			currentProgramUnit = resolvedProgramUnit;
		}
		log
.debug("Segment " + singleQuoted(segment) + " resolved as super reference" +
singleQuoted(currentProgramUnit.getFullyQualifiedName())); //$NON-NLS-1$
	}
	// check if the reference is a reference to class
	if (segment.equals("class")) {
		resolvedVariable = null;
		variableFound = true;
		resolvedProgramUnit = Class.CLASS_CLASS;
		currentProgramUnit = resolvedProgramUnit;
		log.debug("Segment " + singleQuoted(segment) + " resolved as " + singleQuoted
(currentProgramUnit.getFullyQualifiedName())); //$NON-NLS-1$
	}

	if (!variableFound) {
		// check if we have fully-qualified name
		if (typeName.length() > 0)
			typeName.append(".");
		typeName.append(segment);

		IProgramUnit accessedProgramUnit = resolveProgramUnit(typeName.toString(), hostClass, true);
		if (accessedProgramUnit != null) {
			resolvedProgramUnit = accessedProgramUnit;
			currentProgramUnit = resolvedProgramUnit;
			log
.debug("Segment " + singleQuoted(typeName.toString()) + " resolved
as " + singleQuoted(accessedProgramUnit.getFullyQualifiedName())); //$NON-NLS-1$
		}
	}
```

```
                }

                if (resolvedVariable != null) {
                        ((Reference) reference).setTargetVariable(resolvedVariable);
                        return resolvedVariable;
                }

                String varName = ((Reference) reference).getIdentifiers().toString();

                if (resolvedProgramUnit != null) {
                        resolvedVariable = new Variable(varName);
                        Type type = new Type(resolvedProgramUnit.getName());
                        type.setProgramUnit(resolvedProgramUnit);
                        ((Variable) resolvedVariable).setType(type);
                        ((Reference) reference).setTargetVariable(resolvedVariable);
                        return resolvedVariable;
                }
                log
                                .debug("Could not resolve reference " + singleQuoted(varName) + " in " + singleQuoted
(hostClass.getFullyQualifiedName() + "." + hostMethod.getName())); //$NON-NLS-1$
                return resolvedVariable;
        }

        private IProgramUnit resolveType(IType type, IProgramUnit programUnit, boolean resolveTypes) throws Exception {
                IProgramUnit result = resolveProgramUnit(type.getName(), programUnit, resolveTypes);

                if (result != null) {
                        type.setProgramUnit(result);
                        log.debug("Type " + singleQuoted(type.getName()) + " resolved as " + singleQuoted(result.getFullyQualifiedName())); //$NON-
NLS-1$
                } else {
                        String message = "Could not resolve type " + singleQuoted(type.getName()) + " accessed in "
                                        + singleQuoted(programUnit.getFullyQualifiedName());
                        getStatus(programUnit).addWarn(message);
                        log.error(message);
                }

                return result;
        }

        private IProgramUnit resolveVariableType(IVariable variable, IProgramUnit programUnit) throws Exception {
                String varTypeName = variable.getType().getName();
                IProgramUnit result = resolveProgramUnit(varTypeName, programUnit, false);

                if (result != null) {
                        variable.getType().setProgramUnit(result);
                        if (varTypeName.endsWith("[]")) {
                                variable.getType().setArray(true);
                        }
                        log.debug("Variable " + singleQuoted(variable.getName()) + " resolved as " + singleQuoted(result.getFullyQualifiedName())); //
$NON-NLS-1$
                } else {
                        String message = "Could not resolve variable " + singleQuoted(variable.getName()) + ", of type "
                                        + singleQuoted(variable.getType().getName()) + " accessed in " + singleQuoted
(programUnit.getFullyQualifiedName());
                        getStatus(programUnit).addWarn(message);
                        log.error(message);
                }

                return result;
        }

        private void resolveVariablesTypesInBlock(IBlock block, IProgramUnit programUnit) throws Exception {
                for (Iterator iter = block.getVariables().iterator(); iter.hasNext();) {
                        IVariable variable = (IVariable) iter.next();
                        resolveVariableType(variable, programUnit);
                }
                for (Iterator iter = block.getBlocks().iterator(); iter.hasNext();) {
                        IBlock subBlock = (IBlock) iter.next();
                        resolveVariablesTypesInBlock(subBlock, programUnit);
                }
        }

        private void resolveMethodCallArgumentsTypesInMethodAccess(MethodCall.MethodAccess methodAccess, IMethod hostMethod, IBlock hostBlock,
                        IClass hostClass) throws Exception {
                if (methodAccess.isArgumentsProcessed())
                        return;
                methodAccess.setArgumentsProcessed(true);

                List resolvedArguments = new ArrayList();
                for (Iterator it = methodAccess.getArguments().iterator(); it.hasNext();) {
                        IProgramUnit resolvedProgramUnit = null;
                        Expression expression = null;
                        expression = (Expression) it.next();
                        if (expression != null) {
                                if (expression.getTypeSource() == Expression.TYPE_SOURCE_TYPE_CAST) {
                                        String varName = expression.getType().getName();

                                        if (varName.equals("null")) {
                                                log.debug("Expression " + singleQuoted(varName) + " resolved as UNKNOWN TYPE
```

```
(null argument)"); //$NON-NLS-1$
                                                                                resolvedProgramUnit = Class.CLASS_WRAPPER_NULL;
                                                                        }

                                                                        if (resolvedProgramUnit == null)
                                                                                resolvedProgramUnit = resolveType(expression.getType(), hostClass, false);
                                                                        if (resolvedProgramUnit == null)
                                                                                log.error("Could not resolve type for expression (TYPE CAST): " +
expression.getType().getName());
                                                                } else if (expression.getTypeSource() == Expression.TYPE_SOURCE_VARIABLE_ACCESS) {

                                                                        if (resolvedProgramUnit == null)
                                                                                if (expression.getReference().getTargetVariable() != null)
                                                                                        resolvedProgramUnit = expression.getReference().getTargetVariable().
getType().getProgramUnit();
                                                                        if (resolvedProgramUnit == null)
                                                                                log.error("Could not resolve type for expression (REFERENCE): " +
expression.getReference().getName());
                                                                } else if (expression.getTypeSource() == Expression.TYPE_SOURCE_METHOD_CALL) {
                                                                        // if the method call was not already resolved then
                                                                        // resolve it
                                                                        IMethodCall methodCall = expression.getMethodCall();
                                                                        if (!methodCall.isProcessed())
                                                                                resolveMethodCall(hostClass, hostMethod, hostBlock, methodCall);
                                                                        if (methodCall.isResolved())
                                                                                resolvedProgramUnit = methodCall.getTargetMethod().getReturningType().
getProgramUnit();
                                                                        if (resolvedProgramUnit == null)
                                                                                log.error("Could not resolve type for expression (METHOD CALL): " + methodCall);
                                                                }
                                                        }
                                                        resolvedArguments.add(resolvedProgramUnit);
                                                }
                                                List arguments = methodAccess.getArguments();
                                                arguments.clear();
                                                arguments.addAll(resolvedArguments);
                                        }

                                        private void resolveAllTypesInProgramUnit(IProgramUnit programUnit) throws Exception {

                                                if (typesResolveProcessingList.containsKey(programUnit))
                                                        return;
                                                typesResolveProcessingList.put(programUnit, programUnit);

                                                resolveParentProgramUnits(programUnit);
                                                // resolve varaibles types for variables in class members
                                                for (Iterator iter = programUnit.getAttributes().values().iterator(); iter.hasNext();) {
                                                        IVariable variable = ((IAttribute) iter.next()).getVariable();
                                                        resolveVariableType(variable, programUnit);
                                                }
                                                for (Iterator iter = programUnit.getMethods().iterator(); iter.hasNext();) {
                                                        IMethod method = (IMethod) iter.next();

                                                        // resolve types for method parameters
                                                        for (Iterator iterator = method.getParameters().iterator(); iterator.hasNext();) {
                                                                IVariable variable = ((IVariable) iterator.next());
                                                                resolveVariableType(variable, programUnit);
                                                        }

                                                        if (programUnit instanceof IClass) {
                                                                // resolve types for variables in block and sub blocks
                                                                resolveVariablesTypesInBlock(method.getBlock(), programUnit);

                                                                // resolve attribute accesses in the method
                                                                resolveAttributeAccessesInBlockRecursively(method.getBlock(), method, programUnit);
                                                        }

                                                        // resolve type for method returning type
                                                        IProgramUnit returnTypeProgramUnit = resolveProgramUnit(method.getReturningType().getName(), programUnit, false);
                                                        method.getReturningType().setProgramUnit(returnTypeProgramUnit);
                                                }
                                        }

                                        /*
                                         * private void resolveAllVariableAccessesInProgramUnit(IProgramUnit
                                         * programUnit) throws Exception { for (Iterator iter =
                                         * programUnit.getMethods().iterator(); iter.hasNext();) { IMethod method =
                                         * (IMethod) iter.next();
                                         *
                                         * if (programUnit instanceof IClass) { // resolve attribute accesses in the
                                         * method resolveAttributeAccessesInBlockRecursively(method.getBlock(),
                                         * method, programUnit); } } }
                                         */
                                        private MethodResolveResult resolveMethodInMembers(String methodName, List parameterTypes, IProgramUnit programUnit) throws Exception {
                                                MethodResolveResult result = new MethodResolveResult();
                                                int closestMethodsDistance = -1;

                                                resolveAllTypesInProgramUnit(programUnit);

                                                for (Iterator iter = programUnit.getMethods().iterator(); iter.hasNext();) {
```

```
                                        IMethod method = (IMethod) iter.next();
                                        // System.out.println("method: "+method.toString());
                                        if (result.approximate(method.compliesWith(methodName, parameterTypes), method))
                                                break;
                                }
                                return result;
        }

        private MethodResolveResult resolveMethodInProgramUnitHierarcy(String methodName, List parameterTypes, IProgramUnit hostProgramUnit)
                                throws Exception {
                        MethodResolveResult result = new MethodResolveResult();
                        if (hostProgramUnit instanceof IClass) {
                                // check members in direct parent classes
                                for (Iterator iter = ((IClass) hostProgramUnit).getSuperClasses().values().iterator(); iter.hasNext();) {
                                        IProgramUnit current = (IProgramUnit) iter.next();
                                        if (current != hostProgramUnit) {
                                                result.approximate(resolveMethodInMembers(methodName, parameterTypes, current));
                                                if (result.fullMatchFound())
                                                        return result;
                                        }

                                }
                                // check members in interfaces impemented directly
                                for (Iterator iter = ((IClass) hostProgramUnit).getSuperInterfaces().values().iterator(); iter.hasNext();) {
                                        result.approximate(resolveMethodInMembers(methodName, parameterTypes, (IProgramUnit) iter.next()));
                                        if (result.fullMatchFound())
                                                return result;
                                }

                                // recurse check in direct parent class(es)
                                for (Iterator iter = ((IClass) hostProgramUnit).getSuperClasses().values().iterator(); iter.hasNext();) {
                                        IProgramUnit current = (IProgramUnit) iter.next();
                                        if (current != hostProgramUnit)
                                                result.approximate(resolveMethodInProgramUnitHierarcy(methodName, parameterTypes, current));
                                        if (result.fullMatchFound())
                                                return result;
                                }
                                // recurse check in interfaces implemented
                                for (Iterator iter = ((IClass) hostProgramUnit).getSuperInterfaces().values().iterator(); iter.hasNext();) {
                                        IProgramUnit superInterface = (IProgramUnit) iter.next();
                                        if (superInterface != hostProgramUnit) {
                                                result.approximate(resolveMethodInProgramUnitHierarcy(methodName, parameterTypes,
superInterface));
                                                if (result.fullMatchFound())
                                                        return result;
                                        }
                                }
                        } else if (hostProgramUnit instanceof IInterface) {
                                // check member methods in the interfaces extended directly
                                for (Iterator iter = ((IInterface) hostProgramUnit).getSuperInterfaces().values().iterator(); iter.hasNext();) {
                                        IProgramUnit current = (IProgramUnit) iter.next();
                                        if (current != hostProgramUnit)
                                                result.approximate(resolveMethodInMembers(methodName, parameterTypes, current));
                                        if (result.fullMatchFound())
                                                return result;
                                }
                                // check recursively interfaces extended directly
                                for (Iterator iter = ((IInterface) hostProgramUnit).getSuperInterfaces().values().iterator(); iter.hasNext();) {
                                        IProgramUnit current = (IProgramUnit) iter.next();
                                        if (current != hostProgramUnit)
                                                result.approximate(resolveMethodInProgramUnitHierarcy(methodName, parameterTypes, current));
                                        if (result.fullMatchFound())
                                                return result;
                                }
                        }

                        return result;
        }

        private String extractTypeName(String typeName) {
                        int startPos = typeName.lastIndexOf(".");
                        int endPos = typeName.lastIndexOf("[");
                        startPos++;
                        if (endPos == -1)
                                endPos = typeName.length();
                        return typeName.substring(startPos, endPos);
        }

        private void resolveMethodCall(IClass hostClass, IMethod hostMethod, IBlock hostBlock, IMethodCall methodCall
        /* , HashMap resolvedTypesCache */) throws Exception {

                        ((MethodCall) methodCall).setProcessed(true);
                        IProgramUnit currentProgramUnit = hostClass;
                        List identifiers = ((MethodCall) methodCall).getIdentifiers();
                        String methodCallContent = ((MethodCall) methodCall).getParsedMethodName();

                        List joinedReferences = new ArrayList();
                        for (Iterator iter = identifiers.iterator(); iter.hasNext();) {
                                INamed named = (INamed) iter.next();
                                if (named instanceof MethodCall.ReferenceAccess) {
```

```
                                                MethodCall.ReferenceAccess referenceAccess = (ReferenceAccess) named;
                                                if (joinedReferences.size() > 0) {
                                                        Object element = joinedReferences.get(joinedReferences.size() - 1);
                                                        if (element instanceof MethodCall.ReferenceAccess) {
                                                                MethodCall.ReferenceAccess access = (ReferenceAccess) element;
                                                                ((Reference) access.getReference()).getIdentifiers().addAll(0,
                                                                                ((Reference) referenceAccess.getReference()).
getIdentifiers());
                                                        } else
                                                                joinedReferences.add(named);
                                                } else
                                                        joinedReferences.add(named);
                                        } else
                                                joinedReferences.add(named);
                                }
                                identifiers = joinedReferences;

                                log
                                                .debug("resolveMethodCall " + singleQuoted(identifiers.toString()) + " in " + singleQuoted
(hostClass.getFullyQualifiedName() + "." + hostMethod.getName())); //$NON-NLS-1$ //$NON-NLS-2$
                                boolean firstMemberAccess = true;
                                for (ListIterator iter = identifiers.listIterator(identifiers.size()); iter.hasPrevious();) {
                                        INamed memberAccess = (INamed) iter.previous();

                                        log.debug("Looking for " + singleQuoted(memberAccess) + " in " + singleQuoted(currentProgramUnit.getFullyQualifiedName())); //
$NON-NLS-1$ //$NON-NLS-2$

                                        IMethod resolvedMethod = null;
                                        IProgramUnit resolvedProgramUnit = null;
                                        if (memberAccess instanceof MethodCall.TypeCast) {
                                                resolvedProgramUnit = resolveProgramUnit(memberAccess.getName(), hostClass, true);
                                                if (resolvedProgramUnit != null)
                                                        log
                                                                        .debug("Resolved TypeCast " + singleQuoted(memberAccess.getName
()) + " as " + singleQuoted(resolvedProgramUnit.getFullyQualifiedName())); //$NON-NLS-1$ //$NON-NLS-2$
                                                else {
                                                        String message = "Could not resolve TypeCast " + singleQuoted(memberAccess.getName()) + " in " +
singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName()); //$NON-NLS-1$ //$NON-NLS-
2$
                                                        log.error(message);
                                                        getStatus(hostClass).addWarn(message);
                                                        return;
                                                }
                                        } else if (memberAccess instanceof MethodCall.MethodAccess) {
                                                // TODO: resolve types from expressions in the parameters list
                                                // when supported

                                                MethodCall.MethodAccess methodAccess = (MethodCall.MethodAccess) memberAccess;
                                                List parameterTypes = methodAccess.getArguments();

                                                // check if it is an empty constructor
                                                if (memberAccess instanceof MethodCall.ConstructorAccess) {
                                                        // if it is a default constructor - we need this explicit
                                                        // handling because there
                                                        // is no IMethod instance created for the default
                                                        // constructor
                                                        if (parameterTypes.size() == 0) {
                                                                resolvedProgramUnit = resolveProgramUnit(memberAccess.getName(),
currentProgramUnit, true);
                                                                if (resolvedProgramUnit != null) {
                                                                        if (resolvedProgramUnit instanceof Class)
                                                                                resolvedMethod = ((Class) resolvedProgramUnit).
getDefaultConstructor();
                                                                        else {
                                                                                // we have a constructor called over an
                                                                                // anonymous class that implements the interface
                                                                                // we do not currently support anonymous classes
                                                                                String message = "Unsupported anonymous class
definitions. Tried to resolve constructor " + singleQuoted(methodAccess.getName()) + " in " + singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName()); //
$NON-NLS-1$ //$NON-NLS-2$
                                                                                log.error(message);
                                                                                getStatus(hostClass).addWarn(message);
                                                                                return;
                                                                        }
                                                                } else {
                                                                        String message = "Could not resolve constructor " + singleQuoted
(methodAccess.getName()) + " in " + singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName()); //$NON-NLS-1$ //$NON-NLS-
2$
                                                                        log.error(message);
                                                                        getStatus(hostClass).addWarn(message);
                                                                        return;
                                                                }
                                                                // if it is a constructor with parameters
                                                        } else {
                                                                // locate the class where the constructor is declared
                                                                IProgramUnit constructorProgramUnit = resolveProgramUnit(methodAccess.getName
(), currentProgramUnit, true);

                                                                // resolve types for method invocation arguments in
                                                                // block
```

```
                                                                        // and
                                                                        // sub blocks
                                                                        resolveMethodCallArgumentsTypesInMethodAccess(methodAccess, hostMethod,
hostBlock, hostClass);

                                                                        if (constructorProgramUnit == null) {
                                                                                String message = "Could not resolve programUnit " + singleQuoted
(methodAccess.getName()) + " in " + singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName()); //$NON-NLS-1$ //$NON-NLS-
2$
                                                                                log.error(message);
                                                                                getStatus(hostClass).addWarn(message);
                                                                                return;
                                                                        }
                                                                        String constructorName = extractTypeName(methodAccess.getName());
                                                                        resolvedMethod = resolveMethodInMembers(constructorName, parameterTypes,
constructorProgramUnit).getMethod();

                                                                        if (resolvedMethod != null) {
                                                                                resolvedProgramUnit = constructorProgramUnit;
                                                                                log
                                                                                                .debug("Resolved constructor call "
                                                                                                                +
singleQuoted(methodCall.getName())
                                                                                                                + " in "
                                                                                                                +
singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName())
                                                                                                                + " as "
                                                                                                                +
singleQuoted(resolvedMethod.getEnclosingProgramUnit().getFullyQualifiedName() + "."

                                        + resolvedMethod));
                                                                        } else {
                                                                                String message = "Could not resolve constructor " + singleQuoted
(constructorName) + " in " + singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName()); //$NON-NLS-1$ //$NON-NLS-
2$
                                                                                log.error(message);
                                                                                getStatus(hostClass).addWarn(message);
                                                                                return;
                                                                        }
                                                                }
                                                        } else {
                                                                /*
                                                                 * if (methodAccess.getName().equals("cloneNode"))
                                                                 * System.out.println("debug here"); if
                                                                 * (methodAccess.getName().equals("removeChild"))
                                                                 * System.out.println("debug here"); if
                                                                 * (methodAccess.getName().equals("log"))
                                                                 * System.out.println("debug here");
                                                                 */
                                                                // resolve types for method invocation arguments in block
                                                                // and
                                                                // sub blocks
                                                                resolveMethodCallArgumentsTypesInMethodAccess(methodAccess, hostMethod, hostBlock,
hostClass);

                                                                MethodResolveResult resolveResult = new MethodResolveResult();
                                                                // search the method in methods of the current class
                                                                resolveResult.approximate(resolveMethodInMembers(methodAccess.getName(), parameterTypes,
currentProgramUnit));
                                                                // search the method in methods of ancestors of the current
                                                                // class
                                                                if (!resolveResult.fullMatchFound())
                                                                        resolveResult.approximate(resolveMethodInProgramUnitHierarcy
(methodAccess.getName(), parameterTypes,
                                                                                        currentProgramUnit));
                                                                // search the method in methods in the default root object
                                                                // (java.lang.Object)
                                                                if (!resolveResult.fullMatchFound())
                                                                        resolveResult.approximate(resolveMethodInMembers(methodAccess.getName(),
parameterTypes, javaLangObject));

                                                                resolvedMethod = resolveResult.getMethod();
                                                                if (resolvedMethod != null) {
                                                                        resolvedProgramUnit = resolvedMethod.getReturningType().getProgramUnit();
                                                                        log
                                                                                        .debug("Resolved method call " + singleQuoted
(methodCall.getName()) + " as " + singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName()) + " as " + singleQuoted
(resolvedMethod.getEnclosingProgramUnit().getFullyQualifiedName() + "." //$NON-NLS-1$ //$NON-NLS-2$

                                                + resolvedMethod));
                                                                } else {
                                                                        String message = "Could not resolve method call " + singleQuoted
(methodAccess.getName()) + " in " + singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName()); //$NON-NLS-1$ //$NON-NLS-
2$
                                                                        log.error(message);
                                                                        getStatus(hostClass).addWarn(message);
                                                                        return;
                                                                }
                                                        }
                                                } else if (memberAccess instanceof MethodCall.ReferenceAccess) {
```

```
                                        // if it is the first MemberAccess instance to be processed
                                        MethodCall.ReferenceAccess referenceAccess = (MethodCall.ReferenceAccess) memberAccess;

                                        IVariable resolvedVariable = null;
                                        if (firstMemberAccess) {
                                                resolvedVariable = resolveReference(referenceAccess.getReference(), hostClass, hostMethod,
hostBlock);
                                                // check if it is a type
                                                if (resolvedVariable == null)
                                                        resolvedProgramUnit = resolveProgramUnit(memberAccess.getName(),
currentProgramUnit, true);
                                        } else
                                                resolvedVariable = resolveVariableReferenceInProgramUnitHierarcy(memberAccess.getName(),
currentProgramUnit);
                                        // if it was not a type then check for variable

                                        if (resolvedProgramUnit == null) {
                                                if (resolvedVariable != null) {
                                                        resolvedProgramUnit = resolvedVariable.getType().getProgramUnit();
                                                        if (resolvedProgramUnit != null)
                                                                log
                                                                                .debug("Resolved local variable " +
singleQuoted(resolvedVariable.getName()) + " of type " + singleQuoted(resolvedProgramUnit.getFullyQualifiedName())); //$NON-NLS-1$ //$NON-NLS-2$
                                                } else {
                                                        String message = "Could not resolve variable " + singleQuoted
(memberAccess.getName()) + " referred in "
                                                                        + singleQuoted(hostClass.getFullyQualifiedName() +
"." + hostMethod.getName());
                                                        log.error(message);
                                                        getStatus(hostClass).addWarn(message);
                                                        return;
                                                }
                                        }
                                } else
                                        throw new Exception("Unknown type here: " + memberAccess); //$NON-NLS-1$

                                if (resolvedMethod != null) {
                                        ((MethodCall) methodCall).setTargetMethod(resolvedMethod);
                                }

                                firstMemberAccess = false;
                                // cache the resolvedProgramUnit
                                // if ((resolvedProgramUnit != null) &&
                                // (!resolvedTypesCache.containsKey(resolvedProgramUnit)))
                                // resolvedTypesCache.put(methodCall.getName(),
                                // resolvedProgramUnit);
                                if (resolvedProgramUnit != null)
                                        currentProgramUnit = resolvedProgramUnit;
                        }
                        if (methodCall.getTargetMethod() == null) {
                                String message = "Could not resolve method call " + singleQuoted(methodCall.getName()) + " in "
                                                + singleQuoted(hostClass.getFullyQualifiedName() + "." + hostMethod.getName());
                                log.error(message);
                                getStatus(hostClass).addWarn(message);
                        }
                }

                private void resolveMethodCallsInBlock(IClass hostClass, IMethod hostMethod, IBlock hostBlock) throws Exception {
                        // HashMap resolvedMethodCallsCache = new HashMap();

                        for (Iterator iter = hostBlock.getMethodCalls().iterator(); iter.hasNext();) {
                                MethodCall methodCall = (MethodCall) iter.next();
                                if (!methodCall.isProcessed())
                                        resolveMethodCall(hostClass, hostMethod, hostBlock, methodCall);
                        }
                        for (Iterator iter = hostBlock.getBlocks().iterator(); iter.hasNext();) {
                                resolveMethodCallsInBlock(hostClass, hostMethod, (IBlock) iter.next());
                        }
                }

                private void resolveAllMethodCallsInClass(IClass hostClass) throws Exception {
                        for (Iterator iter = hostClass.getMethods().iterator(); iter.hasNext();) {
                                IMethod method = (IMethod) iter.next();
                                log.debug("----------------------------------------------------------------------------------------------"); //$NON-NLS-1$
                                log.debug("Resolving method calls in method " + singleQuoted(hostClass.getFullyQualifiedName() + "." + method.getName())); //
$NON-NLS-1$ //$NON-NLS-2$
                                log.debug("----------------------------------------------------------------------------------------------"); //$NON-NLS-1$
                                resolveMethodCallsInBlock(hostClass, method, method.getBlock());
                        }
                }

                private void resolveAttributeAccessesInBlockRecursively(IBlock hostBlock, IMethod hostMethod, IProgramUnit hostProgramUnit)
                                throws Exception {
                        List codeSnippets = hostBlock.getCode();
                        HashMap cachedVariableAccesses = new HashMap();

                        for (Iterator iter = codeSnippets.listIterator(); iter.hasNext();) {
                                Object codeSnippet = iter.next();
                                if (codeSnippet instanceof IVariable) {
                                        IVariable variable = (IVariable) codeSnippet;
```

```
                                        if (cachedVariableAccesses.containsKey(variable.getName()))
                                                cachedVariableAccesses.remove(variable.getName());
                                        log.info("removed from cache: " + variable.getName());
                                } else if (codeSnippet instanceof IReference) {
                                        IReference variableReference = (IReference) codeSnippet;
                                        String referenceName = ((Reference) variableReference).getIdentifiers().toString();

                                        IVariable variable = null;
                                        if (cachedVariableAccesses.containsKey(referenceName)) {
                                                log.info("found cached: " + referenceName);
                                                variable = (IVariable) cachedVariableAccesses.get(referenceName);
                                        } else {
                                                variable = resolveReference(variableReference, (IClass) hostProgramUnit, hostMethod, hostBlock);
                                                if (variable != null)
                                                        cachedVariableAccesses.put(referenceName, variable);
                                        }

                                        if (hostProgramUnit.getName().equals("AbstractAnalysisPackage"))
                                                System.out.println("debug here");
                                        ((Reference) variableReference).setTargetVariable(variable);

                                        // set target attribute if it is a reference to a variable
                                        // member (attribute) of the current class
                                        if (variable != null) {
                                                String varName = variable.getName();
                                                if ((hostProgramUnit.getAttributes().containsKey(varName)) && (!variable.isDefinedLocally()))
                                                        ((Reference) variableReference).setTargetAttribute((IAttribute)
hostProgramUnit.getAttributes().get(varName));
                                        } else
                                                log.error("Could not resolve attribute acess: " + singleQuoted(referenceName) + " in "
                                                                + singleQuoted(hostProgramUnit.getFullyQualifiedName() + "." +
hostMethod.getName()));

                                        /*
                                         * if
                                         * (hostProgramUnit.getAttributes().containsKey(referenceName)) { //
                                         * fix required: should ignore attribute access of local //
                                         * variables // IAttribute attribute = (IAttribute)
                                         * hostProgramUnit.getAttributes().get(referenceName); //
                                         * IVariable localVariable =
                                         * resolveVariableReferenceInBlock(attribute.getVariable().getName(),
                                         * hostBlock); IVariable variable =
                                         * resolveLocalVariableReference(attribute.getVariable().getName(),(IClass)hostProgramUnit,
                                         * hostMethod, hostBlock); if (localVariable != null) {
                                         * ((Reference) variableReference).setLocalVariableAccess(true);
                                         * variableReference.setName(variableReference.getName() +
                                         * "_LOCAL_VAR"); } else ((Reference)
                                         * variableReference).setTargetAttribute(attribute); } else {
                                         * variableReference.setName(variableReference.getName() +
                                         * "_LOCAL_VAR"); }
                                         */
                                } else if (codeSnippet instanceof IBlock) {
                                        IBlock block = (IBlock) codeSnippet;
                                        resolveAttributeAccessesInBlockRecursively(block, hostMethod, hostProgramUnit);
                                }
                        }
                        // attributeAccesses.clear();
                        // attributeAccesses.addAll(resolvedAttributeAccesses.values());
                }

                private void resolveParentProgramUnits(IProgramUnit programUnit) throws Exception {
                        log.debug("Resolving super classes and interfaces ..."); //$NON-NLS-1$

                        if (programUnit instanceof IClass) {
                                Hashtable supers = ((IClass) programUnit).getSuperClasses();
                                Hashtable interfaces = ((IClass) programUnit).getSuperInterfaces();
                                Hashtable resolvedSupers = new Hashtable();
                                Hashtable resolvedInterfaces = new Hashtable();

                                for (Iterator iter = supers.keySet().iterator(); iter.hasNext();) {
                                        String superName = (String) iter.next();
                                        IProgramUnit superProgramUnit = resolveProgramUnit(superName, programUnit, false);
                                        if (superProgramUnit != null) {
                                                resolvedSupers.put(superProgramUnit.getFullyQualifiedName(), superProgramUnit);
                                                log
                                                                .debug("Resolved Super Class " + singleQuoted(superName) + " as " +
singleQuoted(superProgramUnit.getFullyQualifiedName())); //$NON-NLS-1$ //$NON-NLS-2$
                                        } else {
                                                String message = "Could not resolve Super Class " + singleQuoted(superName) + " referred in "
                                                                + singleQuoted(programUnit.getFullyQualifiedName());
                                                getStatus(programUnit).addWarn(message);
                                                getStatus(programUnit).addWarn(Messages.getString("ModelPreProcessor.40") + singleQuoted
(superName)); //$NON-NLS-1$
                                                log.error(message);
                                        }
                                }
                                for (Iterator iter = interfaces.keySet().iterator(); iter.hasNext();) {
                                        String superName = (String) iter.next();
                                        IProgramUnit superProgramUnit = resolveProgramUnit(superName, programUnit, false);
                                        if (superProgramUnit != null) {
```

```
                                                resolvedInterfaces.put(superProgramUnit.getFullyQualifiedName(), superProgramUnit);
                                                log
                                                                .debug("Resolved Interface " + singleQuoted(superName) + " as " +
singleQuoted(superProgramUnit.getFullyQualifiedName()) + "\t"); //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$
                                        } else {
                                                String message = "Could not resolve Interface " + singleQuoted(superName) + " referred in "
                                                                + singleQuoted(programUnit.getFullyQualifiedName());
                                                getStatus(programUnit).addWarn(message);
                                                getStatus(programUnit).addWarn(Messages.getString("ModelPreProcessor.46") + singleQuoted
(superName)); //$NON-NLS-1$
                                                log.error(message);
                                        }
                                }

                                ((IClass) programUnit).getSuperClasses().clear();
                                ((IClass) programUnit).getSuperClasses().putAll(resolvedSupers);
                                ((IClass) programUnit).getSuperInterfaces().clear();
                                ((IClass) programUnit).getSuperInterfaces().putAll(resolvedInterfaces);
                        } else {
                                Hashtable interfaces = ((IInterface) programUnit).getSuperInterfaces();
                                Hashtable resolvedInterfaces = new Hashtable();

                                for (Iterator iter = interfaces.keySet().iterator(); iter.hasNext();) {
                                        String superName = (String) iter.next();
                                        IProgramUnit superProgramUnit = resolveProgramUnit(superName, programUnit, false);
                                        if (superProgramUnit != null) {
                                                resolvedInterfaces.put(superProgramUnit.getFullyQualifiedName(), superProgramUnit);
                                                log
                                                                .debug("Resolved Super Interface " + singleQuoted(superName) + " as
" + singleQuoted(superProgramUnit.getFullyQualifiedName()) + "\t"); //$NON-NLS-1$ //$NON-NLS-2$ //$NON-NLS-3$
                                        } else {
                                                getStatus(programUnit).addWarn(Messages.getString("ModelPreProcessor.52") + singleQuoted
(superName)); //$NON-NLS-1$
                                        }
                                }

                                ((IInterface) programUnit).getSuperInterfaces().clear();
                                ((IInterface) programUnit).getSuperInterfaces().putAll(resolvedInterfaces);
                        }
                }

                private IProgramUnit getSupplementProgramUnit(String fullyQualifiedName) throws Exception {
                        IProgramUnit programUnit = null;
                        programUnit = (IProgramUnit) supplementProgramUnits.get(fullyQualifiedName);
                        if (programUnit == null) {
                                for (Iterator projectsIterator = model.getProjects().values().iterator(); projectsIterator.hasNext();) {
                                        IProject project = (IProject) projectsIterator.next();
                                        if (!project.isIncludedInReport()) {
                                                programUnit = modelExtractor.analyzeProgramUnit(fullyQualifiedName, project, ((Project) project).
getJarFile(),
                                                                new NullAnalysisMonitor());
                                                if (programUnit != null) {
                                                        supplementProgramUnits.put(fullyQualifiedName, programUnit);
                                                        break;
                                                }
                                        }
                                }
                                /*
                                 * if (programUnit != null)
                                 * resolveAllTypesInProgramUnit(programUnit);
                                 */
                        }

                        return programUnit;
                }

                private IProgramUnit resolveProgramUnit(String typeName, IProgramUnit hostProgramUnit, boolean resolveTypes) throws Exception {
                        IProgramUnit programUnit = resolveProgramUnitNoTypesResolved(typeName, hostProgramUnit);
                        if (resolveTypes && (programUnit != null))
                                resolveAllTypesInProgramUnit(programUnit);
                        return programUnit;
                }

                private IProgramUnit resolveProgramUnitNoTypesResolved(String typeName, IProgramUnit hostProgramUnit) throws Exception {
                        IProgramUnit resolvedProgramUnit = null;
                        String fullyQualifiedClassName = null;

                        // don't care if it is array, the type is important to know
                        if (typeName.endsWith("[]")) //$NON-NLS-1$
                                typeName = typeName.substring(0, typeName.indexOf("["));

                        if (typeName.indexOf(".") > -1) { // we have fully qualified
                                // //$NON-NLS-1$
                                fullyQualifiedClassName = typeName;

                                if (getProgramUnits().containsKey(fullyQualifiedClassName))
                                        return (IProgramUnit) getProgramUnits().getCompatibleResultForKey(fullyQualifiedClassName,
                                                        hostProgramUnit.getEnclosingNamespace().getEnclosingProject());
                                else
```

```
                                        // check if the class is an inner class - brute force check
                                        if (resolvedProgramUnit == null) {
                                                if (typeName.indexOf(".") > -1) {
                                                        for (SubIterator iter = getProgramUnits().valuesSubIterator(); iter.hasNext();) {
                                                                IProgramUnit element = (IProgramUnit) iter.next();
                                                                if (element.getFullyQualifiedName().endsWith(fullyQualifiedClassName))
                                                                        return element;
                                                        }

                                                }
                                        }

                                        log.debug("Detected fully-qualified type " + singleQuoted(fullyQualifiedClassName)); //$NON-NLS-1$
                                } else {
                                        // we have class name only

                                        // check whether hostClass is an inner class and the typeName
                                        // matches to it or other siblings if any
                                        IProgramUnit enclosingProgramUnit = hostProgramUnit.getEnclosingProgramUnit();
                                        if (enclosingProgramUnit != null) {
                                                for (Iterator iter = enclosingProgramUnit.getChildProgramUnits().iterator(); iter.hasNext();) {
                                                        IProgramUnit element = (IProgramUnit) iter.next();
                                                        if (element.getName().endsWith(Class.INTERNAL_CLASS_DELIMITER + typeName)) {
                                                                log.debug("Inner class found " + singleQuoted(element.getFullyQualifiedName())); //
$NON-NLS-1$
                                                                return element;
                                                        }
                                                }
                                        }

                                        // check whether i
                                        // check the imports
                                        if (fullyQualifiedClassName == null) {
                                                IImports imports = hostProgramUnit.getImports();
                                                if (imports != null) {
                                                        for (Iterator iterator = imports.getImports().values().iterator(); iterator.hasNext();) {
                                                                IImport _import = (IImport) iterator.next();
                                                                // System.out.print("Import: " + _import.getName());
                                                                if (!_import.getName().endsWith("*")) { //$NON-NLS-
1$
                                                                        // fully qualified type is imported
                                                                        if (_import.getName().endsWith("." + typeName)) { //$NON-NLS-1$
                                                                                fullyQualifiedClassName = _import.getName();
                                                                                log.debug("Import matched for " + singleQuoted
(_import.getName())); //$NON-NLS-1$
                                                                                break;
                                                                        }
                                                                } else { // wildcard is imported
                                                                        // REMOVE THIS CODE if
                                                                        // (_import.getName().endsWith("node.*"))
                                                                        // System.out.println("debug here");
                                                                        String key = _import.getName().substring(0, _import.getName().length
() - 2) + "." + typeName; //$NON-NLS-1$
                                                                        if (getProgramUnits().containsKey(key)) {
                                                                                fullyQualifiedClassName = ((IProgramUnit)
getProgramUnits().getCompatibleResultForKey(key,

hostProgramUnit.getEnclosingNamespace().getEnclosingProject())).getFullyQualifiedName();
                                                                                log.debug("Import matched for " + singleQuoted
(fullyQualifiedClassName)); //$NON-NLS-1$
                                                                                break;
                                                                        } else {
                                                                                resolvedProgramUnit = getSupplementProgramUnit
(key);
                                                                                if (resolvedProgramUnit != null)
                                                                                        return resolvedProgramUnit;
                                                                        }
                                                                }
                                                                // System.out.println();
                                                        }
                                                }
                                        }
                                        // maybe the type is in the same package
                                        if (fullyQualifiedClassName == null) {
                                                String key = hostProgramUnit.getEnclosingNamespace().getName() + "." + typeName; //$NON-NLS-1$
                                                if (getProgramUnits().containsKey(key)) {
                                                        fullyQualifiedClassName = ((IProgramUnit) getProgramUnits().getCompatibleResultForKey(key,
                                                                        hostProgramUnit.getEnclosingNamespace().getEnclosingProject())).
getFullyQualifiedName();
                                                        log.debug("Class found in the same package: " + singleQuoted(fullyQualifiedClassName)); //$NON-
NLS-1$
                                                }
                                        }
                                        // maybe the type is inner for the current class
                                        if (fullyQualifiedClassName == null) {
                                                String key = hostProgramUnit.getFullyQualifiedName() + Class.INTERNAL_CLASS_DELIMITER + typeName; //
$NON-NLS-1$
                                                if (getProgramUnits().containsKey(key)) {
                                                        fullyQualifiedClassName = ((IProgramUnit) getProgramUnits().getCompatibleResultForKey(key,
                                                                        hostProgramUnit.getEnclosingNamespace().getEnclosingProject())).
```

```
getFullyQualifiedName();
                                                                        log.debug("Inner class found " + singleQuoted(fullyQualifiedClassName)); //$NON-NLS-1$
                                                }
                                        }
                                        if (fullyQualifiedClassName == null) {
                                                // if the type cannot be found in the imports or in the same
                                                // package
                                                // then it must be in the java.lang package

                                                // convert primitive type to its wrapper class
                                                if (typeName.equals("int")) //$NON-NLS-1$
                                                        typeName = "Integer"; //$NON-NLS-1$
                                                else if (typeName.equals("long") || typeName.equals("longint")) //$NON-NLS-1$ //$NON-NLS-2$
                                                        typeName = "Long"; //$NON-NLS-1$
                                                else if (typeName.equals("short") || typeName.equals("shortint")) //$NON-NLS-1$ //$NON-NLS-2$
                                                        typeName = "Short"; //$NON-NLS-1$
                                                else if (typeName.equals("double")) //$NON-NLS-1$
                                                        typeName = "Double"; //$NON-NLS-1$
                                                else if (typeName.equals("float")) //$NON-NLS-1$
                                                        typeName = "Float"; //$NON-NLS-1$
                                                else if (typeName.equals("boolean")) //$NON-NLS-1$
                                                        typeName = "Boolean"; //$NON-NLS-1$
                                                else if (typeName.equals("byte")) //$NON-NLS-1$
                                                        typeName = "Byte"; //$NON-NLS-1$
                                                else if (typeName.equals("char")) //$NON-NLS-1$
                                                        typeName = "Character"; //$NON-NLS-1$
                                                else if (typeName.equals("void")) //$NON-NLS-1$
                                                        typeName = "Void"; //$NON-NLS-1$

                                                fullyQualifiedClassName = "java.lang." + typeName; //$NON-NLS-1$

                                        }

                                }

                                if (fullyQualifiedClassName != null)
                                        if (getProgramUnits().containsKey(fullyQualifiedClassName))
                                                resolvedProgramUnit = (IProgramUnit) getProgramUnits().getCompatibleResultForKey(fullyQualifiedClassName,
                                                                hostProgramUnit.getEnclosingNamespace().getEnclosingProject());
                                        else
                                                resolvedProgramUnit = getSupplementProgramUnit(fullyQualifiedClassName);

                                if (resolvedProgramUnit == null) {
                                        String message = "Could not resolve programUnit " + singleQuoted(typeName) + " referred in "
                                                        + singleQuoted(hostProgramUnit.getFullyQualifiedName());
                                        getStatus(hostProgramUnit).addWarn(message);
                                        log.error(message);

                                }
                                return resolvedProgramUnit;
                }

                public Status getStatus(IProgramUnit programUnit) {
                                Status status = (Status) unitsStatus.get(programUnit);
                                if (status == null)
                                        unitsStatus.put(programUnit, new Status());
                                return (Status) unitsStatus.get(programUnit);
                }

                /**
                 * @return Returns the methods.
                 */
                public AnalysisHashMap getMethods() {
                                return methods;
                }

                /**
                 * @return Returns the namespaces.
                 */
                public AnalysisHashMap getNamespaces() {
                                return namespaces;
                }

                /**
                 * @return Returns the programUnits.
                 */
                public AnalysisHashMap getProgramUnits() {
                                return programUnits;
                }

                public Status getPreProcessorStatus() {
                                return preProcessorStatus;
                }

}
```

Пакет org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst

```
package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;
```

```
import java.util.List;

import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TIdentifier;


public class Attribute extends org.bog.lachesis.metamodel.impl.AbstractAttribute {

   public Attribute(String name) {
                                super(name);
                }

   public void do1() {

   }
   /* (non-Javadoc)
    * @see bog.lachesis.model.independent.AbstractNamed#parse()
    */
   public void parse(PType type,PVariableDeclaratorId variableDeclaratorId,  TIdentifier identifier, List modifiers) throws Exception
   {
      Variable variable = new Variable(identifier.getText());
      variable.setDefinedLocally(false);
      variable.parse(variableDeclaratorId, type);
      variable.setModified(new Modified(modifiers));
      setName(identifier.getText());
      setVariable(variable);

      /*if (lp.checkType(currentNode,VARIABLE_DEF))
      {
         currentNode++;
         Modified modified = new Modified();
         currentNode = modified.parse(lp,currentNode,parseMonitor);
         setModified(modified);
         Variable variable = new Variable("");
         currentNode = variable.parse(lp,currentNode, parseMonitor);
         setVariable(variable);
         setName(getVariable().getName());
      }
      return currentNode;
      */
   }
}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;


import java.lang.reflect.InvocationTargetException;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.impl.AbstractBlock;
import org.bog.lachesis.profiler.Profiler;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AAssignVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ACatchClause;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AExpressionCastExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFormalParameter;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIdentifierVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ALocalVariableDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorVariableDeclarators;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorsVariableDeclarators;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PBlock;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PMethodInvocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclarators;

public class Block extends AbstractBlock {
                private static Logger log = Logger.getLogger(Block.class);

                private final static String IF = "IF";

                private final static String PROFILED_TASK = "Block.parse";

                int indexInEnclosingBlock = -1;

                private void parsePVariableDeclaratorId(PVariableDeclaratorId variableDeclaratorId, PType type, List modifiers) throws Exception {
                                if (variableDeclaratorId instanceof AVariableDeclaratorIdVariableDeclaratorId) {
                                                AVariableDeclaratorIdVariableDeclaratorId declaratorId = (AVariableDeclaratorIdVariableDeclaratorId) variableDeclaratorId;
                                                parsePVariableDeclaratorId(declaratorId.getVariableDeclaratorId(), type, modifiers);
                                } else if (variableDeclaratorId instanceof AIdentifierVariableDeclaratorId) {
                                                AIdentifierVariableDeclaratorId declaratorId = (AIdentifierVariableDeclaratorId) variableDeclaratorId;
```

```
                        Variable variable = new Variable("");
                        variable.parse(variableDeclaratorId, type);
                        addVariable(variable);
                } else
                        throw new Exception("Unknown type here: " + variableDeclaratorId);
}

private void parsePVariableDeclarators(PVariableDeclarators variableDeclarators, PType type, List modifiers) throws Exception {
                PVariableDeclarator variableDeclarator = null;
                if (variableDeclarators instanceof AVariableDeclaratorVariableDeclarators) {
                        variableDeclarator = ((AVariableDeclaratorVariableDeclarators) variableDeclarators).getVariableDeclarator();
                } else if (variableDeclarators instanceof AVariableDeclaratorsVariableDeclarators) {
                        AVariableDeclaratorsVariableDeclarators variableDeclarators2 = (AVariableDeclaratorsVariableDeclarators) variableDeclarators;
                        parsePVariableDeclarators(variableDeclarators2.getVariableDeclarators(), type, modifiers);
                        variableDeclarator = variableDeclarators2.getVariableDeclarator();
                } else
                        throw new Exception("Unknown type here: " + variableDeclarators);

                PVariableDeclaratorId declaratorId = null;
                if (variableDeclarator instanceof AVariableDeclaratorIdVariableDeclarator) {
                        declaratorId = ((AVariableDeclaratorIdVariableDeclarator) variableDeclarator).getVariableDeclaratorId();
                } else if (variableDeclarator instanceof AAssignVariableDeclarator) {
                        declaratorId = ((AAssignVariableDeclarator) variableDeclarator).getVariableDeclaratorId();
                } else
                        throw new Exception("Unknoen type here: " + variableDeclarator);
                parsePVariableDeclaratorId(declaratorId, type, modifiers);

}

String temp = "";

private String levelStr(int level) {
                StringBuffer buffer = new StringBuffer();
                for (int i = 0; i < level; i++)
                        buffer.append(".");
                return buffer.toString();
}

public void parse(Object node, boolean initialRun, int level) throws Exception {
                String current = node.toString();
                if (!temp.equals(current)) {
                        log.debug("<span class=styleBlockClass >[Block]" + levelStr(level)
                                        + node.getClass().getName().substring(node.getClass().getName().lastIndexOf(".") + 1)
                                        + "</span> <span class=styleBlockValue >" + current + "</span><br>");
                        level++;
                        temp = current;
                }
                // + " -> " + node.toString());
                // System.out.println(node.getClass().getName().substring(node.getClass().getName().lastIndexOf(".")+1)
                // + " -> " + node.toString());
                if (node instanceof ACatchClause) {
                        ACatchClause catchClause = (ACatchClause) node;
                        Block block = new Block();

                        Parameter parameter = new Parameter("");
                        parameter.parse((AFormalParameter) catchClause.getFormalParameter());
                        block.addVariable(parameter);

                        block.parse(catchClause.getBlock());
                        block.indexInEnclosingBlock = getCode().size();
                        addBlock(block);
                        // stop recursive parsing here
                        return;
                } else if ((node instanceof PBlock) && (!initialRun)) {
                        Block block = new Block();
                        block.parse((PBlock) node);
                        block.indexInEnclosingBlock = getCode().size();
                        addBlock(block);
                        // stop recursive parsing here
                        return;
                } else if (node instanceof PMethodInvocation) {
                        getOperators().incItemUsage("METHOD_CALL");
                        MethodCall methodCall = new MethodCall("");
                        methodCall.setRawName(node.toString());
                        methodCall.parse(node, this, level);
                        // stop recursive parsing here
                        return;
                } else if (node instanceof ALocalVariableDeclaration) {
                        ALocalVariableDeclaration localVariableDeclaration = (ALocalVariableDeclaration) node;
                        parsePVariableDeclarators(localVariableDeclaration.getVariableDeclarators(), localVariableDeclaration.getType(),
                                        localVariableDeclaration.getModifier());
                } else if (node instanceof AExpressionCastExpression) {
                        /**
                         * ignore the cast itself but parse the expression that was casted
                         */
                        parse(((AExpressionCastExpression)node).getUnaryExpressionNotPlusMinus(), false, level+1);
                        // stop recursive parsing here
                        return;
                } else
                        BlockContent.parseBlockContent(this, node, initialRun, level + 1);
```

```
                        java.lang.reflect.Method[] methods = node.getClass().getMethods();
                        for (int i = 0; i < methods.length; i++) {
                                java.lang.reflect.Method method = methods[i];
                                if (method.getName().startsWith("get")
                                                && (method.getReturnType() != null)
                                                && (method.getReturnType().getPackage() != null)
                                                && (method.getReturnType().getPackage().getName().startsWith("org.bog.lachesis") ||
method.getReturnType().getName()
                                                                                                .endsWith("List"))) {
                                        try {
                                                Object result = method.invoke(node, null);
                                                // System.out.println("Invoked: "+node.getClass()+"."
                                                // +method.getName());
                                                if (result != null)
                                                        if (result instanceof Collection)
                                                                for (Iterator iter = ((Collection) result).iterator(); iter.hasNext();)
                                                                        parse(iter.next(), false, level);
                                                        else
                                                                parse(result, false, level);
                                        } catch (IllegalArgumentException e) {
                                                e.printStackTrace();
                                        } catch (IllegalAccessException e) {
                                                e.printStackTrace();
                                        } catch (InvocationTargetException e) {
                                                e.printStackTrace();
                                        }
                                }

                        }
                } /*
                        * (non-Javadoc)
                        *
                        * @see bog.lachesis.model.independent.AbstractNamed#parse()
                        */

                public void parse(PBlock methodBody) throws Exception {
                        Profiler.startProfile(PROFILED_TASK);
                        parse(methodBody, true, 0);
                        Profiler.stopProfile(PROFILED_TASK);
                }

                public int getIndexInEnclosingBlock() {
                        return indexInEnclosingBlock;
                }

}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.util.ArrayList;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ACaseSwitchLabel;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ACatchClause;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ADefaultSwitchLabel;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ADimExpr;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ADivMultiplicativeExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ADoStatement;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFinally;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AForStatement;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AForStatementNoShortIf;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIfThenElseStatement;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIfThenElseStatementNoShortIf;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIfThenStatement;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AMinusAdditiveExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AModMultiplicativeExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ANameLeftHandSide;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ANamePostfixExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APlusAdditiveExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APostDecrementExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APostIncrementExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APreDecrementExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APreIncrementExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APrimaryFieldAccess;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AStarMultiplicativeExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASuperExplicitConstructorInvocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AThisExplicitConstructorInvocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AThisPrimaryNoNewArray;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AWhileStatement;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AWhileStatementNoShortIf;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PCastExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PPrimaryNoNewArray;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TAnd;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TBitAnd;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TBitAndAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TBitComplement;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TBitOr;
```

```
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TBitOrAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TBitXor;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TBitXorAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TCharacterLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TComplement;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TDecimalIntegerLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TDivAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TEq;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TFalse;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TFloatingPointLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TGt;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TGteq;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.THexIntegerLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TIdentifier;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TInstanceof;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TLt;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TLteq;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TMinusAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TModAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TNeq;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TNew;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TNull;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TOr;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TPlusAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TQuestion;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TShiftLeft;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TShiftLeftAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TSignedShiftRight;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TStarAssign;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TStringLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TSuper;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TThis;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TTrue;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TUnsignedShiftRight;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TUnsignedShiftRightAssign;

public class BlockContent {
	private static Logger log = Logger.getLogger(Block.class);

	static String temp = "";

	private static String levelStr(int level) {
		StringBuffer buffer = new StringBuffer();
		for (int i = 0; i < level; i++)
			buffer.append(".");
		return buffer.toString();
	}

	public static List parseBlockContent(Block enclosingBlock, Object node, boolean initialRun, int level) throws Exception {
		List result = null;
		String current = node.toString();
		if (!temp.equals(current)) {
			log.debug("<span class=styleBlockContentClass >[BlockContent]" + levelStr(level)
					+ node.getClass().getName().substring(node.getClass().getName().lastIndexOf(".") + 1)
					+ "</span> <span class=styleBlockContentValue >" + current + "</span><br>");
			level++;
			temp = current;
		}
		// + " -> " + node.toString());
		// System.out.println(node.getClass().getName().substring(node.getClass().getName().lastIndexOf(".")+1)
		// + " -> " + node.toString());
		/*
		 * if ((node instanceof PBlock) && (!initialRun)) { Block block = new
		 * Block(); block.parse((PBlock) node); block.indexInEnclosingBlock =
		 * enclosingBlock.getBlocks().size(); enclosingBlock.addBlock(block);
		 * return; } else if (node instanceof PMethodInvocation) {
		 * enclosingBlock.getOperators().incItemUsage("METHOD_CALL"); MethodCall
		 * methodCall = new MethodCall(node.toString().replaceAll(" ", ""));
		 * methodCall.parsePMethodInvocation((PMethodInvocation) node,
		 * enclosingBlock, level); return; } else if (node instanceof
		 * ALocalVariableDeclaration) { ALocalVariableDeclaration
		 * localVariableDeclaration = (ALocalVariableDeclaration) node;
		 * parsePVariableDeclarators(enclosingBlock,
		 * localVariableDeclaration.getVariableDeclarators(),
		 * localVariableDeclaration.getType(),
		 * localVariableDeclaration.getModifier()); } else
		 */

		if ((node instanceof ANameLeftHandSide) || (node instanceof ANamePostfixExpression) || (node instanceof APrimaryFieldAccess)) {
			//if ((node instanceof ANamePostfixExpression) && (node.toString().startsWith("I")))
				//System.out.println("debug here");
			// should we go for qualified name->TIdentifier approach here?
			//System.out.println("Reference: "+node.toString()+" from: "+node.getClass().getName());
			Reference attributeAccess = new Reference(node.toString());
			attributeAccess.parse(node);
			if (attributeAccess.getIdentifiers().size() > 0) {
				enclosingBlock.addAttributeAccess(attributeAccess);
				if (result == null)
					result = new ArrayList();
				result.add(attributeAccess);
```

```
                }
        } else if (node instanceof PPrimaryNoNewArray) {
                if (node instanceof AThisPrimaryNoNewArray) {
                        Reference attributeAccess = new Reference(node.toString());
                        attributeAccess.getIdentifiers().add("this");

                        enclosingBlock.addAttributeAccess(attributeAccess);
                        if (result == null)
                                result = new ArrayList();
                        result.add(attributeAccess);

                }
        } else if ((node instanceof AIfThenStatement) || (node instanceof AIfThenElseStatement)
                        || (node instanceof AIfThenElseStatementNoShortIf) || (node instanceof ACaseSwitchLabel)
                        || (node instanceof ADefaultSwitchLabel) || (node instanceof ACatchClause) || (node instanceof AFinally)
                        || (node instanceof AForStatement) || (node instanceof AForStatementNoShortIf) || (node instanceof
AWhileStatement)
                        || (node instanceof AWhileStatementNoShortIf) || (node instanceof ADoStatement) || (node instanceof
AWhileStatement)
                        || (node instanceof AWhileStatement) || (node instanceof AWhileStatement))
                enclosingBlock.getBranches().incItemUsage("IF");
        else if (node instanceof APlusAdditiveExpression)
                enclosingBlock.getOperators().incItemUsage("PLUS");
        else if (node instanceof AMinusAdditiveExpression)
                enclosingBlock.getOperators().incItemUsage("MINUS");
        else if (node instanceof APostIncrementExpression)
                enclosingBlock.getOperators().incItemUsage("POST_INC");
        else if (node instanceof APostDecrementExpression)
                enclosingBlock.getOperators().incItemUsage("POST_DEC");
        else if (node instanceof APreIncrementExpression)
                enclosingBlock.getOperators().incItemUsage("PRE_INC");
        else if (node instanceof APreDecrementExpression)
                enclosingBlock.getOperators().incItemUsage("PRE_DEC");
        else if (node instanceof AStarMultiplicativeExpression)
                enclosingBlock.getOperators().incItemUsage("STAR");
        else if (node instanceof ADivMultiplicativeExpression)
                enclosingBlock.getOperators().incItemUsage("DIV");
        else if (node instanceof AModMultiplicativeExpression)
                enclosingBlock.getOperators().incItemUsage("MOD");
        else if (node instanceof TInstanceof)
                enclosingBlock.getOperators().incItemUsage("LITERAL_instanceof");
        else if (node instanceof TShiftLeft)
                enclosingBlock.getOperators().incItemUsage("SL");
        else if (node instanceof TSignedShiftRight)
                enclosingBlock.getOperators().incItemUsage("SSR");
        else if (node instanceof TUnsignedShiftRight)
                enclosingBlock.getOperators().incItemUsage("USR");
        else if (node instanceof TGt)
                enclosingBlock.getOperators().incItemUsage("GT");
        else if (node instanceof TLt)
                enclosingBlock.getOperators().incItemUsage("LT");
        else if (node instanceof TLteq)
                enclosingBlock.getOperators().incItemUsage("LE");
        else if (node instanceof TGteq)
                enclosingBlock.getOperators().incItemUsage("GE");
        else if (node instanceof TEq)
                enclosingBlock.getOperators().incItemUsage("EQUAL");
        else if (node instanceof TNeq)
                enclosingBlock.getOperators().incItemUsage("NOT_EQUAL");
        else if (node instanceof TOr)
                enclosingBlock.getOperators().incItemUsage("LOG_OR");
        else if (node instanceof TAnd)
                enclosingBlock.getOperators().incItemUsage("LOG_AND");
        else if (node instanceof TComplement)
                enclosingBlock.getOperators().incItemUsage("LOG_NOT");
        else if (node instanceof TQuestion) {
                enclosingBlock.getOperators().incItemUsage("TERNARY");
                enclosingBlock.getBranches().incItemUsage("TERNARY");
        }
        // Bitwise
        else if (node instanceof TBitAnd)
                enclosingBlock.getOperators().incItemUsage("BIT_AND");
        else if (node instanceof TBitComplement)
                enclosingBlock.getOperators().incItemUsage("BIT_NOT");
        else if (node instanceof TBitOr)
                enclosingBlock.getOperators().incItemUsage("BIT_OR");
        else if (node instanceof TBitXor)
                enclosingBlock.getOperators().incItemUsage("BIT_XOR");

        // Assignment
        else if (node instanceof TAssign)
                enclosingBlock.getOperators().incItemUsage("ASSIGN");
        else if (node instanceof TPlusAssign)
                enclosingBlock.getOperators().incItemUsage("PLUS_ASSIGN");
        else if (node instanceof TMinusAssign)
                enclosingBlock.getOperators().incItemUsage("MINUS_ASSIGN");
        else if (node instanceof TStarAssign)
                enclosingBlock.getOperators().incItemUsage("STAR_ASSIGN");
        else if (node instanceof TDivAssign)
                enclosingBlock.getOperators().incItemUsage("DIV_ASSIGN");
```

```
                        else if (node instanceof TModAssign)
                                enclosingBlock.getOperators().incItemUsage("MOD_ASSIGN");
                        else if (node instanceof TBitAndAssign)
                                enclosingBlock.getOperators().incItemUsage("BIT_AND_ASSIGN");
                        else if (node instanceof TBitOrAssign)
                                enclosingBlock.getOperators().incItemUsage("BIT_OR_ASSIGN");
                        else if (node instanceof TSignedShiftRight)
                                enclosingBlock.getOperators().incItemUsage("SSR_ASSIGN");
                        else if (node instanceof TUnsignedShiftRightAssign)
                                enclosingBlock.getOperators().incItemUsage("USR_ASSIGN");
                        else if (node instanceof TShiftLeftAssign)
                                enclosingBlock.getOperators().incItemUsage("SL_ASSIGN");
                        else if (node instanceof TBitXorAssign)
                                enclosingBlock.getOperators().incItemUsage("BIT_XOR_ASSIGN");
                        else if (node instanceof PCastExpression)
                                enclosingBlock.getOperators().incItemUsage("TYPECAST");
                        else if (node instanceof ADimExpr)
                                enclosingBlock.getOperators().incItemUsage("INDEX_OP");

                        // methods
                        else if (node instanceof ASuperExplicitConstructorInvocation)
                                enclosingBlock.getOperators().incItemUsage("SUPER_CTOR_CALL");
                        else if (node instanceof AThisExplicitConstructorInvocation)
                                enclosingBlock.getOperators().incItemUsage("CTOR_CALL");
                        // ARRAY_INIT: enclosingBlock.getOperators().incItemUsage("ARRAY_INIT");

                        //
                        // operands
                        //
                        else if (node instanceof TIdentifier) {

                                enclosingBlock.getOperands().incItemUsage("IDENTIFIER:" + node.toString());
                        } else if (node instanceof TThis)
                                enclosingBlock.getOperands().incItemUsage("LITERAL_this:" + node.toString());
                        else if (node instanceof TSuper)
                                enclosingBlock.getOperands().incItemUsage("LITERAL_super:" + node.toString());
                        else if (node instanceof TTrue)
                                enclosingBlock.getOperands().incItemUsage("LITERAL_true:" + node.toString());
                        else if (node instanceof TFalse)
                                enclosingBlock.getOperands().incItemUsage("LITERAL_false:" + node.toString());
                        else if (node instanceof TNull)
                                enclosingBlock.getOperands().incItemUsage("LITERAL_null:" + node.toString());
                        else if (node instanceof TNew)
                                enclosingBlock.getOperands().incItemUsage("LITERAL_new:" + node.toString());
                        else if (node instanceof TDecimalIntegerLiteral)
                                enclosingBlock.getOperands().incItemUsage("DECIMAL_INT_LITERAL:" + node.toString());
                        else if (node instanceof TCharacterLiteral)
                                enclosingBlock.getOperands().incItemUsage("CHAR_LITERAL:" + node.toString());
                        else if (node instanceof TStringLiteral)
                                enclosingBlock.getOperands().incItemUsage("STRING_LITERAL:" + node.toString());
                        else if (node instanceof TFloatingPointLiteral)
                                enclosingBlock.getOperands().incItemUsage("FLOAT_LITERAL:" + node.toString());
                        else if (node instanceof THexIntegerLiteral)
                                enclosingBlock.getOperands().incItemUsage("HEX_INTEGER_LITERAL:" + node.toString());

                        return result;
                }

}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.metamodel.ILinesOfCode;
import org.bog.lachesis.metamodel.ILocation;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IResourceLocateable;
import org.bog.lachesis.metamodel.impl.AbstractClass;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AAssignVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassBody;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassDeclarationClassMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassMemberDeclarationClassBodyDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassOrInterfaceType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AConstructorDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AConstructorDeclarationClassBodyDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFieldDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFieldDeclarationClassMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIdentifierVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInterfaceType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInterfaceTypeInterfaceTypeList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInterfaceTypeListInterfaceTypeList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInterfaces;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AMethodDeclaration;
```

```
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AMethodDeclarationClassMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASuper;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorVariableDeclarators;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorsVariableDeclarators;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PClassBodyDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PClassDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PClassMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PInterfaceTypeList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclarators;

public class Class extends AbstractClass implements IResourceLocateable, ILinesOfCode {

	private final static Package PACKAGE_JAVA_LANG = new Package("java.lang");

	// these are only used for comparison issues (should not be added to the
	// model!!!)
	public final static Class CLASS_DOUBLE = new Class("Double", PACKAGE_JAVA_LANG);

	public final static Class CLASS_FLOAT = new Class("Float", PACKAGE_JAVA_LANG);

	public final static Class CLASS_LONG = new Class("Long", PACKAGE_JAVA_LANG);

	public final static Class CLASS_INT = new Class("Int", PACKAGE_JAVA_LANG);

	public final static Class CLASS_SHORT = new Class("Short", PACKAGE_JAVA_LANG);

	public final static Class CLASS_BYTE = new Class("Byte", PACKAGE_JAVA_LANG);

	public final static Class CLASS_CHARACTER = new Class("Character", PACKAGE_JAVA_LANG);

	public final static Class CLASS_STRING = new Class("String", PACKAGE_JAVA_LANG);

	public final static Class CLASS_BOOLEAN = new Class("Boolean", PACKAGE_JAVA_LANG);

	public final static Class CLASS_WRAPPER_NULL = new Class("null", PACKAGE_JAVA_LANG);

	public final static Class CLASS_CLASS = new Class("Class", PACKAGE_JAVA_LANG);

	public final static Class CLASS_ARRAY = new Class("ArrayWrapper", PACKAGE_JAVA_LANG);

	public final static String INTERNAL_CLASS_DELIMITER = ".";

	private static Logger log = Logger.getLogger(Class.class);

	private ILocation absoluteLocation;

	private int firstLineNumber = 0;

	private int lastLineNumber = 0;

	private IMethod defaultConstructor = null;

	static {
		Method method = new Method("clone");
		Attribute attribute = new Attribute("length");
		Variable variable =  new Variable("length");
		Type type =  new Type("int");
		type.setProgramUnit(CLASS_INT);
		variable.setType(type);
		attribute.setVariable(variable);
		CLASS_ARRAY.addMethod(method);
		CLASS_ARRAY.addAttribute(attribute);
	}

	/**
	 * @param name
	 */
	public Class(String name, ILocation absoluteLocation) {
		super(name);
		this.absoluteLocation = absoluteLocation;
	}

	private Class(String name, Package thePackage) {
		super(name);
		setEnclosingNamespace(thePackage);
	}

	public IMethod getDefaultConstructor() {
		if (defaultConstructor == null) {
			defaultConstructor = new Constructor(getName());
			Type returnType = new Type(getName());
			returnType.setProgramUnit(this);
			defaultConstructor.setReturningType(returnType);
			defaultConstructor.setEnclosingProgramUnit(this);
		}
```

```
                return defaultConstructor;
        }

        private void parsePInterfaceTypeList(PInterfaceTypeList interfaceTypeList) throws Exception {
                AInterfaceType interfaceType = null;
                if (interfaceTypeList instanceof AInterfaceTypeInterfaceTypeList) {
                        interfaceType = (AInterfaceType) ((AInterfaceTypeInterfaceTypeList) interfaceTypeList).getInterfaceType();
                } else if (interfaceTypeList instanceof AInterfaceTypeListInterfaceTypeList) {
                        AInterfaceTypeListInterfaceTypeList typeList = (AInterfaceTypeListInterfaceTypeList) interfaceTypeList;
                        parsePInterfaceTypeList(typeList.getInterfaceTypeList());
                        interfaceType = (AInterfaceType) typeList.getInterfaceType();
                } else
                        throw new Exception("Unknown type here:" + interfaceTypeList);
                AClassOrInterfaceType type = (AClassOrInterfaceType) interfaceType.getClassOrInterfaceType();
                addSuperInterface(type.getName().toString(), this);
        }

        private void parsePVariableDeclaratorId(PVariableDeclaratorId variableDeclaratorId, PType type, List modifiers) throws Exception {
                if (variableDeclaratorId instanceof AVariableDeclaratorIdVariableDeclaratorId) {
                        AVariableDeclaratorIdVariableDeclaratorId declaratorId = (AVariableDeclaratorIdVariableDeclaratorId) variableDeclaratorId;
                        parsePVariableDeclaratorId(declaratorId.getVariableDeclaratorId(), type, modifiers);
                } else if (variableDeclaratorId instanceof AIdentifierVariableDeclaratorId) {
                        AIdentifierVariableDeclaratorId declaratorId = (AIdentifierVariableDeclaratorId) variableDeclaratorId;
                        Attribute attribute = new Attribute("");
                        attribute.parse(type, variableDeclaratorId, declaratorId.getIdentifier(), modifiers);
                        addAttribute(attribute);
                } else
                        throw new Exception("Unknown type here: " + variableDeclaratorId);
        }

        private void parsePVariableDeclarators(PVariableDeclarators variableDeclarators, PType type, List modifiers) throws Exception {
                PVariableDeclarator variableDeclarator = null;
                if (variableDeclarators instanceof AVariableDeclaratorVariableDeclarators) {
                        variableDeclarator = ((AVariableDeclaratorVariableDeclarators) variableDeclarators).getVariableDeclarator();
                } else if (variableDeclarators instanceof AVariableDeclaratorsVariableDeclarators) {
                        AVariableDeclaratorsVariableDeclarators variableDeclarators2 = (AVariableDeclaratorsVariableDeclarators) variableDeclarators;
                        parsePVariableDeclarators(variableDeclarators2.getVariableDeclarators(), type, modifiers);
                        variableDeclarator = variableDeclarators2.getVariableDeclarator();
                } else
                        throw new Exception("Unknown type here: " + variableDeclarators);

                PVariableDeclaratorId declaratorId = null;
                if (variableDeclarator instanceof AVariableDeclaratorIdVariableDeclarator) {
                        declaratorId = ((AVariableDeclaratorIdVariableDeclarator) variableDeclarator).getVariableDeclaratorId();
                } else if (variableDeclarator instanceof AAssignVariableDeclarator) {
                        declaratorId = ((AAssignVariableDeclarator) variableDeclarator).getVariableDeclaratorId();
                } else
                        throw new Exception("Unknoen type here: " + variableDeclarator);
                parsePVariableDeclaratorId(declaratorId, type, modifiers);

        }

        /*
         * Returns list of classes if there are internal classes in the parsed class
         *
         * @see bog.lachesis.model.independent.AbstractNamed#parse()
         */
        public void parse(AClassDeclaration classDeclaration, IAnalysisMonitor analysisMonitor) throws Exception {
                log
                                .debug("<html xmlns=\"http://www.w3.org/1999/xhtml\" xml:lang=\"en\" lang=\"en\"> <!DOCTYPE HTML PUBLIC
\"-//W3C//DTD HTML 4.01 Transitional//EN\">"
                                                + "<head><style type=\"text/css\"> <!-- "
                                                + ".styleBlockClass {font-size: 14px; font-weight:bold; background:CCCCFF;} "
                                                + ".styleBlockValue {font-size: 14px; background:CCCCFF;} "
                                                + ".styleBlockContentClass {font-size: 14px; font-weight:bold; background:yellow;} "
                                                + ".styleBlockContentValue {font-size: 14px; background:yellow;} "
                                                + ".styleMethodCallClass {font-size: 14px;  font-weight:bold; background:CCCCFF;}
"
                                                + ".styleMethodCallValue {font-size: 14px;  background:CCCCFF;} "
                                                + ".styleExpressionClass {font-size: 14px;  font-weight:bold; background:#FFBBBB;}
"
                                                + ".styleExpressionValue {font-size: 14px;  background:#FFBBBB;}  "
                                                + ".styleExpressionType {font-size: 14px;  background:#BBFFBB;}  " +
"</style></head><body>");

                List internalClasses = new ArrayList();
                // parse identifier
                setName(classDeclaration.getIdentifier().getText());

                if (getName().equals("AnalysisDescriptor"))
                 System.out.println("debug here");
                // parse modifiers
                setModified(new Modified(classDeclaration.getModifier()));
                firstLineNumber = classDeclaration.getTClass().getLine();
                lastLineNumber = ((AClassBody) classDeclaration.getClassBody()).getRBrace().getLine();

                // parse "extends" clause

                if (classDeclaration.getSuper() != null)
                        addSuperClass(((ASuper) classDeclaration.getSuper()).getClassType().toString(), this);
```

```
                        // parse "implements" clause
                        if (classDeclaration.getInterfaces() != null)
                                parsePInterfaceTypeList(((AInterfaces) classDeclaration.getInterfaces()).getInterfaceTypeList());

                        // parse member variables
                        for (Iterator iter = ((AClassBody) classDeclaration.getClassBody()).getClassBodyDeclaration().iterator(); iter.hasNext();) {
                                PClassBodyDeclaration classBodyDeclaration = (PClassBodyDeclaration) iter.next();
                                if (classBodyDeclaration instanceof AClassMemberDeclarationClassBodyDeclaration) {
                                        AClassMemberDeclarationClassBodyDeclaration declaration = (AClassMemberDeclarationClassBodyDeclaration)
classBodyDeclaration;
                                        PClassMemberDeclaration classMemberDeclaration = declaration.getClassMemberDeclaration();
                                        if (classMemberDeclaration instanceof AFieldDeclarationClassMemberDeclaration) {
                                                AFieldDeclarationClassMemberDeclaration fieldDeclaration =
(AFieldDeclarationClassMemberDeclaration) classMemberDeclaration;
                                                AFieldDeclaration fieldDeclaration2 = (AFieldDeclaration) fieldDeclaration.getFieldDeclaration();
                                                PType type = fieldDeclaration2.getType();
                                                List modifiers = fieldDeclaration2.getModifier();
                                                parsePVariableDeclarators(fieldDeclaration2.getVariableDeclarators(), type, modifiers);
                                        } else if (classMemberDeclaration instanceof AMethodDeclarationClassMemberDeclaration) {
                                                AMethodDeclarationClassMemberDeclaration methodDeclaration =
(AMethodDeclarationClassMemberDeclaration) classMemberDeclaration;
                                                AMethodDeclaration methodDeclaration2 = (AMethodDeclaration)
methodDeclaration.getMethodDeclaration();
                                                Method method = new Method("");
                                                method.parse(methodDeclaration2, analysisMonitor);
                                                addMethod(method);
                                        } else if (classMemberDeclaration instanceof AClassDeclarationClassMemberDeclaration) {
                                                Class internalClass = new Class("", absoluteLocation);
                                                PClassDeclaration classDeclaration2 = ((AClassDeclarationClassMemberDeclaration)
classMemberDeclaration)
                                                                .getClassDeclaration();
                                                internalClass.parse((AClassDeclaration) classDeclaration2, analysisMonitor);
                                                internalClass.setName(this.getName() + INTERNAL_CLASS_DELIMITER + internalClass.getName
());
                                                internalClass.setEnclosingProgramUnit(this);
                                                internalClasses.add(internalClass);
                                        } else
                                                log.error("Unsupported class member declaration: " + classMemberDeclaration.getClass() + " -> "
                                                                + classMemberDeclaration);
                                } else if (classBodyDeclaration instanceof AConstructorDeclarationClassBodyDeclaration) {
                                        AConstructorDeclarationClassBodyDeclaration constructorDeclaration =
(AConstructorDeclarationClassBodyDeclaration) classBodyDeclaration;
                                        Constructor constructor = new Constructor("");
                                        constructor.parse((AConstructorDeclaration) constructorDeclaration.getConstructorDeclaration(), analysisMonitor);
                                        addMethod(constructor);
                                }
                        }
                        log.debug("</body></html>");

                        childProgramUnits = internalClasses;
                }

                /*
                 * (non-Javadoc)
                 *
                 * @see bog.lachesis.metamodel.independent.ILinesOfCode#getFirstLineNumber()
                 */
                public int getFirstLineNumber() {
                        return firstLineNumber;
                }

                /*
                 * (non-Javadoc)
                 *
                 * @see bog.lachesis.metamodel.independent.ILinesOfCode#getLastLineNumber()
                 */
                public int getLastLineNumber() {
                        return lastLineNumber;
                }

                /*
                 * (non-Javadoc)
                 *
                 * @see org.bog.lachesis.metamodel.independent.IResourceLocateable#getAbsoluteLocation()
                 */
                public ILocation getAbsoluteLocation() {
                        return absoluteLocation;
                }

                /*
                 * (non-Javadoc)
                 *
                 * @see org.bog.lachesis.metamodel.independent.IResourceLocateable#setAbsoluteLocation()
                 */
                public void setAbsoluteLocation(ILocation location) {
                        absoluteLocation = location;
                }

                public boolean isAbstract() {
```

```
                return getModified().getModifiers().contains(Modifier.ABSTRACT);
        }
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AConstructorBody;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AConstructorDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AConstructorDeclarator;

public class Constructor extends Method {
        private static Logger log = Logger.getLogger(Constructor.class);

        public Constructor(String name) {
                super(name);
        }

        private void parsePConstructorDeclarator(AConstructorDeclarator constructorDeclarator) throws Exception {
                        setName(constructorDeclarator.getSimpleName().toString());
                        parsePFormalParameterList(constructorDeclarator.getFormalParameterList());
        }

        /*
         * (non-Javadoc)
         *
         * @see bog.lachesis.model.independent.AbstractNamed#parse()
         */
        public void parse(AConstructorDeclaration constructorDeclaration, IAnalysisMonitor analysisMonitor) throws Exception {
                AConstructorDeclarator constructorDeclarator = (AConstructorDeclarator) constructorDeclaration.getConstructorDeclarator();
                List modifiers = null;

                modifiers = constructorDeclaration.getModifier();

                // parse name and arguments
                parsePConstructorDeclarator(constructorDeclarator);

                log.debug("--------------------------------------------------------------------------<br>");
                log.debug(getName() + "<br>");
                log.debug("--------------------------------------------------------------------------<br>");

                if (getName().startsWith("Messages"))
                        System.out.println("debug here");


                analysisMonitor.setSubTaskAddition(" [" + getName() + "]");

                // parse modifiers
                Modified modified = new Modified(modifiers);
                setModified(modified);

                Type type = new Type(getName());
                setReturningType(type);

                Block block = new Block();
                // long start = System.currentTimeMillis();
                AConstructorBody constructorBody = (AConstructorBody) constructorDeclaration.getConstructorBody();

                block.parse(constructorBody,true, 1);
                // long finish = System.currentTimeMillis();
                // System.out.println("Parse of method "+getName()+" took:
                // "+(finish-start)+" ms.");

                setBlock(block);
        }

}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.lang.reflect.InvocationTargetException;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.impl.AbstractExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ABooleanLiteralLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ABooleanPrimitiveType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AByteIntegralType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ACharIntegralType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ACharacterLiteralLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassOrInterfaceType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ADoubleFloatingPointType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AExpressionCastExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFloatFloatingPointType;
```

```
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFloatingPointLiteralLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIntIntegralType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIntegerLiteralLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ALongIntegralType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ANameArrayType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ANamedTypePrimaryNoNewArray;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ANullLiteralLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APrimitiveTypePrimaryNoNewArray;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AQualifiedThisPrimaryNoNewArray;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AShortIntegralType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AStringLiteralLiteral;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVoidPrimaryNoNewArray;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.Node;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PClassInstanceCreationExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PMethodInvocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PPrimaryNoNewArray;

public class Expression extends AbstractExpression {
		private static Logger log = Logger.getLogger(MethodCall.class);

		private boolean typeIsSet = false;

		private int lineNumber;

		public Expression() {
			setType(new Type(null));
		}

		private void setTypeName(String name) throws Exception {
			if (name == null)
				throw new Exception("Cannot set null name");
			if (!typeIsSet) {
				getType().setName(name);
				setTypeSource(TYPE_SOURCE_TYPE_CAST);
				typeIsSet = true;
			}
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see bog.lachesis.model.independent.AbstractExpression#parse()
		 */
		String temp = "";

		private String levelStr(int level) {
			StringBuffer buffer = new StringBuffer();
			for (int i = 0; i < level; i++)
				buffer.append(".");
			return buffer.toString();
		}

		private void parse(Node node, Block enclosingBlock, boolean initialRun, int level) throws Exception {
			String current = node.toString();
			if (!temp.equals(current)) {
				log.debug("<span class=styleExpressionClass >[Expression]" + levelStr(level)
						+ node.getClass().getName().substring(node.getClass().getName().lastIndexOf(".") + 1)
						+ "</span> <span class=styleExpressionValue >" + current + "</span><br>");
				level++;
				temp = current;
			}

			if ((node instanceof PClassInstanceCreationExpression) || (node instanceof PMethodInvocation)) {
				enclosingBlock.getOperators().incItemUsage("METHOD_CALL");
				MethodCall methodCall = new MethodCall("");
				methodCall.setRawName(node.toString());
				methodCall.parse(node, enclosingBlock, level + 1);
				if (!typeIsSet) {
					setMethodCall(methodCall);
					setTypeSource(TYPE_SOURCE_METHOD_CALL);
					typeIsSet = true;
				}
				// stop recursive parsing here
				return;
			// check for 'class' REFERENCE
			} else if (node instanceof PPrimaryNoNewArray) {
				if (node instanceof ANamedTypePrimaryNoNewArray) {
					setTypeName("java.lang.Class");
				} else if (node instanceof AVoidPrimaryNoNewArray) {
					setTypeName("java.lang.Class");
				} else if (node instanceof APrimitiveTypePrimaryNoNewArray) {
					setTypeName("java.lang.Class");
				} else if (node instanceof AQualifiedThisPrimaryNoNewArray) {
					setTypeName(((AQualifiedThisPrimaryNoNewArray)node).getName().toString());
				}
			}
			// check for TYPE CAST
			else if (node instanceof AExpressionCastExpression) {
				setTypeName(((AExpressionCastExpression)node).getExpression().toString());
```

```
                }
                // check for primitive types' LITERALS
                else if ((node instanceof ABooleanLiteralLiteral) || (node instanceof ABooleanPrimitiveType)) {
                        setTypeName("boolean");
                } else if (node instanceof ANullLiteralLiteral) {
                        setTypeName("null");
                } else if ((node instanceof AIntegerLiteralLiteral) || (node instanceof AIntIntegralType)) {
                        setTypeName("int");
                } else if (node instanceof ALongIntegralType) {
                        setTypeName("long");
                } else if (node instanceof AShortIntegralType) {
                        setTypeName("short");
                } else if (node instanceof AByteIntegralType) {
                        setTypeName("byte");
                } else if ((node instanceof AFloatingPointLiteralLiteral) || (node instanceof AFloatFloatingPointType)) {
                        setTypeName("float");
                } else if (node instanceof ADoubleFloatingPointType) {
                        setTypeName("double");
                } else if (node instanceof AStringLiteralLiteral) {
                        setTypeName("String");
                } else if ((node instanceof ACharacterLiteralLiteral) || (node instanceof ACharIntegralType)) {
                        setTypeName("char");
                } else if (node instanceof ANameArrayType) {
                        setTypeName(((ANameArrayType) node).getName().toString());
                } else if (node instanceof AClassOrInterfaceType) {
                        setTypeName(((AClassOrInterfaceType) node).getName().toString());
                }

                List itemsCreated = BlockContent.parseBlockContent(enclosingBlock, node, false, level + 1);

                // if attributeAccesses were parsed during parseBlockContent
                if ((itemsCreated != null) && (!typeIsSet)) {
                        Reference attributeAccess = (Reference) itemsCreated.get(0);
                        setReference(attributeAccess);
                        setTypeSource(TYPE_SOURCE_VARIABLE_ACCESS);
                        typeIsSet = true;
                }

                java.lang.reflect.Method[] methods = node.getClass().getMethods();
                for (int i = 0; i < methods.length; i++) {
                        java.lang.reflect.Method method = methods[i];
                        if (method.getName().startsWith("get")
                                        && (method.getReturnType() != null)
                                        && (method.getReturnType().getPackage() != null)
                                        && (method.getReturnType().getPackage().getName().startsWith("org.bog.lachesis") ||
method.getReturnType().getName()
                                                        .endsWith("List"))) {
                                try {
                                        Object result = method.invoke(node, null);
                                        // System.out.println("Invoked: "+node.getClass()+"."
                                        // +method.getName());
                                        if (result != null)
                                                if (result instanceof Collection)
                                                        for (Iterator iter = ((Collection) result).iterator(); iter.hasNext();)
                                                                parse((Node) iter.next(), enclosingBlock, false, level +
1);
                                                else
                                                        parse((Node) result, enclosingBlock, false, level + 1);
                                } catch (IllegalArgumentException e) {
                                        e.printStackTrace();
                                } catch (IllegalAccessException e) {
                                        e.printStackTrace();
                                } catch (InvocationTargetException e) {
                                        e.printStackTrace();
                                }
                        }

                }

        }

        public void parse(PExpression expression, Block enclosingBlock, int level) throws Exception {
                parse(expression, enclosingBlock, true, level);
                if (!typeIsSet)
                        throw new Exception("Could not resolve the type of the expression!");
                // if (typeIsSet) {
                String type = "";
                switch (getTypeSource()) {
                case TYPE_SOURCE_TYPE_CAST:
                        type = "eCAST: " + getType().getName();
                        break;
                case TYPE_SOURCE_METHOD_CALL:
                        type = "eMETHOD: " + getMethodCall().getName();
                        break;
                case TYPE_SOURCE_VARIABLE_ACCESS:
                        type = "eREFERENCE: " + getReference().getName();
                        break;
                }
                log.debug("<span class=styleExpressionType >[Type Parsed]" + levelStr(level) + type + "</span><br>");
                // }
```

```
		}

		public int getLineNumber() {
			return lineNumber;
		}

		public void setLineNumber(int lineNumber) {
			this.lineNumber = lineNumber;
		}

		public String toString() {
			switch (getTypeSource()) {
			case TYPE_SOURCE_TYPE_CAST:
				return "eCAST: '" + getType().getName()+"'";
			case TYPE_SOURCE_METHOD_CALL:
				return "eMETHOD: '" + getMethodCall().getName()+"'";
			case TYPE_SOURCE_VARIABLE_ACCESS:
				return "eREFERENCE: '" + getReference().getName()+"'";
			}
			return "unknown";
		}
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;


import org.bog.lachesis.metamodel.impl.AbstractImport;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASingleTypeImportDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASingleTypeImportDeclarationImportDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ATypeImportOnDemandDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ATypeImportOnDemandDeclarationImportDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PImportDeclaration;


public class Import extends AbstractImport {
   /**
    * @param name
    */
   public Import(String name) {
      super(name);
   }

   /* (non-Javadoc)
    * @see bog.lachesis.model.independent.AbstractImport#parse()
    */
   public void parse(PImportDeclaration importDeclaration) throws Exception
   {
			if (importDeclaration instanceof ASingleTypeImportDeclarationImportDeclaration) {
				setName(((ASingleTypeImportDeclaration)((ASingleTypeImportDeclarationImportDeclaration)importDeclaration).
getSingleTypeImportDeclaration()).getName().toString());
			} else if (importDeclaration instanceof ATypeImportOnDemandDeclarationImportDeclaration) {
				setName(((ATypeImportOnDemandDeclaration)((ATypeImportOnDemandDeclarationImportDeclaration)importDeclaration).
getTypeImportOnDemandDeclaration()).getName().toString()+".*");
			} else
				throw new Exception("Unknown type here: " + importDeclaration);

   }
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;

import org.bog.lachesis.metamodel.IImport;
import org.bog.lachesis.metamodel.impl.AbstractImports;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PImportDeclaration;


public class Imports extends AbstractImports {

		private static final HashMap excludeImports = new HashMap();

		static {
			excludeImports.put("void", "void");
			excludeImports.put("short", "short");
			excludeImports.put("int", "int");
			excludeImports.put("long", "long");
			excludeImports.put("byte", "byte");
			excludeImports.put("char", "char");
			excludeImports.put("float", "float");
			excludeImports.put("double", "double");
			excludeImports.put("boolean", "boolean");
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see org.bog.lachesis.metamodel.independent.impl.AbstractImports#addImport(org.bog.lachesis.metamodel.independent.IImport)
```

```
		 */
		public IImport addImport(IImport _import) {
			if (!excludeImports.containsKey(_import.getName()))
				return super.addImport(_import);
			else
				return null;
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see bog.lachesis.metamodel.independent.impl.AbstractImports#parse(bog.lachesis.recognizers.sourcecode.AbstractLinguisticProcessor,
		 *      int)
		 */
		public void parse(LinkedList list) throws Exception {
			for (Iterator iter = list.iterator(); iter.hasNext();) {
				PImportDeclaration importDeclaration = (PImportDeclaration) iter.next();
				Import import1 = new Import(null);
				import1.parse(importDeclaration);
				addImport(import1);
			}
		}
}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;


import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.metamodel.ILinesOfCode;
import org.bog.lachesis.metamodel.ILocation;
import org.bog.lachesis.metamodel.IResourceLocateable;
import org.bog.lachesis.metamodel.impl.AbstractInterface;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AAbstractMethodDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AAbstractMethodDeclarationInterfaceMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AAssignVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassDeclarationInterfaceMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AClassOrInterfaceType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AConstantDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AConstantDeclarationInterfaceMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AExtendsExtendsInterfaces;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AExtendsInterfacesExtendsInterfaces;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFieldDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIdentifierVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInterfaceBody;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInterfaceDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInterfaceDeclarationInterfaceMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInterfaceType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorVariableDeclarators;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorsVariableDeclarators;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PClassDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PExtendsInterfaces;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PInterfaceDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PInterfaceMemberDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclarators;

public class Interface extends AbstractInterface implements IResourceLocateable, ILinesOfCode {

		private static Logger log = Logger.getLogger(Class.class);

		private ILocation absoluteLocation;

		private int firstLineNumber = 0;

		private int lastLineNumber = 0;

		/**
		 * @param name
		 */
		public Interface(String name, ILocation absoluteLocation) {
			super(name);
			this.absoluteLocation = absoluteLocation;
		}

		private void parsePExtendsInterfaces(PExtendsInterfaces extendsInterfaces) throws Exception {
			AInterfaceType interfaceType = null;
			if (extendsInterfaces instanceof AExtendsExtendsInterfaces) {
				interfaceType = (AInterfaceType) ((AExtendsExtendsInterfaces) extendsInterfaces).getInterfaceType();
			} else if (extendsInterfaces instanceof AExtendsInterfacesExtendsInterfaces) {
```

```
                                        AExtendsInterfacesExtendsInterfaces interfacesExtendsInterfaces = (AExtendsInterfacesExtendsInterfaces) extendsInterfaces;
                                        parsePExtendsInterfaces(interfacesExtendsInterfaces.getExtendsInterfaces());
                                        interfaceType = (AInterfaceType) interfacesExtendsInterfaces.getInterfaceType();
                                } else
                                        throw new Exception("Unknown type here:" + extendsInterfaces);
                                AClassOrInterfaceType type = (AClassOrInterfaceType) interfaceType.getClassOrInterfaceType();
                                addSuperInterface(type.getName().toString(), this);
                }

                private void parsePVariableDeclaratorId(PVariableDeclaratorId variableDeclaratorId, PType type, List modifiers) throws Exception {
                                if (variableDeclaratorId instanceof AVariableDeclaratorIdVariableDeclaratorId) {
                                        AVariableDeclaratorIdVariableDeclaratorId declaratorId = (AVariableDeclaratorIdVariableDeclaratorId) variableDeclaratorId;
                                        parsePVariableDeclaratorId(declaratorId.getVariableDeclaratorId(), type, modifiers);
                                } else if (variableDeclaratorId instanceof AIdentifierVariableDeclaratorId) {
                                        AIdentifierVariableDeclaratorId declaratorId = (AIdentifierVariableDeclaratorId) variableDeclaratorId;
                                        Attribute attribute = new Attribute("");
                                        attribute.parse(type, variableDeclaratorId, declaratorId.getIdentifier(), modifiers);
                                        addAttribute(attribute);
                                } else
                                        throw new Exception("Unknown type here: " + variableDeclaratorId);
                }

                private void parsePVariableDeclarators(PVariableDeclarators variableDeclarators, PType type, List modifiers) throws Exception {
                                PVariableDeclarator variableDeclarator = null;
                                if (variableDeclarators instanceof AVariableDeclaratorVariableDeclarators) {
                                        variableDeclarator = ((AVariableDeclaratorVariableDeclarators) variableDeclarators).getVariableDeclarator();
                                } else if (variableDeclarators instanceof AVariableDeclaratorsVariableDeclarators) {
                                        AVariableDeclaratorsVariableDeclarators variableDeclarators2 = (AVariableDeclaratorsVariableDeclarators) variableDeclarators;
                                        parsePVariableDeclarators(variableDeclarators2.getVariableDeclarators(), type, modifiers);
                                        variableDeclarator = variableDeclarators2.getVariableDeclarator();
                                } else
                                        throw new Exception("Unknown type here: " + variableDeclarators);

                                PVariableDeclaratorId declaratorId = null;
                                if (variableDeclarator instanceof AVariableDeclaratorIdVariableDeclarator) {
                                        declaratorId = ((AVariableDeclaratorIdVariableDeclarator) variableDeclarator).getVariableDeclaratorId();
                                } else if (variableDeclarator instanceof AAssignVariableDeclarator) {
                                        declaratorId = ((AAssignVariableDeclarator) variableDeclarator).getVariableDeclaratorId();
                                } else
                                        throw new Exception("Unknoen type here: " + variableDeclarator);
                                parsePVariableDeclaratorId(declaratorId, type, modifiers);

                }

                /*
                 * (non-Javadoc)
                 *
                 * @see bog.lachesis.model.independent.AbstractNamed#parse()
                 */
                public void parse(AInterfaceDeclaration interfaceDeclaration, IAnalysisMonitor analysisMonitor) throws Exception {
                                List internalProgramUnits = new ArrayList();

                                setName(interfaceDeclaration.getIdentifier().getText());
                                // parse modifiers
                                // parse modifiers
                                setModified(new Modified(interfaceDeclaration.getModifier()));

                                // parse "extends" clause
                                if (interfaceDeclaration.getExtendsInterfaces() != null)
                                        parsePExtendsInterfaces(interfaceDeclaration.getExtendsInterfaces());

                                // parse members
                                for (Iterator iter = ((AInterfaceBody) interfaceDeclaration.getInterfaceBody()).getInterfaceMemberDeclaration().iterator(); iter
                                                .hasNext();) {
                                        PInterfaceMemberDeclaration memberDeclaration = (PInterfaceMemberDeclaration) iter.next();
                                        if (memberDeclaration instanceof AConstantDeclarationInterfaceMemberDeclaration) {
                                                AConstantDeclarationInterfaceMemberDeclaration declaration = (AConstantDeclarationInterfaceMemberDeclaration)
memberDeclaration;
                                                AConstantDeclaration constantDeclaration = (AConstantDeclaration) declaration.getConstantDeclaration();
                                                AFieldDeclaration fieldDeclaration = (AFieldDeclaration) constantDeclaration.getFieldDeclaration();
                                                PType type = fieldDeclaration.getType();
                                                List modifiers = fieldDeclaration.getModifier();
                                                parsePVariableDeclarators(fieldDeclaration.getVariableDeclarators(), type, modifiers);
                                        } else if (memberDeclaration instanceof AAbstractMethodDeclarationInterfaceMemberDeclaration) {
                                                AAbstractMethodDeclarationInterfaceMemberDeclaration methodDeclaration =
(AAbstractMethodDeclarationInterfaceMemberDeclaration) memberDeclaration;
                                                AAbstractMethodDeclaration abstractMethodDeclaration = (AAbstractMethodDeclaration) methodDeclaration
                                                                .getAbstractMethodDeclaration();
                                                Method method = new Method("");
                                                method.parse(abstractMethodDeclaration, analysisMonitor);
                                                addMethod(method);
                                        } else if (memberDeclaration instanceof AClassDeclarationInterfaceMemberDeclaration) {
                                                Class internalClass = new Class("", absoluteLocation);
                                                PClassDeclaration classDeclaration2 = ((AClassDeclarationInterfaceMemberDeclaration) memberDeclaration)
                                                                .getClassDeclaration();
                                                internalClass.parse((AClassDeclaration) classDeclaration2, analysisMonitor);
                                                internalClass.setName(this.getName() + Class.INTERNAL_CLASS_DELIMITER + internalClass.getName());
                                                internalClass.setEnclosingProgramUnit(this);
                                                internalProgramUnits.add(internalClass);
                                        } else if (memberDeclaration instanceof AInterfaceDeclarationInterfaceMemberDeclaration) {
```

```
                                                Interface internalInterface = new Interface("", absoluteLocation);
                                                PInterfaceDeclaration interfaceDeclaration2 = ((AInterfaceDeclarationInterfaceMemberDeclaration)
memberDeclaration)
                                                                .getInterfaceDeclaration();
                                                internalInterface.parse((AInterfaceDeclaration) interfaceDeclaration2, analysisMonitor);
                                                internalInterface.setName(this.getName() + Class.INTERNAL_CLASS_DELIMITER + internalInterface.getName());
                                                internalInterface.setEnclosingProgramUnit(this);
                                                internalProgramUnits.add(internalInterface);
                                        } else
                                                log.error("Unsupported class member declaration: " + memberDeclaration.getClass() + " -> " + memberDeclaration);

                        }
                        childProgramUnits = internalProgramUnits;
        }

        /*
         * (non-Javadoc)
         *
         * @see bog.lachesis.metamodel.independent.ILinesOfCode#getFirstLineNumber()
         */
        public int getFirstLineNumber() {
                return firstLineNumber;
        }

        /*
         * (non-Javadoc)
         *
         * @see bog.lachesis.metamodel.independent.ILinesOfCode#getLastLineNumber()
         */
        public int getLastLineNumber() {
                return lastLineNumber;
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.metamodel.independent.IResourceLocateable#getAbsoluteLocation()
         */
        public ILocation getAbsoluteLocation() {
                return absoluteLocation;
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.metamodel.independent.IResourceLocateable#setAbsoluteLocation()
         */
        public void setAbsoluteLocation(ILocation location) {
                absoluteLocation = location;
        }

        public boolean isAbstract() {
                return true;
        }
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import org.bog.lachesis.metamodel.ILocation;

public class JavaFileLocation extends Named implements ILocation {

        public JavaFileLocation(String name) {
                super(name);
        }

}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.io.File;

import org.bog.lachesis.metamodel.ILocation;

public class JavaProjectLocation extends Named implements ILocation {

        private File file;
        public JavaProjectLocation(String name) {
                super(name);
        }
        public JavaProjectLocation(File file) {
                super(file.getAbsolutePath());
                this.file = file;
        }
        public File getFile() {
                return file;
        }
        public void setFile(File file) {
                this.file = file;
```

```
		}

}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;


import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.metamodel.IParameter;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.impl.AbstractMethod;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AAbstractMethodDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ABlock;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ABlockMethodBody;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFormalParameter;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFormalParameterFormalParameterList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFormalParameterListFormalParameterList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIdentifierMethodDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AMethodDeclaration;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AMethodDeclaratorMethodDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ATypeMethodHeader;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVoidMethodHeader;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PFormalParameter;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PFormalParameterList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PMethodBody;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PMethodDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PMethodHeader;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PType;

public class Method extends AbstractMethod {
		private static Logger log = Logger.getLogger(Method.class);

		int firstLineNumber = 0;

		int lastLineNumber = 0;

		/**
		 * @param name
		 */
		public Method(String name) {
			super(name);

		}

		public void parsePFormalParameterList(PFormalParameterList formalParameterList) throws Exception {
			if (formalParameterList == null)
				return;
			PFormalParameter formalParameter = null;
			if (formalParameterList instanceof AFormalParameterFormalParameterList) {
				AFormalParameterFormalParameterList formalParameterList2 = (AFormalParameterFormalParameterList) formalParameterList;
				formalParameter = formalParameterList2.getFormalParameter();
			} else if (formalParameterList instanceof AFormalParameterListFormalParameterList) {
				AFormalParameterListFormalParameterList formalParameterList2 = (AFormalParameterListFormalParameterList)
formalParameterList;
				parsePFormalParameterList(formalParameterList2.getFormalParameterList());
				formalParameter = formalParameterList2.getFormalParameter();
			} else
				throw new Exception("Unknown type here: " + formalParameterList);
			Parameter parameter = new Parameter("");
			parameter.parse((AFormalParameter) formalParameter);
			addParameter(parameter);
		}

		private void parsePMethodDeclarator(PMethodDeclarator methodDeclarator) throws Exception {
			if (methodDeclarator instanceof AIdentifierMethodDeclarator) {
				AIdentifierMethodDeclarator methodDeclarator2 = (AIdentifierMethodDeclarator) methodDeclarator;
				setName(methodDeclarator2.getIdentifier().getText());
				firstLineNumber = methodDeclarator2.getIdentifier().getLine();
				parsePFormalParameterList(methodDeclarator2.getFormalParameterList());
			} else if (methodDeclarator instanceof AMethodDeclaratorMethodDeclarator) {
				parsePMethodDeclarator(((AMethodDeclaratorMethodDeclarator) methodDeclarator).getMethodDeclarator());
			} else
				throw new Exception("Unknown type here: " + methodDeclarator);
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see bog.lachesis.model.independent.AbstractNamed#parse()
		 */
		public void parse(AMethodDeclaration methodDeclaration, IAnalysisMonitor analysisMonitor) throws Exception {
			PMethodHeader methodHeader = methodDeclaration.getMethodHeader();
			PMethodDeclarator methodDeclarator = null;
			List modifiers = null;
			PType ptype = null;
			if (methodHeader instanceof ATypeMethodHeader) {
				ATypeMethodHeader header = (ATypeMethodHeader) methodHeader;
```

```
                        modifiers = header.getModifier();
                        methodDeclarator = header.getMethodDeclarator();
                        ptype = header.getType();
                } else if (methodHeader instanceof AVoidMethodHeader) {
                        AVoidMethodHeader header = (AVoidMethodHeader) methodHeader;
                        modifiers = header.getModifier();
                        methodDeclarator = header.getMethodDeclarator();
                } else
                        throw new Exception("Unknown type here: " + methodHeader);
                // parse identifier
                parsePMethodDeclarator(methodDeclarator);

                log.debug("--------------------------------------------------------------------------<br>");
                log.debug(getName() + "<br>");
                log.debug("--------------------------------------------------------------------------<br>");

                if (getName().equals("deleteMeLater"))
                        System.out.println("debug here");
                analysisMonitor.setSubTaskAddition(" [" + getName() + "]");

                // parse modifiers
                Modified modified = new Modified(modifiers);
                setModified(modified);

                Type type = new Type(null);
                type.parse(ptype);
                setReturningType(type);

                Block block = new Block();

                PMethodBody methodBody = methodDeclaration.getMethodBody();
                if (methodBody instanceof ABlockMethodBody) {
                        block.parse(((ABlockMethodBody) methodBody).getBlock());
                        lastLineNumber = ((ABlock) ((ABlockMethodBody) methodBody).getBlock()).getRBrace().getLine();
                }
                setBlock(block);
        }

        public void parse(AAbstractMethodDeclaration methodDeclaration, IAnalysisMonitor analysisMonitor) throws Exception {
                PMethodHeader methodHeader = methodDeclaration.getMethodHeader();
                PMethodDeclarator methodDeclarator = null;
                List modifiers = null;
                PType ptype = null;
                if (methodHeader instanceof ATypeMethodHeader) {
                        ATypeMethodHeader header = (ATypeMethodHeader) methodHeader;
                        modifiers = header.getModifier();
                        methodDeclarator = header.getMethodDeclarator();
                        ptype = header.getType();
                } else if (methodHeader instanceof AVoidMethodHeader) {
                        AVoidMethodHeader header = (AVoidMethodHeader) methodHeader;
                        modifiers = header.getModifier();
                        methodDeclarator = header.getMethodDeclarator();
                } else
                        throw new Exception("Unknown type here: " + methodHeader);
                // parse identifier
                parsePMethodDeclarator(methodDeclarator);

                analysisMonitor.setSubTaskAddition(" [" + getName() + "]");

                // parse modifiers
                Modified modified = new Modified(modifiers);
                setModified(modified);

                Type type = new Type(null);
                type.parse(ptype);
                setReturningType(type);
        }

        /*
         * (non-Javadoc)
         *
         * @see bog.lachesis.metamodel.independent.ISourceLocateable#getLineNumber()
         */
        public int getFirstLineNumber() {
                return firstLineNumber;
        }

        public int getLastLineNumber() {
                return lastLineNumber;
        }

        private final static int LEVEL1 = 100000;

        private final static int LEVEL2 = 200000;

        private final static int LEVEL3 = 300000;

        private final static int LEVEL4 = 400000;

        private final static int LEVEL5 = 500000;
```

```
private int addOverLevel(int value, int level) {
		if (value > level)
			return value + 1;
		return level + 1;
}

/**
 * @return the compatibility distance. Returns 0 if the method is the same,
 *         greater than 1 when some of the arguments are compatible but not
 *         equal, and returns -1 if there is no compatibility.
 */
public int compliesWith(String methodName, List parameterTypes) {
		int distance = 0;

		if (!getName().equals(methodName))
			return -1;
		if (getParameters().size() != parameterTypes.size())
			return -1;
		for (int i = 0; i < parameterTypes.size(); i++) {
			IProgramUnit argumentType = null;
			/**
			 * TODO may remove try-catch ...
			 */
			try {
				argumentType = (IProgramUnit) parameterTypes.get(i);
			} catch (Exception e) {
				String message = "Could not test for method equality since the argument type for argument " + (i + 1)
						+ " was not processed. Method tested: " + methodName;
				log.error(message);
				// System.out.println(message);
				e.printStackTrace();
			}
			IParameter formalParameter = (IParameter) getParameters().get(i);
			IProgramUnit formalParameterType = formalParameter.getType().getProgramUnit();

			if (argumentType == null) {
				log.error("Could not test for method equality since the argument type for argument " + (i + 1) + " was not resolved.");
				return -1;
			}

			// ------------------------------------------------------------------
			// test for null parameter (it's a compatibility check but is more
			// faster and thus placed on first place)
			// ------------------------------------------------------------------

			// a null argument is passed and the corresponding formal parameter
			// declaration is not of primitive type (allows null)
			if ((argumentType == Class.CLASS_WRAPPER_NULL) && (!((Type) formalParameter.getType()).isPrimitive())) {
				distance++;
				continue;
			}

			// ------------------------------------------------------------------
			// test for type equality
			// ------------------------------------------------------------------
			if (argumentType.equals(formalParameterType))
				continue;

			// ------------------------------------------------------------------
			// test for compatibility
			// ------------------------------------------------------------------

			// ------------------------------------------------------------------
			// Java Language Specification
			// 5.1.2 Widening Primitive Conversion
			// http://java.sun.com/docs/books/jls/second_edition/html/conversions.doc.html#25214
			// ------------------------------------------------------------------

			// byte to short, int, long, float, or double
			if (argumentType.equals(Class.CLASS_BYTE)) {
				if (formalParameterType.equals(Class.CLASS_SHORT)) {
					distance = addOverLevel(distance, LEVEL1);
					continue;
				}
				if (formalParameterType.equals(Class.CLASS_INT)) {
					distance = addOverLevel(distance, LEVEL2);
					continue;
				}
				if (formalParameterType.equals(Class.CLASS_LONG)) {
					distance = addOverLevel(distance, LEVEL3);
					continue;
				}
				if (formalParameterType.equals(Class.CLASS_FLOAT)) {
					distance = addOverLevel(distance, LEVEL4);
					continue;
				}
				if (formalParameterType.equals(Class.CLASS_DOUBLE)) {
					distance = addOverLevel(distance, LEVEL5);
					continue;
```

```
                        }
                    }
                    // short to int, long, float, or double
                    if (argumentType.equals(Class.CLASS_SHORT)) {
                        if (formalParameterType.equals(Class.CLASS_INT)) {
                            distance = addOverLevel(distance, LEVEL1);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_LONG)) {
                            distance = addOverLevel(distance, LEVEL2);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_FLOAT)) {
                            distance = addOverLevel(distance, LEVEL3);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_DOUBLE)) {
                            distance = addOverLevel(distance, LEVEL4);
                            continue;
                        }
                    }
                    // char to int, long, float, or double
                    if (argumentType.equals(Class.CLASS_CHARACTER)) {
                        if (formalParameterType.equals(Class.CLASS_INT)) {
                            distance = addOverLevel(distance, LEVEL1);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_LONG)) {
                            distance = addOverLevel(distance, LEVEL2);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_FLOAT)) {
                            distance = addOverLevel(distance, LEVEL3);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_DOUBLE)) {
                            distance = addOverLevel(distance, LEVEL4);
                            continue;
                        }
                    }
                    // int to long, float, or double
                    if (argumentType.equals(Class.CLASS_INT)) {
                        if (formalParameterType.equals(Class.CLASS_LONG)) {
                            distance = addOverLevel(distance, LEVEL1);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_FLOAT)) {
                            distance = addOverLevel(distance, LEVEL2);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_DOUBLE)) {
                            distance = addOverLevel(distance, LEVEL3);
                            continue;
                        }
                    }
                    // long to float or double
                    if (argumentType.equals(Class.CLASS_LONG)) {
                        if (formalParameterType.equals(Class.CLASS_FLOAT)) {
                            distance = addOverLevel(distance, LEVEL1);
                            continue;
                        }
                        if (formalParameterType.equals(Class.CLASS_DOUBLE)) {
                            distance = addOverLevel(distance, LEVEL2);
                            continue;
                        }
                    }
                    // float to double
                    if (argumentType.equals(Class.CLASS_FLOAT)) {
                        if (formalParameterType.equals(Class.CLASS_DOUBLE)) {
                            distance = addOverLevel(distance, LEVEL1);
                            continue;
                        }
                    }

                    /**
                     * TODO should check whether JLS is followed by this method
                     * 'isSuper'
                     *
                     * 5.1.4 Widening Reference Conversions The following conversions
                     */
                    if (argumentType.isSuper(formalParameterType)) {
                        distance = addOverLevel(distance, LEVEL1);
                        continue;
                    }

                }
                // TODO: add parameter types comparation here when supported
                return distance;
            }
        }
```

```
package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.IBlock;
import org.bog.lachesis.metamodel.INamed;
import org.bog.lachesis.metamodel.IReference;
import org.bog.lachesis.metamodel.ISourceCodeLineNumberLocateable;
import org.bog.lachesis.metamodel.impl.AbstractMethodCall;
import org.bog.lachesis.metamodel.impl.AbstractNamed;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AArgumentListArgumentList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AExpressionArgumentList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AExpressionCastExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFieldAccessPrimaryNoNewArray;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AInnerclassClassInstanceCreationExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ANameMethodInvocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APrimaryFieldAccess;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APrimaryMethodInvocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AQualifiedClassInstanceCreationExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AQualifiedName;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AQualifiedNameName;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASimpleClassInstanceCreationExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASimpleName;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASimpleNameName;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASuperFieldAccess;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.ASuperMethodInvocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AThisPrimaryNoNewArray;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PArgumentList;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PClassInstanceCreationExpression;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PFieldAccess;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PMethodInvocation;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PName;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PPrimary;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.TIdentifier;

public class MethodCall extends AbstractMethodCall {
	private static Logger log = Logger.getLogger(MethodCall.class);

	private String parsedMethodName;

	private List identifiers = new ArrayList();

	public MethodCall(String name) {
		super(name);
	}

	private void parsePName(PName name, IBlock hostBlock) throws Exception {
		String memberName = null;
		INamed memberAccess = null;
		if (name instanceof ASimpleNameName) {
			TIdentifier identifier = ((ASimpleName) ((ASimpleNameName) name).getSimpleName()).getIdentifier();
			memberName = identifier.getText();
			if (identifiers.size() == 0)
				memberAccess = new MethodAccess(memberName, identifier.getLine());
			else {
				Reference reference = new Reference(memberName);
				reference.parse(memberName);
				memberAccess = new ReferenceAccess(reference, identifier.getLine());
				hostBlock.addAttributeAccess(reference);
			}
			identifiers.add(memberAccess);
		} else if (name instanceof AQualifiedNameName) {
			TIdentifier identifier = ((AQualifiedName) ((AQualifiedNameName) name).getQualifiedName()).getIdentifier();
			memberName = identifier.getText();
			if (identifiers.size() == 0)
				memberAccess = new MethodAccess(memberName, identifier.getLine());
			else {
				Reference reference = new Reference("");
				reference.parse(memberName);
				memberAccess = new ReferenceAccess(reference, identifier.getLine());
			}
			identifiers.add(memberAccess);
			parsePName(((AQualifiedName) ((AQualifiedNameName) name).getQualifiedName()).getName(), hostBlock);
		} else
			throw new Exception("Unknown type here: " + name);
		/*
		 * name = {simple_name} simple_name |
		 *
		 * {qualified_name} qualified_name;
		 *
		 * simple_name = identifier;
		 *
		 * qualified_name = name dot identifier;
		 *
```

```
		 */
	}

	private void parsePPrimary(PPrimary primary) {

	}

	String temp = "";

	private String levelStr(int level) {
			StringBuffer buffer = new StringBuffer();
			for (int i = 0; i < level; i++)
				buffer.append(".");
			return buffer.toString();
	}

	private void parsePArgumentList(MethodAccess methodAccess, Block enclosingBlock, PArgumentList argumentList, int level)
				throws Exception {
			if (argumentList == null)
				return;
			if (argumentList instanceof AExpressionArgumentList) {
				AExpressionArgumentList expressionArgumentList = (AExpressionArgumentList) argumentList;
				Expression expression = new Expression();
				expression.parse(expressionArgumentList.getExpression(), enclosingBlock, level);
				methodAccess.getArguments().add(expression);
			} else if (argumentList instanceof AArgumentListArgumentList) {
				AArgumentListArgumentList argumentListArgumentList = (AArgumentListArgumentList) argumentList;
				parsePArgumentList(methodAccess, enclosingBlock, argumentListArgumentList.getArgumentList(), level + 1);
				Expression expression = new Expression();
				expression.parse(argumentListArgumentList.getExpression(), enclosingBlock, level);
				expression.setLineNumber(argumentListArgumentList.getComma().getLine());
				methodAccess.getArguments().add(expression);
			} else
				throw new Exception("Unknown type here: " + argumentList);
	}

	private void parseAPrimaryMethodInvocation(Object node, boolean initialRun, Block enclosingBlock, int level) throws Exception {
			// System.out.println("APrimaryMethodInvocation: " + node.getClass()+ "
			// = "+node);

			/*
			 * primary_no_new_array = {literal} literal |
			 *
			 * {this} this |
			 *
			 * {l_parenthese} l_parenthese expression r_parenthese |
			 *
			 * {class_instance_creation_expression}
			 * class_instance_creation_expression |
			 *
			 * {field_access} field_access |
			 *
			 * {method_invocation} method_invocation |
			 *
			 * {array_access} array_access |
			 *
			 * {qualified_this} name dot this |
			 *
			 * {primitive_type} primitive_type dims? dot [t_class]:class |
			 *
			 * {named_type} name dims? dot [t_class]:class |
			 *
			 * {void} void dot [t_class]:class;
			 */

			if (((node instanceof PClassInstanceCreationExpression) || (node instanceof PMethodInvocation)) && (!initialRun)) {
				MethodCall methodCall = new MethodCall("");
				methodCall.setRawName(node.toString());
				methodCall.parse(node, enclosingBlock, level);
				identifiers.addAll(methodCall.getIdentifiers());
				return;
			}

			if (node instanceof AExpressionCastExpression) {
				AExpressionCastExpression castExpression = (AExpressionCastExpression) node;
				MethodCall methodCall = new MethodCall("");
				methodCall.setRawName(node.toString());
				methodCall.parseAPrimaryMethodInvocation(castExpression.getUnaryExpressionNotPlusMinus(), false, enclosingBlock, level);
				identifiers.add(new TypeCast(castExpression.getExpression().toString(), castExpression.getLParenthese().getLine()));
				identifiers.addAll(methodCall.getIdentifiers());
				return;
			}

			if (node instanceof AFieldAccessPrimaryNoNewArray) {
				PFieldAccess fieldAccess = ((AFieldAccessPrimaryNoNewArray) node).getFieldAccess();
				int line = 0;
				if (fieldAccess instanceof ASuperFieldAccess)
					line = ((ASuperFieldAccess) fieldAccess).getIdentifier().getLine();
				else if (fieldAccess instanceof APrimaryFieldAccess)
					line = ((APrimaryFieldAccess) fieldAccess).getIdentifier().getLine();
```

```
                                        else
                                                throw new Exception("Unknown type here" + node);

                                        Reference reference = new Reference("");
                                        reference.parse(node);
                                        ReferenceAccess referenceAccess = new ReferenceAccess(reference, line);
                                        identifiers.add(referenceAccess);

                                        /*
                                         * if (fieldAccess instanceof ASuperFieldAccess) { ASuperFieldAccess
                                         * superFieldAccess = (ASuperFieldAccess) fieldAccess;
                                         * ReferenceAccess referenceAccess = new
                                         * ReferenceAccess(superFieldAccess.getIdentifier().getText(),
                                         * superFieldAccess.getIdentifier().getLine());
                                         * identifiers.add(referenceAccess); ReferenceAccess
                                         * superReferenceAccess = new ReferenceAccess("super",
                                         * superFieldAccess.getIdentifier().getLine());
                                         * identifiers.add(superReferenceAccess); } else if (fieldAccess
                                         * instanceof APrimaryFieldAccess) { APrimaryFieldAccess
                                         * primaryFieldAccess = (APrimaryFieldAccess) fieldAccess; Reference
                                         * reference = new Reference("");
                                         * reference.parse(primaryFieldAccess.getIdentifier().getText());
                                         * ReferenceAccess referenceAccess = new ReferenceAccess(reference,
                                         * primaryFieldAccess.getIdentifier().getLine());
                                         * identifiers.add(referenceAccess);
                                         * parseAPrimaryMethodInvocation(primaryFieldAccess.getPrimary(),
                                         * false, enclosingBlock, level+1); } else throw new Exception
                                         * ("Unknown type here"+node);
                                         */
                        }

                        if (node instanceof AThisPrimaryNoNewArray) {
                                Reference reference = new Reference("");
                                reference.parse(node);
                                ReferenceAccess thisReferenceAccess = new ReferenceAccess(reference, ((AThisPrimaryNoNewArray) node).getThis().getLine());
                                identifiers.add(thisReferenceAccess);
                        }
                        if (node instanceof PArgumentList)
                                return;
                        java.lang.reflect.Method[] methods = node.getClass().getMethods();
                        for (int i = 0; i < methods.length; i++) {
                                java.lang.reflect.Method method = methods[i];
                                if (method.getName().startsWith("get")
                                                && (method.getReturnType() != null)
                                                && (method.getReturnType().getPackage() != null)
                                                && (method.getReturnType().getPackage().getName().startsWith("org.bog.lachesis") ||
method.getReturnType().getName()
                                                                                .endsWith("List"))) {
                                        try {
                                                Object result = method.invoke(node, null);
                                                // System.out.println("Invoked: "+node.getClass()+"."
                                                // +method.getName());
                                                if (result != null)
                                                        if (result instanceof Collection)
                                                                for (Iterator iter = ((Collection) result).iterator(); iter.hasNext();)
                                                                        parseAPrimaryMethodInvocation(iter.next(), false,
enclosingBlock, level);
                                                        else
                                                                parseAPrimaryMethodInvocation(result, false, enclosingBlock, level);
                                        } catch (IllegalArgumentException e) {
                                                e.printStackTrace();
                                        } catch (IllegalAccessException e) {
                                                e.printStackTrace();
                                        } catch (InvocationTargetException e) {
                                                e.printStackTrace();
                                        }
                                }

                        }

                } /*
                         * method_invocation =
                         *
                         * {name} name l_parenthese argument_list? r_parenthese |
                         *
                         * {primary} primary dot identifier l_parenthese argument_list?
                         * r_parenthese |
                         *
                         * {super} T.super dot identifier l_parenthese argument_list?
                         * r_parenthese;
                         *
                         */

                public void parse(Object node, Block enclosingBlock, int level) throws Exception {
                        String current = node.toString();
                        if (!temp.equals(current)) {
                                log.debug("<span class=styleMethodCallClass >[MethodCall]" + levelStr(level)
                                                + node.getClass().getName().substring(node.getClass().getName().lastIndexOf(".") + 1)
                                                + "</span> <span class=styleMethodCallValue >" + current + "</span><br>");
                                level++;
```

```
                        temp = current;
                }

                if (node instanceof ANameMethodInvocation) {
                        ANameMethodInvocation nameMethodInvocation = (ANameMethodInvocation) node;
                        parsePName(nameMethodInvocation.getName(), enclosingBlock);
                        MethodAccess methodAccess = (MethodAccess) identifiers.get(0);
                        parsedMethodName = methodAccess.getName();
                        parsePArgumentList(methodAccess, enclosingBlock, nameMethodInvocation.getArgumentList(), level + 1);
                } else if (node instanceof ASuperMethodInvocation) {
                        ASuperMethodInvocation superMethodInvocation = (ASuperMethodInvocation) node;
                        parsedMethodName = superMethodInvocation.getIdentifier().getText().replaceAll(" ", "");
                        MethodAccess methodAccess = new MethodAccess(parsedMethodName, superMethodInvocation.getIdentifier().getLine());
                        identifiers.add(methodAccess);
                        parsePArgumentList(methodAccess, enclosingBlock, superMethodInvocation.getArgumentList(), level + 1);
                } else if (node instanceof APrimaryMethodInvocation) {
                        APrimaryMethodInvocation primaryMethodInvocation = (APrimaryMethodInvocation) node;
                        parsedMethodName = primaryMethodInvocation.getIdentifier().getText().replaceAll(" ", "");
                        MethodAccess methodAccess = new MethodAccess(parsedMethodName, primaryMethodInvocation.getIdentifier().getLine());
                        identifiers.add(methodAccess);
                        parseAPrimaryMethodInvocation(node, true, enclosingBlock, level);
                        parsePArgumentList(methodAccess, enclosingBlock, primaryMethodInvocation.getArgumentList(), level + 1);
                } else if (node instanceof ASimpleClassInstanceCreationExpression) {
                        ASimpleClassInstanceCreationExpression creationExpression = (ASimpleClassInstanceCreationExpression) node;
                        parsedMethodName = creationExpression.getName().toString().replaceAll(" ", "");
                        ConstructorAccess methodAccess = new ConstructorAccess(parsedMethodName, creationExpression.getLParenthese().getLine());
                        identifiers.add(methodAccess);
                        parsePArgumentList(methodAccess, enclosingBlock, creationExpression.getArgumentList(), level + 1);
                } else if (node instanceof AQualifiedClassInstanceCreationExpression) {
                        AQualifiedClassInstanceCreationExpression creationExpression = (AQualifiedClassInstanceCreationExpression) node;
                        parsedMethodName = creationExpression.getIdentifier().getText().replaceAll(" ", "");
                        ConstructorAccess methodAccess = new ConstructorAccess(parsedMethodName, creationExpression.getIdentifier().getLine());
                        identifiers.add(methodAccess);
                        parsePArgumentList(methodAccess, enclosingBlock, creationExpression.getArgumentList(), level + 1);
                } else if (node instanceof AInnerclassClassInstanceCreationExpression) {
                        AInnerclassClassInstanceCreationExpression creationExpression = (AInnerclassClassInstanceCreationExpression) node;
                        parsedMethodName = creationExpression.getIdentifier().getText().replaceAll(" ", "");
                        ConstructorAccess methodAccess = new ConstructorAccess(parsedMethodName, creationExpression.getIdentifier().getLine());
                        identifiers.add(methodAccess);
                        parsePArgumentList(methodAccess, enclosingBlock, creationExpression.getArgumentList(), level + 1);
                } else
                        throw new Exception("Unknown type here: " + node);

                parsedMethodName += node.getClass();
                enclosingBlock.addMethodCall(this);

                // setName(getName() + "_RESOLVE_NOT_SUPPORTED_FOR_NOW");
}

public String getParsedMethodName() {
        return parsedMethodName;
}

public List getIdentifiers() {
        return identifiers;
}

public String toString() {
        Iterator iterator = identifiers.iterator();
        if (iterator.hasNext())
                return identifiers.toString() + "@" + ((SourceLocated) iterator.next()).getLineNumber();
        else
                return identifiers.toString();
}

public class SourceLocated extends AbstractNamed implements ISourceCodeLineNumberLocateable {
        int lineNumber;

        public SourceLocated(String name, int lineNumber) {
                super(name);
                this.lineNumber = lineNumber;
        }

        public int getLineNumber() {
                return lineNumber;
        }
}

public class ReferenceAccess extends SourceLocated {
        IReference reference;

        public ReferenceAccess(IReference reference, int lineNumber) {
                super(reference.toString(), lineNumber);
                this.reference = reference;
        }

        public String toString() {
                return "REFERENCE: <" + reference.toString() + ">";
        }
```

```
                        public IReference getReference() {
                                return reference;
                        }

                        public String getName() {
                                return reference.toString();
                        }

        }

        public class MethodAccess extends SourceLocated {
                        protected List arguments = new ArrayList();

                        private boolean argumentsProcessed = false;

                        public MethodAccess(String name, int lineNumber) {
                                super(name, lineNumber);
                        }

                        public String toString() {
                                return "METHODCALL: <" + getName() + " (" + arguments + ")>";
                        }

                        public List getArguments() {
                                return arguments;
                        }

                        public boolean isArgumentsProcessed() {
                                return argumentsProcessed;
                        }

                        public void setArgumentsProcessed(boolean argumentsProcessed) {
                                this.argumentsProcessed = argumentsProcessed;
                        }
        }

        public class ConstructorAccess extends MethodAccess {
                        public ConstructorAccess(String name, int lineNumber) {
                                super(name, lineNumber);
                        }

                        public String toString() {
                                return "CONSTRUCTOR: <" + getName() + " (" + arguments + ")>";
                        }
        }

        public class TypeCast extends SourceLocated {
                        public TypeCast(String name, int lineNumber) {
                                super(name, lineNumber);
                        }

                        public String toString() {
                                return "CAST: <" + getName() + ">";
                        }
        }

}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import org.bog.lachesis.metamodel.impl.AbstractModel;

public class Model extends AbstractModel {

    /**
     * @param name
     */
    public Model(String name) {
        super(name);
    }

}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.metamodel.ISourceCodeLineNumberLocateable;
import org.bog.lachesis.metamodel.impl.AbstractModified;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PModifier;


public class Modified extends AbstractModified implements ISourceCodeLineNumberLocateable{
    private int lineNumber = 0;
    public Modified() {
    }

    public Modified(List list) {
```

```
                parse(list);
        }
        /* (non-Javadoc)
         * @see bog.lachesis.model.independent.AbstractNamed#parse()
         */
        public void parse(List list)
        {
                for (Iterator iter = list.iterator(); iter.hasNext();) {
                                                PModifier element = (PModifier) iter.next();
                addModifier(new Modifier(element.toString()));
                                }
        }
        /* (non-Javadoc)
         * @see bog.lachesis.metamodel.independent.ISourceLocateable#getLineNumber()
         */
        public int getLineNumber() {
           return lineNumber;
        }
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import org.bog.lachesis.metamodel.impl.AbstractModifier;

public class Modifier extends AbstractModifier {
                public final static Modifier PRIVATE = new Modifier("private");
                public final static Modifier PUBLIC = new Modifier("public");
                public final static Modifier PROTECTED = new Modifier("protected");
                public final static Modifier STATIC = new Modifier("static");
                public final static Modifier FINAL = new Modifier("final");
                public final static Modifier ABSTRACT = new Modifier("abstract");
                public final static Modifier VOLATILE = new Modifier("volatile");
                public final static Modifier TRANSIENT = new Modifier("transient");
                public final static Modifier NATIVE = new Modifier("native");
                public final static Modifier STRICTFP = new Modifier("strictfp");
                public final static Modifier SYNCHRONIZED = new Modifier("synchronized");

                /**
                 * @param name
                 */
                public Modifier(String name) {
                                super(name);
                }
}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import org.bog.lachesis.metamodel.impl.AbstractNamed;

public class Named extends AbstractNamed {
                /**
                 * @param name
                 */
                public Named(String name) {
                   super(name);
                }


}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import org.bog.lachesis.metamodel.impl.AbstractNamespace;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APackageDeclaration;

public class Package extends AbstractNamespace  {
                /**
                 * @param name
                 */
                public Package(String name) {
                                super(name);
                }

                /*
                 * (non-Javadoc)
                 *
                 * @see bog.lachesis.model.independent.AbstractNamed#parse()
                 */
                public void parse(APackageDeclaration packageDeclaration) {
                                setName(packageDeclaration.getName().toString());
                }
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import org.bog.lachesis.metamodel.impl.AbstractParameter;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AFormalParameter;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIdentifierVariableDeclaratorId;
```

```
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclaratorId;


public class Parameter extends AbstractParameter
{
   /**
    * @param name
    */
   public Parameter(String name)
   {
      super(name);
   }

                private void parsePVariableDeclaratorId(PVariableDeclaratorId variableDeclaratorId) throws Exception {
                        if (variableDeclaratorId instanceof AVariableDeclaratorIdVariableDeclaratorId) {
                                AVariableDeclaratorIdVariableDeclaratorId declaratorId = (AVariableDeclaratorIdVariableDeclaratorId) variableDeclaratorId;
                                parsePVariableDeclaratorId(declaratorId.getVariableDeclaratorId());
                        } else if (variableDeclaratorId instanceof AIdentifierVariableDeclaratorId) {
                                AIdentifierVariableDeclaratorId declaratorId = (AIdentifierVariableDeclaratorId)
variableDeclaratorId;
                                setName(declaratorId.getIdentifier().getText());
                        } else
                                throw new Exception("Unknown type here: " + variableDeclaratorId);
                }

   /*
    * (non-Javadoc)
    *
    * @see bog.lachesis.model.independent.AbstractNamed#parse()
    */
   public void parse(AFormalParameter formalParameter) throws Exception
   {
                parsePVariableDeclaratorId(formalParameter.getVariableDeclaratorId());
      Type type = new Type(null);
      type.parse(formalParameter.getType());
      setType(type);
   }
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.io.File;
import java.io.IOException;
import java.util.jar.JarFile;

import org.apache.log4j.Logger;
import org.bog.lachesis.metamodel.ILocation;
import org.bog.lachesis.metamodel.impl.AbstractProject;

public class Project extends AbstractProject {
                private final static Logger log = Logger.getLogger(Project.class);

                private transient JarFile jarFile;

                /**
                 * @param name
                 */
                public Project(String name, ILocation location) {
                        super(name, location);
                        try {
                                File file = ((JavaProjectLocation) location).getFile();
                                if (file != null)
                                        jarFile = new JarFile(file);
                        } catch (IOException e) {
                                log.error("EXCEPTION:", e);
                                e.printStackTrace();
                        }
                }

                /*
                 * (non-Javadoc)
                 *
                 * @see bog.lachesis.model.independent.AbstractNamed#parse()
                 */
                public void parse() {
                }

                public JarFile getJarFile() {
                        return jarFile;
                }
}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import java.util.ArrayList;
import java.util.List;

import org.bog.lachesis.metamodel.IAttribute;
import org.bog.lachesis.metamodel.IVariable;
```

```
import org.bog.lachesis.metamodel.impl.AbstractReference;

public class Reference extends AbstractReference {
		private IAttribute targetAttribute;
		private IVariable targetVariable;

		private List identifiers = new ArrayList();

		public Reference(String name) {
			super(name);
		}

		public IAttribute getTargetAttribute() {
			return targetAttribute;
		}

		public void setTargetAttribute(IAttribute attribute) {
			this.targetAttribute = attribute;
		}

		public IVariable getTargetVariable() {
			return targetVariable;
		}

		public void setTargetVariable(IVariable targetVariable) {
			this.targetVariable = targetVariable;
		}

		private String extractIdentifierName(String name) {
			name = name.replaceAll(" ", "");
			int dotPos = name.indexOf(".");
			if (dotPos > -1) {
				return name.substring(0, dotPos);
			} else
				return name;
		}

		public void parse(Object node) throws Exception {

			String[] split = node.toString().split("\\.");
			for (int i=0;i <split.length; i++)
				identifiers.add(split[i].replaceAll(" ", ""));

			/*
			if (node instanceof ANameLeftHandSide) {
				parseANameLeftHandSide((ANameLeftHandSide) node);
			} else if (node instanceof ANamePostfixExpression) {
				parseANamePostfixExpression((ANamePostfixExpression) node);
			} else if (node instanceof APrimaryFieldAccess) {
				parseAPrimaryFieldAccess((APrimaryFieldAccess) node);
			} else
				throw new Exception("Unknown type here: " + node);
				*/
		}



		public String toString() {
			return identifiers.toString();
		}

		public List getIdentifiers() {
			return identifiers;
		}
}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;


import org.bog.lachesis.metamodel.impl.AbstractType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.APrimitiveTypeType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AReferenceTypeType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PType;

public class Type extends AbstractType {

		private final static String INT = "int";

		private final static String LONG_INT = "long int";

		private final static String SHORT_INT = "short int";

		private final static String FLOAT = "float";

		private final static String CHAR = "char";

		private final static String BYTE = "byte";

		private final static String DOUBLE = "double";
```

```
	private final static String STRING = "String";

	private final static String OBJECT = "Object";

	/**
	 * @param name
	 */
	public Type(String name) {
		super(name);
	}

	public boolean isPrimitive() {
		String name = getName();
		return (name.equals(Type.DOUBLE) || name.equals(Type.FLOAT) || name.equals(Type.INT) || name.equals(Type.LONG_INT)
				|| name.equals(Type.SHORT_INT) || name.equals(Type.STRING) || name.equals(Type.CHAR) || name.equals
(Type.BYTE));
	}

	/*
	 * (non-Javadoc)
	 *
	 * @see bog.lachesis.model.independent.AbstractNamed#parse()
	 */

	public void parse(PType type) throws Exception {
		if (type == null)
			setName("void");
		else {
			if (type instanceof APrimitiveTypeType) {
				setName(((APrimitiveTypeType) type).getPrimitiveType().toString());
			} else if (type instanceof AReferenceTypeType) {
				setName(((AReferenceTypeType) type).getReferenceType().toString());
			} else
				throw new Exception("Unknown type here: " + type);
		}
	}

}

package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

public class TypeList {
	public static boolean isInList(int x, int[] isIn) {
		for (int i = 0; i < isIn.length; i++)
			if (x == isIn[i])
				return true;
		return false;
	}

}


package org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst;

import org.bog.lachesis.metamodel.impl.AbstractVariable;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AAssignVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AIdentifierVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.AVariableDeclaratorIdVariableDeclaratorId;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PType;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclarator;
import org.bog.lachesis.recognizers.sourcecode.java.sablecc.node.PVariableDeclaratorId;

public class Variable extends AbstractVariable {
	/**
	 * @param name
	 */
	public Variable(String name) {
		super(name);
	}

	private void parsePVariableDeclaratorId(PVariableDeclaratorId variableDeclaratorId) throws Exception {
		if (variableDeclaratorId instanceof AVariableDeclaratorIdVariableDeclaratorId) {
			AVariableDeclaratorIdVariableDeclaratorId declaratorId = (AVariableDeclaratorIdVariableDeclaratorId) variableDeclaratorId;
			parsePVariableDeclaratorId(declaratorId.getVariableDeclaratorId());
		} else if (variableDeclaratorId instanceof AIdentifierVariableDeclaratorId) {
			AIdentifierVariableDeclaratorId declaratorId = (AIdentifierVariableDeclaratorId) variableDeclaratorId;
			setName(declaratorId.getIdentifier().getText());
		} else
			throw new Exception("Unknown type here: " + variableDeclaratorId);
	}

	public void parse(PVariableDeclaratorId variableDeclaratorId, PType ptype) throws Exception {
		parsePVariableDeclaratorId(variableDeclaratorId);
		Type type = new Type(null);
		type.parse(ptype);
		setType(type);
	}
```

```
		/*
		 * (non-Javadoc)
		 *
		 * @see bog.lachesis.model.independent.AbstractNamed#parse()
		 */
		public void parse(PVariableDeclarator variableDeclarator, PType ptype) throws Exception {
			PVariableDeclaratorId declaratorId = null;
			if (variableDeclarator instanceof AVariableDeclaratorIdVariableDeclarator) {
				declaratorId = ((AVariableDeclaratorIdVariableDeclarator) variableDeclarator).getVariableDeclaratorId();
			} else if (variableDeclarator instanceof AAssignVariableDeclarator) {
				declaratorId = ((AAssignVariableDeclarator) variableDeclarator).getVariableDeclaratorId();
			} else
				throw new Exception("Unknoen type here: " + variableDeclarator);
			parsePVariableDeclaratorId(declaratorId);
			parse(declaratorId, ptype);

		}

}
```

Пакет org.bog.lachesis.reports

```
package org.bog.lachesis.reports;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;

import org.bog.lachesis.analysis.tools.GlobalAnalysis;

public class AnalysisReport {
		private static AnalysisReport instance;

		private AnalysisReport() {
		}

		public final static AnalysisReport getInstance() {
			if (instance == null)
				instance = new AnalysisReport();
			return instance;
		}

		public void printAnalysisToFile(GlobalAnalysis globalAnalysis, File analysisReportFile) {
			FileWriter fw = null;
			try {
				fw = new FileWriter(analysisReportFile);
				globalAnalysis.toXML(0, "   ", new BufferedWriter(fw));
				fw.close();
				String zipFileName = analysisReportFile.getAbsolutePath() + "." + new Date().toString().replaceAll(":", "_") + ".zip";
				ZipTools.addToZip(zipFileName, new String[] { analysisReportFile.getAbsolutePath() });
			} catch (Throwable t) {
				if (fw != null)
					try {
						fw.close();
					} catch (IOException e) {
						e.printStackTrace();
					}
				t.printStackTrace();
			}

		}
}


package org.bog.lachesis.reports;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;

import org.bog.lachesis.metamodel.IModel;

public class ModelReport {
		private static ModelReport instance;

		private ModelReport() {
		}

		public final static ModelReport getInstance() {
			if (instance == null)
				instance = new ModelReport();
			return instance;
		}
		public void printModelToFile(IModel model, File modelReportFile) {
			FileWriter fw = null;
			try {
```

```
                                        fw = new FileWriter(modelReportFile);
                                        model.toXML(0, "   ", new BufferedWriter(fw));
                                        fw.close();
                                        String zipFileName = modelReportFile.getAbsolutePath() + "." + new Date().toString().replaceAll(":", "_") + ".zip";
                                        ZipTools.addToZip(zipFileName, new String[] { modelReportFile.getAbsolutePath() });
                                } catch (Throwable t) {
                                        if (fw != null)
                                                try {
                                                        fw.close();
                                                } catch (IOException e) {
                                                        e.printStackTrace();
                                                }
                                        t.printStackTrace();
                                }
                        }
}

package org.bog.lachesis.reports;

public class XMLTools {
                public final static String DEFAULT_ENCODING = "windows-1251";
                public static String escapeForXML(String str) {
                        StringBuffer newstr = new StringBuffer();
                        if (str == null)
                                return null;
                        int len = str.length();
                        for (int i = 0; i < len; i++) {
                                char ch = str.charAt(i);
                                switch (ch) {
                                case '&':
                                        newstr.append("&amp;");
                                        break;
                                case '<':
                                        newstr.append("&lt;");
                                        break;
                                case '>':
                                        newstr.append("&gt;");
                                        break;
                                case '\'':
                                        newstr.append("&apos;");
                                        break;
                                case '\"':
                                        newstr.append("&quot;");
                                        break;
                                default:
                                        newstr.append(ch);
                                }
                        }
                        return newstr.toString();
                }
}


package org.bog.lachesis.reports;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;

import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

public class XSLTransformer {
                public static void transform(String inputXmlFileName, String xslFileName, String outputXmlFileName) throws TransformerException,
TransformerConfigurationException, FileNotFoundException, IOException {
                        File inputXmlFile = new File(inputXmlFileName);
                        File xslFile =  new File(xslFileName);
                        File outputXmlFile =  new File(outputXmlFileName);
                        transform(inputXmlFile, xslFile, outputXmlFile);
                }
                public static void transform(File inputXmlFile, File xslFile, File outputXmlFile) throws TransformerException, TransformerConfigurationException,
FileNotFoundException, IOException {
                        if(!outputXmlFile.exists())
                                outputXmlFile.createNewFile();
                        TransformerFactory tFactory = TransformerFactory.newInstance();
                        Transformer transformer = tFactory.newTransformer(new StreamSource(xslFile));
                        OutputStreamWriter outputStreamWriter = new OutputStreamWriter(new FileOutputStream(outputXmlFile), "windows-1252");
                        StreamResult streamResult = new StreamResult(new FileOutputStream(outputXmlFile));
                        transformer.transform(new StreamSource(inputXmlFile), streamResult);
                }
                public static void main(String[] args) throws TransformerConfigurationException, FileNotFoundException, TransformerException, IOException {
                        if (args.length !=3) {
                                System.out.println("Usage: XSTLTransformer <input XML file> <XSL file> <output XML file>");
                                return;
```

```
                }
                transform(args[0], args[1], args[2]);
        }

}

package org.bog.lachesis.reports;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;

import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

public class XSLTransformer {
        public static void transform(String inputXmlFileName, String xslFileName, String outputXmlFileName) throws TransformerException,
TransformerConfigurationException, FileNotFoundException, IOException {
                File inputXmlFile = new File(inputXmlFileName);
                File xslFile =  new File(xslFileName);
                File outputXmlFile =  new File(outputXmlFileName);
                transform(inputXmlFile, xslFile, outputXmlFile);
        }
        public static void transform(File inputXmlFile, File xslFile, File outputXmlFile) throws TransformerException, TransformerConfigurationException,
FileNotFoundException, IOException {
                if(!outputXmlFile.exists())
                        outputXmlFile.createNewFile();
                TransformerFactory tFactory = TransformerFactory.newInstance();
                Transformer transformer = tFactory.newTransformer(new StreamSource(xslFile));
                OutputStreamWriter outputStreamWriter = new OutputStreamWriter(new FileOutputStream(outputXmlFile), "windows-1252");
                StreamResult streamResult = new StreamResult(new FileOutputStream(outputXmlFile));
                transformer.transform(new StreamSource(inputXmlFile), streamResult);
        }
        public static void main(String[] args) throws TransformerConfigurationException, FileNotFoundException, TransformerException, IOException {
                if(args.length !=3) {
                        System.out.println("Usage: XSTLTransformer <input XML file> <XSL file> <output XML file>");
                        return;
                }
                transform(args[0], args[1], args[2]);
        }

}
```

Пакет org.bog.lachesis.tools

```
package org.bog.lachesis.tools;

import org.bog.lachesis.analysis.AbstractAnalysisMonitor;

public class ConsolePrintMonitor extends AbstractAnalysisMonitor {
        private String subTask;

        public void processingPart(int worked) {
                System.out.println(".");
        }

        public boolean isCanceled() {
                return false;
        }

        protected void setBeginTaskImpl(String taskName, int count) {
                System.out.println("Start tast: " + taskName + " (" + count + ")");
        }

        protected void setTaskImpl(String taskName) {
                System.out.println(taskName);
        }

        public void setSubTask(String taskName) {
                subTask = taskName;
                System.out.println(taskName);
        }

        public void setSubTaskAddition(String addition) {
                System.out.println(subTask + addition);
        }

}


package org.bog.lachesis.tools;

import org.bog.lachesis.analysis.AbstractAnalysisMonitor;
```

```
public class NullAnalysisMonitor extends AbstractAnalysisMonitor {

		public void processingPart(int worked) {

		}

		public boolean isCanceled() {
				return false;
		}

		protected void setBeginTaskImpl(String taskName, int count) {

		}

		protected void setTaskImpl(String taskName) {

		}

		public void setSubTask(String taskName) {

		}

		public void setSubTaskAddition(String addition) {

		}

}
```

Пакет org.bog.lachesis.eclipse.plugin

```
package org.bog.lachesis.eclipse.plugin;

import java.util.HashMap;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.tools.GlobalAnalysis;
import org.bog.lachesis.eclipse.plugin.jobs.AnalysisRequest;
import org.bog.lachesis.metamodel.IModel;

public class LachesisModelPool {
		private static Logger log = Logger.getLogger(LachesisModelPool.class);


		public final static LachesisModelPool lachesisModelPool = new LachesisModelPool();

		public HashMap modelPool = new HashMap();

		public HashMap globalAnalysisPool = new HashMap();

		private LachesisModelPool() {
				log.debug("Lachesis Model Pool was instantiated."); //$NON-NLS-1$
		}

		public static LachesisModelPool getInstance() {
				return lachesisModelPool;
		}


		/**
		 * @return Returns the globalAnalysisPool.
		 */
		public GlobalAnalysis getGlobalAnalysis(AnalysisRequest analysisRequest) {
				return (GlobalAnalysis) globalAnalysisPool.get(analysisRequest);
		}

		/**
		 * @return Returns the modelPool.
		 */
		public IModel getModel(AnalysisRequest analysisRequest) {
				return (IModel) modelPool.get(analysisRequest);
		}

		public void addModel(AnalysisRequest analysisRequest, IModel model) {
				modelPool.put(analysisRequest, model);
				log.debug("Added model in pool for: "+analysisRequest.getName()); //$NON-NLS-1$
		}
		public void addGlobalAnalysis(AnalysisRequest analysisRequest, GlobalAnalysis globalAnalysis) {
				globalAnalysisPool.put(analysisRequest, globalAnalysis);
				log.debug("Added global analysis in pool for: "+analysisRequest.getName()); //$NON-NLS-1$
		}
		public void removeModel(AnalysisRequest analysisRequest) {
				modelPool.remove(analysisRequest);
				log.debug("Removed model from pool for: "+analysisRequest.getName()); //$NON-NLS-1$
		}
		public void removeGlobalAnalysis(AnalysisRequest analysisRequest) {
				globalAnalysisPool.remove(analysisRequest);
				log.debug("Removed global analysis from pool for: "+analysisRequest.getName()); //$NON-NLS-1$
```

```
		}
}

package org.bog.lachesis.eclipse.plugin;

import java.util.HashMap;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.tools.GlobalAnalysis;
import org.bog.lachesis.eclipse.plugin.jobs.AnalysisRequest;
import org.bog.lachesis.metamodel.IModel;

public class LachesisModelPool {
		private static Logger log = Logger.getLogger(LachesisModelPool.class);


		public final static LachesisModelPool lachesisModelPool = new LachesisModelPool();

		public HashMap modelPool = new HashMap();

		public HashMap globalAnalysisPool = new HashMap();

		private LachesisModelPool() {
			log.debug("Lachesis Model Pool was instantiated."); //$NON-NLS-1$
		}

		public static LachesisModelPool getInstance() {
			return lachesisModelPool;
		}


		/**
		 * @return Returns the globalAnalysisPool.
		 */
		public GlobalAnalysis getGlobalAnalysis(AnalysisRequest analysisRequest) {
			return (GlobalAnalysis) globalAnalysisPool.get(analysisRequest);
		}

		/**
		 * @return Returns the modelPool.
		 */
		public IModel getModel(AnalysisRequest analysisRequest) {
			return (IModel) modelPool.get(analysisRequest);
		}

		public void addModel(AnalysisRequest analysisRequest, IModel model) {
			modelPool.put(analysisRequest, model);
			log.debug("Added model in pool for: "+analysisRequest.getName()); //$NON-NLS-1$
		}
		public void addGlobalAnalysis(AnalysisRequest analysisRequest, GlobalAnalysis globalAnalysis) {
			globalAnalysisPool.put(analysisRequest, globalAnalysis);
			log.debug("Added global analysis in pool for: "+analysisRequest.getName()); //$NON-NLS-1$
		}
		public void removeModel(AnalysisRequest analysisRequest) {
			modelPool.remove(analysisRequest);
			log.debug("Removed model from pool for: "+analysisRequest.getName()); //$NON-NLS-1$
		}
		public void removeGlobalAnalysis(AnalysisRequest analysisRequest) {
			globalAnalysisPool.remove(analysisRequest);
			log.debug("Removed global analysis from pool for: "+analysisRequest.getName()); //$NON-NLS-1$
		}
}

package org.bog.lachesis.eclipse.plugin;

import java.util.MissingResourceException;
import java.util.ResourceBundle;

public class Messages {
		private static final String BUNDLE_NAME = "org.bog.lachesis.eclipse.plugin.messages"; //$NON-NLS-1$

		private static final ResourceBundle RESOURCE_BUNDLE = ResourceBundle.getBundle(BUNDLE_NAME);

		private Messages() {
		}

		public static String getString(String key) {
			try {
				return RESOURCE_BUNDLE.getString(key);
			} catch (MissingResourceException e) {
				return '!' + key + '!';
			}
		}
		/***
		 * TODO : analyze this
		 *
		 */
		public static String deleteMeLater(String key) {
			try {
				return RESOURCE_BUNDLE.getString(key);
```

```
                                        } catch (MissingResourceException e) {
                                                return '!' + key + '!';
                                        }
                        }
}
```

Пакет org.bog.lachesis.eclipse.plugin.editors

```
package org.bog.lachesis.eclipse.plugin.editors;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.analysis.AbstractAnalysisMonitor;
import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.IAnalysisPackage;
import org.bog.lachesis.analysis.tools.AnalysisRecord;
import org.bog.lachesis.analysis.tools.GlobalAnalysis;
import org.bog.lachesis.eclipse.plugin.LachesisModelPool;
import org.bog.lachesis.eclipse.plugin.LachesisPlugin;
import org.bog.lachesis.eclipse.plugin.Messages;
import org.bog.lachesis.eclipse.plugin.jobs.AnalysisRequest;
import org.bog.lachesis.eclipse.plugin.metamodel.AbstractNamed;
import org.bog.lachesis.eclipse.plugin.metamodel.Method;
import org.bog.lachesis.eclipse.plugin.metamodel.Model;
import org.bog.lachesis.eclipse.plugin.metamodel.Namespace;
import org.bog.lachesis.eclipse.plugin.metamodel.ProgramUnit;
import org.bog.lachesis.eclipse.plugin.metamodel.Project;
import org.bog.lachesis.eclipse.plugin.preferences.AnalysisPackagesPage;
import org.bog.lachesis.eclipse.plugin.preferences.LachesisPreferences;
import org.bog.lachesis.eclipse.plugin.views.AnalysisMonitor;
import org.bog.lachesis.eclipse.plugin.views.AnalysisPackagePropertySource;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.profiler.Profiler;
import org.eclipse.core.resources.IMarker;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.resources.IWorkspaceRoot;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.resources.WorkspaceJob;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.Status;
import org.eclipse.jdt.ui.ISharedImages;
import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.IMenuListener;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.action.MenuManager;
import org.eclipse.jface.action.Separator;
import org.eclipse.jface.preference.IPreferenceNode;
import org.eclipse.jface.preference.IPreferencePage;
import org.eclipse.jface.preference.PreferenceDialog;
import org.eclipse.jface.preference.PreferenceManager;
import org.eclipse.jface.preference.PreferenceNode;
import org.eclipse.jface.viewers.DoubleClickEvent;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.viewers.ILabelProviderListener;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.ITableColorProvider;
import org.eclipse.jface.viewers.ITableLabelProvider;
import org.eclipse.jface.viewers.ITreeContentProvider;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.jface.viewers.ViewerSorter;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Color;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Menu;
import org.eclipse.swt.widgets.Tree;
import org.eclipse.swt.widgets.TreeColumn;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorSite;
import org.eclipse.ui.IWorkbenchActionConstants;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.ide.IDE;
import org.eclipse.ui.part.EditorPart;
import org.eclipse.ui.views.properties.IPropertySource;

public class LachesisEditor extends EditorPart {
```

```
	public final static String ID = LachesisPlugin.ID + ".editors.LachesisEditor"; //$NON-NLS-1$

	private final List columnsConf = new ArrayList();

	private TreeViewer viewer;

	private AnalysisTreeContentProvider contentProvider;

	private Action actionShowProjects;

	private Action actionShowPackages;

	private Action actionShowClasses;

	private Action actionShowMethods;

	private Action actionFilterColumns;

	private Action viewProperties;

	private IModel modelSelected;

	private AnalysisRequest analysisRequest;

	private IResource resource;

	private GlobalAnalysis globalAnalysis;

	public static final String LACHESIS_MARKER_ID = LachesisPlugin.ID + ".problemmarker"; //$NON-NLS-1$

	public static final String VIEW_ID = LachesisPlugin.ID + ".views.LachesisView"; //$NON-NLS-1$

	/*
	 * The content provider class is responsible for providing objects to the
	 * view. It can wrap existing objects in adapters or simply return objects
	 * as-is. These objects may be sensitive to the current input of the view,
	 * or ignore it and always show the same content (like Task List, for
	 * example).
	 */

	private void prepareColumnConf() {
			LachesisPreferences.composeListOfEnabledAnalysisResults(columnsConf);
	}

	class TreeObject implements IAdaptable {

			private String name;

			private Object data;

			private TreeParent parent;

			private IPropertySource propertySource;

			public TreeObject(String name, Object data) {
					this.name = name;
					this.data = data;
			}

			public String getName() {
					return name;
			}

			public Object getData() {
					return data;
			}

			public void setParent(TreeParent parent) {
					this.parent = parent;
			}

			public TreeParent getParent() {
					return parent;
			}

			public String toString() {
					return getName();
			}

			/**
			 * @see org.eclipse.core.runtime.IAdaptable#getAdapter(java.lang.Class)
			 */
			public Object getAdapter(java.lang.Class adapter) {
					if (adapter == IPropertySource.class) {
							if (propertySource == null) {
									// cache the buttonelementpropertysource
									propertySource = new AnalysisPackagePropertySource(globalAnalysis.getAnalysisRecord(data),
columnsConf);
							}
							return propertySource;
```

```
                        }
                        return null;
                }

}

class TreeParent extends TreeObject {
                private ArrayList children;

                public TreeParent(String name, Object data) {
                        super(name, data);
                        children = new ArrayList();
                }

                public void addChild(TreeObject child) {
                        children.add(child);
                        child.setParent(this);
                }

                public void removeChild(TreeObject child) {
                        children.remove(child);
                        child.setParent(null);
                }

                public TreeObject[] getChildren() {
                        return (TreeObject[]) children.toArray(new TreeObject[children.size()]);
                }

                public boolean hasChildren() {
                        return children.size() > 0;
                }
}

class AnalysisTreeContentProvider implements IStructuredContentProvider, ITreeContentProvider {
                private TreeParent invisibleRoot;

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

                public void dispose() {
                }

                public Object[] getElements(Object parent) {
                        if (parent.equals(getSite())) {
                                return getChildren(invisibleRoot);
                        }
                        return getChildren(parent);
                }

                public Object getParent(Object child) {
                        if (child instanceof TreeObject) {
                                return ((TreeObject) child).getParent();
                        }
                        return null;
                }

                public Object[] getChildren(Object parent) {
                        if (parent instanceof TreeParent) {
                                return ((TreeParent) parent).getChildren();
                        }
                        return new Object[0];
                }

                public boolean hasChildren(Object parent) {
                        if (parent instanceof TreeParent)
                                return ((TreeParent) parent).hasChildren();
                        return false;
                }

                public WorkspaceJob createTree() {
                        WorkspaceJob job = new WorkspaceJob(Messages.getString("LachesisEditor.1")) { //$NON-NLS-1$
                                private final static String PROFILED_TASK = "Create Tree Job"; //$NON-NLS-1$

                                public IStatus runInWorkspace(IProgressMonitor monitor) throws CoreException {
                                        Profiler.startProfile(PROFILED_TASK);
                                        AnalysisMonitor analysisMonitor = new AnalysisMonitor(monitor);
                                        try {
                                                createTree(analysisMonitor);
                                                Profiler.stopProfile(PROFILED_TASK);
                                        } catch (InterruptedException e) {
                                                Profiler.stopProfile(PROFILED_TASK);
                                                e.printStackTrace();
                                                return Status.CANCEL_STATUS;
                                        }
                                        Profiler.logStatus();
                                        return Status.OK_STATUS;
                                }
                        };
                        job.setUser(true);
                        job.schedule();
```

```
                return job;
        }

        private void createTree(AbstractAnalysisMonitor analysisMonitor) throws InterruptedException {
                if (modelSelected == null)
                        return;

                Model uiModel = ModelTools.convertToUIModel(modelSelected, globalAnalysis);

                modelSelected = null;

                TreeParent root = new TreeParent("Model", uiModel); //$NON-NLS-1$
                invisibleRoot = new TreeParent("", null); //$NON-NLS-1$
                invisibleRoot.addChild(root);

                // createMarkers = LachesisPlugin.openQuestion("Are you sure you
                // want to create " + statusMessages + " ecilpse markers?",
                // new Shell(PlatformUI.getWorkbench().getDisplay()));
                analysisMonitor.setTask(Messages.getString("LachesisEditor.2")); //$NON-NLS-1$

                Iterator it = uiModel.getProjects().iterator();
                while (it.hasNext()) {
                        Project project = (Project) it.next();
                        TreeParent projectInTree = new TreeParent(project.getName(), project);
                        root.addChild(projectInTree);
                        Iterator namespacesIterator = project.getNamespaces().iterator();
                        while (namespacesIterator.hasNext()) {
                                Namespace namespace = (Namespace) namespacesIterator.next();
                                TreeParent namespaceInTree = new TreeParent(namespace.getName(), namespace);
                                projectInTree.addChild(namespaceInTree);
                                Iterator projectsIterator = namespace.getProgramUnits().iterator();
                                while (projectsIterator.hasNext()) {
                                        ProgramUnit programUnit = (ProgramUnit) projectsIterator.next();
                                        // analysisMonitor.processingResource(programUnit.getName(),
                                        // 1);
                                        if (analysisMonitor.isCanceled())
                                                throw new InterruptedException("canceled"); //$NON-NLS-1$
                                        TreeParent namedInTree = new TreeParent(programUnit.getName(), programUnit);
                                        namespaceInTree.addChild(namedInTree);
                                        Iterator methodsIterator = programUnit.getMethods().iterator();
                                        String resourceLocation = null;
                                        // resourceLocation =
                                        // programUnit.getAbsoluteLocation().getName();

                                        while (methodsIterator.hasNext()) {
                                                Method method = (Method) methodsIterator.next();
                                                TreeObject methodInTree = new TreeObject(method.getName(),
method);
                                                namedInTree.addChild(methodInTree);
                                        }
                                }
                        }
                }
        }

        private static final Image image = PlatformUI.getWorkbench().getSharedImages()
                        .getImage(org.eclipse.ui.ISharedImages.IMG_OBJS_ERROR_TSK); // new

        private double readable(double param) {
                param = param * 1000;
                return Math.ceil(param) / 1000.;
        }

        class AnalysisTreeLabelProvider implements ITableLabelProvider, ITableColorProvider {

                public String getText(Object obj) {
                        return obj.toString();
                }

                public Image getImage(Object obj) {
                        String imageKey = org.eclipse.ui.ISharedImages.IMG_OBJ_ELEMENT;
                        if (obj instanceof TreeParent)
                                imageKey = org.eclipse.ui.ISharedImages.IMG_OBJ_FOLDER;
                        return PlatformUI.getWorkbench().getSharedImages().getImage(imageKey);
                }

                /*
                 * (non-Javadoc)
                 *
                 * @see org.eclipse.jface.viewers.ITableLabelProvider#getColumnImage(java.lang.Object,
                 *      int)
                 */
                public Image getColumnImage(Object element, int columnIndex) {
                        try {
                                if (columnIndex == 0)
                                        return null;

                                element = ((TreeObject) element).getData();
```

```
                                        if (!ModelTools.isIncludedInAnalysis(element))
                                                return null;

                                        AnalysisDescriptor descriptor = (AnalysisDescriptor) columnsConf.get(columnIndex - 1);
                                        AnalysisPackageDescriptor packageDescriptor = descriptor.getAnalysisPackageDescriptor();

                                        AnalysisRecord analysisRecord = globalAnalysis.getAnalysisRecord(element);
                                        if (analysisRecord != null) {
                                                IAnalysisPackage analysisPackage = analysisRecord.getAnalysisPackage(packageDescriptor);
                                                if (analysisPackage != null) {
                                                        if (analysisPackage.getStatus().hasErrors())
                                                                return image;
                                                }
                                        }
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                                return null;
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see org.eclipse.jface.viewers.ITableLabelProvider#getColumnText(java.lang.Object,
                         *      int)
                         */
                        public String getColumnText(Object element, int columnIndex) {
                                String result = null;
                                try {
                                        element = ((TreeObject) element).getData();
                                        if (columnIndex == 0)
                                                result = ((AbstractNamed) element).getName();
                                        else {
                                                if (!ModelTools.isIncludedInAnalysis(element))
                                                        return ""; //$NON-NLS-1$
                                                AnalysisDescriptor descriptor = (AnalysisDescriptor) columnsConf.get(columnIndex - 1);
                                                AnalysisPackageDescriptor packageDescriptor = descriptor.getAnalysisPackageDescriptor();
                                                AnalysisRecord analysisRecord = globalAnalysis.getAnalysisRecord(element);
                                                if (analysisRecord != null) {
                                                        IAnalysisPackage analysisPackage = analysisRecord.getAnalysisPackage
(packageDescriptor);
                                                        if (analysisPackage != null) {
                                                                Object resultValue = analysisPackage.getAnalysisResult(descriptor);
                                                                if (resultValue == null)
                                                                        result = "<NA>";
                                                                else if (resultValue instanceof Double)
                                                                        result = String.valueOf(readable(((Double)
resultValue).doubleValue()));
                                                                else if (resultValue instanceof Float)
                                                                        result = String.valueOf(readable(((Float) resultValue).
doubleValue()));
                                                                else
                                                                        result = String.valueOf(resultValue);
                                                        }
                                                }
                                        }
                                } catch (RuntimeException e) {
                                        e.printStackTrace();
                                }
                                if (result == null)
                                        result = ""; //$NON-NLS-1$
                                return result;
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see org.eclipse.jface.viewers.IBaseLabelProvider#addListener(org.eclipse.jface.viewers.ILabelProviderListener)
                         */
                        public void addListener(ILabelProviderListener listener) {
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see org.eclipse.jface.viewers.IBaseLabelProvider#dispose()
                         */
                        public void dispose() {
                        }

                        /*
                         * (non-Javadoc)
                         *
                         * @see org.eclipse.jface.viewers.IBaseLabelProvider#isLabelProperty(java.lang.Object,
                         *      java.lang.String)
                         */
                        public boolean isLabelProperty(Object element, String property) {
                                return false;
                        }
```

```
		/*
		 * (non-Javadoc)
		 *
		 * @see org.eclipse.jface.viewers.IBaseLabelProvider#removeListener(org.eclipse.jface.viewers.ILabelProviderListener)
		 */
		public void removeListener(ILabelProviderListener listener) {
		}

		public Color getForeground(Object element, int columnIndex) {
			if (columnIndex > 0)
				return null;

			element = ((TreeObject) element).getData();
			if (!ModelTools.isIncludedInAnalysis(element))
				return null;
			AnalysisRecord analysisRecord = globalAnalysis.getAnalysisRecord(element);
			if ((analysisRecord != null) && analysisRecord.hasErrors())
				return Display.getCurrent().getSystemColor(SWT.COLOR_RED);
			return null;
		}

		public Color getBackground(Object element, int columnIndex) {
			element = ((TreeObject) element).getData();

			if (!ModelTools.isIncludedInAnalysis(element))
				return Display.getCurrent().getSystemColor(SWT.COLOR_WIDGET_LIGHT_SHADOW);

			if (columnIndex > 0) {
				AnalysisDescriptor analysisDescriptor = (AnalysisDescriptor) columnsConf.get(columnIndex - 1);
				return new Color(Display.getCurrent(), LachesisPreferences.getAnalysisDescriptorColumnColor(analysisDescriptor));
			}
			return null;
		}
}

class NameSorter extends ViewerSorter {

		public int compare(Viewer viewer, Object e1, Object e2) {
			Object de1 = ((TreeObject) e1).getData();
			if (de1 instanceof Project) {
				if (((Project) de1).isIncludedInReport())
					return -1;
				Object de2 = ((TreeObject) e2).getData();
				if (((Project) de2).isIncludedInReport())
					return 1;
			}

			return super.compare(viewer, e1, e2);
		}
		/*
		 * (non-Javadoc)
		 *
		 * @see org.eclipse.jface.viewers.ViewerSorter#compare(org.eclipse.jface.viewers.Viewer,
		 *      java.lang.Object, java.lang.Object)
		 */

		/*
		 * public int compare(Viewer viewer, Object e1, Object e2) { INamed
		 * element1 = (INamed) ((TreeObject) e1).getData(); INamed element2 =
		 * (INamed) ((TreeObject) e2).getData(); ModuleMetrics moduleMetrics1 =
		 * modelMetrics.getMetricsPackage( element1).getModuleMetrics();
		 * ModuleMetrics moduleMetrics2 = modelMetrics.getMetricsPackage(
		 * element2).getModuleMetrics();
		 *
		 * if ((moduleMetrics1 != null) || (moduleMetrics2 != null)) { if
		 * ((moduleMetrics1 != null) && (moduleMetrics2 != null)) { if
		 * (moduleMetrics1.getMaintainabilityIndex() > moduleMetrics2
		 * .getMaintainabilityIndex()) return -1; else return 1; } else if
		 * (moduleMetrics1 != null) return -1; else return 1; } return
		 * super.compare(viewer, e1, e2); }
		 */
}

private IResource StringToResource(String resourceLocation) {
		IResource result = null;
		try {
			if (resourceLocation != null) {
				IWorkspaceRoot workspaceRoot = ResourcesPlugin.getWorkspace().getRoot();
				IPath path = new Path(resourceLocation);
				result = workspaceRoot.getFileForLocation(path);
			}
		} catch (Exception e) {
			e.printStackTrace();
		}
		return result;
}

/**
 * This is a callback that will allow us to create the viewer and initialize
 * it.
```

```
 */

public void createPartControl(Composite parent) {

		viewer = new TreeViewer(parent, SWT.SINGLE | SWT.FULL_SELECTION | SWT.H_SCROLL | SWT.V_SCROLL);
		// drillDownAdapter = new DrillDownAdapter(viewer);
		contentProvider = new AnalysisTreeContentProvider();
		WorkspaceJob job = contentProvider.createTree();
		while (job.getResult() == null)
			try {
				Thread.sleep(300);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		viewer.setContentProvider(contentProvider);
		viewer.setLabelProvider(new AnalysisTreeLabelProvider());
		viewer.setSorter(new NameSorter());

		// Set up the table
		Tree tree = viewer.getTree();

		initializeTree(tree);

		refreshViewer();
		getSite().setSelectionProvider(viewer);

		makeActions();
		hookContextMenu();
		hookDoubleClickAction();
		contributeToActionBars();
}

void refreshViewer() {
		// Pack the window
		// viewer.pack();

		viewer.setInput(getSite());
		// Expand everything
		viewer.expandToLevel(2);
		// Scroll to top
		// if (viewer.getExpandedElements().length > 0)
		// viewer.reveal(viewer.getTree().getElementAt(0));

}

void initializeTree(Tree tree) {
		prepareColumnConf();

		tree.removeAll();

		// Pack the columns
		for (int i = tree.getColumnCount() - 1; i >= 0; i--) {
			tree.getColumn(i).dispose();
		}

		TreeColumn tc = new TreeColumn(tree, SWT.LEFT);
		tc.setText(Messages.getString("LachesisEditor.3")); //$NON-NLS-1$

		for (Iterator iter = columnsConf.iterator(); iter.hasNext();) {
			AnalysisDescriptor descriptor = (AnalysisDescriptor) iter.next();
			tc = new TreeColumn(tree, SWT.LEFT);
			tc.setText(descriptor.getName());
		}

		// Pack the columns
		for (int i = 0, n = tree.getColumnCount(); i < n; i++) {
			tree.getColumn(i).pack();
		}

		// Turn on the header and the lines
		tree.setHeaderVisible(true);
		tree.setLinesVisible(true);

}

private void hookContextMenu() {
		MenuManager menuMgr = new MenuManager("#PopupMenu"); //$NON-NLS-1$
		menuMgr.setRemoveAllWhenShown(true);
		menuMgr.addMenuListener(new IMenuListener() {
			public void menuAboutToShow(IMenuManager manager) {
				LachesisEditor.this.fillContextMenu(manager);
			}
		});
		Menu menu = menuMgr.createContextMenu(viewer.getControl());
		viewer.getControl().setMenu(menu);
		getSite().registerContextMenu(menuMgr, viewer);
}

private void contributeToActionBars() {
		// IActionBars bars = getViewSite().getActionBars();
```

```
		// fillLocalPullDown(bars.getMenuManager());
		// fillLocalToolBar(bars.getToolBarManager());
}

private void fillContextMenu(IMenuManager manager) {
		manager.add(actionShowProjects);
		manager.add(actionShowPackages);
		manager.add(actionShowClasses);
		manager.add(actionShowMethods);
		manager.add(actionFilterColumns);
		manager.add(new Separator());
		// Other plug-ins can contribute there actions here
		manager.add(new Separator(IWorkbenchActionConstants.MB_ADDITIONS));
}

private void goToLine(IMarker marker) {
		try {
			IDE.openEditor(PlatformUI.getWorkbench().getActiveWorkbenchWindow().getActivePage(), marker);
		} catch (Exception e) {
			e.printStackTrace();
		}
}

private void goToLine(String resourceLocation, int lineNumber) {
		IResource resource = StringToResource(resourceLocation);
		HashMap map = new HashMap();
		map.put(IMarker.LINE_NUMBER, new Integer(lineNumber));
		IMarker marker;
		try {
			if (resource != null) {
				marker = resource.createMarker(IMarker.TEXT);
				marker.setAttributes(map);
				goToLine(marker);
				marker.delete();
			}
		} catch (Exception e) {
			e.printStackTrace();
		}

}

private void makeActions() {

		actionShowProjects = new Action() {
			public void run() {
				viewer.collapseAll();
				viewer.expandToLevel(2);
			}
		};
		actionShowProjects.setText(Messages.getString("LachesisEditor.4")); //$NON-NLS-1$
		actionShowProjects.setToolTipText(Messages.getString("LachesisEditor.5")); //$NON-NLS-1$
		actionShowProjects.setImageDescriptor(PlatformUI.getWorkbench().getSharedImages().getImageDescriptor(
				IDE.SharedImages.IMG_OBJ_PROJECT));

		actionShowPackages = new Action() {
			public void run() {
				viewer.collapseAll();
				viewer.expandToLevel(3);
			}
		};
		actionShowPackages.setText(Messages.getString("LachesisEditor.6")); //$NON-NLS-1$
		actionShowPackages.setToolTipText(Messages.getString("LachesisEditor.7")); //$NON-NLS-1$
		actionShowPackages.setImageDescriptor(org.eclipse.jdt.ui.JavaUI.getSharedImages()
				.getImageDescriptor(ISharedImages.IMG_OBJS_PACKAGE));

		actionShowClasses = new Action() {
			public void run() {
				viewer.collapseAll();
				viewer.expandToLevel(4);
			}
		};
		actionShowClasses.setText(Messages.getString("LachesisEditor.8")); //$NON-NLS-1$
		actionShowClasses.setToolTipText(Messages.getString("LachesisEditor.9")); //$NON-NLS-1$
		actionShowClasses.setImageDescriptor(org.eclipse.jdt.ui.JavaUI.getSharedImages().getImageDescriptor(ISharedImages.IMG_OBJS_CLASS));

		actionShowMethods = new Action() {
			public void run() {
				viewer.expandToLevel(5);
			}
		};
		actionShowMethods.setText(Messages.getString("LachesisEditor.10")); //$NON-NLS-1$
		actionShowMethods.setToolTipText(Messages.getString("LachesisEditor.11")); //$NON-NLS-1$
		actionShowMethods.setImageDescriptor(org.eclipse.jdt.ui.JavaUI.getSharedImages().getImageDescriptor(ISharedImages.IMG_OBJS_PUBLIC));

		final TreeViewer finalViewer = viewer;
		actionFilterColumns = new Action() {
			public void run() {
				IPreferencePage page = new AnalysisPackagesPage();
				PreferenceManager mgr = new PreferenceManager();
				IPreferenceNode node = new PreferenceNode("1", page); //$NON-NLS-1$
```

```
                                                mgr.addToRoot(node);
                                                PreferenceDialog dialog = new PreferenceDialog(null, mgr);
                                                dialog.create();
                                                dialog.setMessage(page.getTitle());
                                                dialog.open();
                                                initializeTree(viewer.getTree());
                                                refreshViewer();
                                                finalViewer.refresh();
                                        }
                                };
                                actionFilterColumns.setText(Messages.getString("LachesisEditor.12")); //$NON-NLS-1$

                                viewProperties = new Action() {
                                        public void run() {
                                                ISelection selection = viewer.getSelection();
                                                Object obj = ((IStructuredSelection) selection).getFirstElement();
                                                Object element = ((TreeObject) obj).getData();
                                                IMarker marker = null;
                                                if (element instanceof ProgramUnit) {
                                                        goToLine(((ProgramUnit) element).getAbsoluteLocation().getName(), ((ProgramUnit) element).
getFirstLineNumber());
                                                } else if (element instanceof Method) {
                                                        ProgramUnit programUnit = ((Method) element).getEnclosingProgramUnit();
                                                        goToLine(programUnit.getAbsoluteLocation().getName(), ((Method) element).getFirstLineNumber
());
                                                }
                                                LachesisPlugin.getView("org.eclipse.ui.views.PropertySheet"); //$NON-NLS-1$
                                        }
                                };
                        }

                        private void hookDoubleClickAction() {
                                viewer.addDoubleClickListener(new IDoubleClickListener() {
                                        public void doubleClick(DoubleClickEvent event) {
                                                viewProperties.run();
                                        }
                                });
                                /*
                                 * viewer.addSelectionChangedListener(new ISelectionChangedListener() {
                                 * public void selectionChanged(SelectionChangedEvent event) {
                                 * viewProperties.run(); } });
                                 */
                        }

                        /**
                         * Passing the focus request to the viewer's control.
                         */
                        public void setFocus() {
                                viewer.getControl().setFocus();
                        }

                        public void refresh() {
                                viewer.refresh();
                        }

                        /*
                         * public void expandMethods() { Object object = viewer.getElementAt(0); if
                         * (object != null) { if (object instanceof TreeParent) {
                         * ((TreeParent)object).getData() } } }
                         */
                        public void deleteMarkers() {
                                int depth = IResource.DEPTH_INFINITE;
                                try {
                                        if (resource != null)
                                                resource.deleteMarkers(LACHESIS_MARKER_ID, true, depth);
                                } catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }

                        public void doSave(IProgressMonitor monitor) {

                        }

                        public void doSaveAs() {

                        }

                        public void init(IEditorSite site, IEditorInput input) throws PartInitException {
                                setSite(site);
                                setInput(input);
                                analysisRequest = ((LachesisInput) input).getAnalysisRequest();
                                modelSelected = LachesisModelPool.getInstance().getModel(analysisRequest);
                                globalAnalysis = LachesisModelPool.getInstance().getGlobalAnalysis(analysisRequest);
                                setPartName(analysisRequest.getName());
                        }

                        public boolean isDirty() {
                                return false;
                        }
```

```
		public boolean isSaveAsAllowed() {
			return false;
		}

		public void dispose() {
			LachesisModelPool.getInstance().removeModel(analysisRequest);
			LachesisModelPool.getInstance().removeGlobalAnalysis(analysisRequest);
			super.dispose();
		}
}


package org.bog.lachesis.eclipse.plugin.editors;

import org.bog.lachesis.eclipse.plugin.Messages;
import org.bog.lachesis.eclipse.plugin.jobs.AnalysisRequest;
import org.eclipse.core.resources.IStorage;
import org.eclipse.core.runtime.PlatformObject;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.ui.IPersistableElement;
import org.eclipse.ui.IStorageEditorInput;

public class LachesisInput extends PlatformObject implements IStorageEditorInput {
		private IStorage storage;

		public LachesisInput(IStorage storage) {
			this.storage = storage;
		}

		public boolean exists() {
			return true;
		}

		public ImageDescriptor getImageDescriptor() {
			return null;
		}

		public String getName() {
			return storage.getName();
		}

		public IPersistableElement getPersistable() {
			return null;
		}

		public AnalysisRequest getAnalysisRequest() {
			return ((LachesisStorage)storage).getAnalysisRequest();
		}
		public IStorage getStorage() {
			return storage;
		}

		public String getToolTipText() {
			return Messages.getString("LachesisInput.0") + storage.getName(); //$NON-NLS-1$
		}
}

package org.bog.lachesis.eclipse.plugin.editors;

import java.io.InputStream;

import org.bog.lachesis.eclipse.plugin.jobs.AnalysisRequest;
import org.eclipse.core.resources.IStorage;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.runtime.PlatformObject;

public class LachesisStorage extends PlatformObject

implements IStorage {
		private AnalysisRequest analysisRequest;

		public LachesisStorage(AnalysisRequest analysisRequest) {
			this.analysisRequest = analysisRequest;
		}

		public InputStream getContents() throws CoreException {
		//		return new ByteArrayInputStream(identifier.getBytes());
			return null;
		}

		public IPath getFullPath() {
			return null;
		}

		public String getName() {
			return analysisRequest.getName();
		}
```

```
	public boolean isReadOnly() {
		return true;
	}

	/**
	 * @return Returns the identifier.
	 */
	public AnalysisRequest getAnalysisRequest() {
		return analysisRequest;
	}

}

package org.bog.lachesis.eclipse.plugin.editors;

import java.util.HashMap;
import java.util.Iterator;

import org.bog.lachesis.analysis.tools.GlobalAnalysis;
import org.bog.lachesis.eclipse.plugin.metamodel.Method;
import org.bog.lachesis.eclipse.plugin.metamodel.Model;
import org.bog.lachesis.eclipse.plugin.metamodel.Namespace;
import org.bog.lachesis.eclipse.plugin.metamodel.ProgramUnit;
import org.bog.lachesis.eclipse.plugin.metamodel.Project;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IProject;

public class ModelTools {

	private static void replaceAnalysisResult(HashMap results, Object oldObject, Object newObject) {
		results.put(newObject, results.get(oldObject));
		results.remove(oldObject);
	}

	public static boolean isIncludedInAnalysis(Object element) {
		Project project = null;
		if (element instanceof Method) {
			project = ((Method) element).getEnclosingProgramUnit().getEnclosingNamespace().getEnclosingProject();
		} else if (element instanceof ProgramUnit) {
			project = ((ProgramUnit) element).getEnclosingNamespace().getEnclosingProject();
		} else if (element instanceof Namespace) {
			project = ((Namespace) element).getEnclosingProject();
		} else if (element instanceof Project) {
			project = (Project) element;
		}
		return !((project != null) && (!project.isIncludedInReport()));
	}

	public static Model convertToUIModel(IModel model, GlobalAnalysis globalAnalysis) throws InterruptedException {
		Model uiModel = new Model(model);
		HashMap analysis = globalAnalysis.getGlobalAnalysis();

		replaceAnalysisResult(analysis, model, uiModel);
		Iterator projectsIterator = model.getProjects().values().iterator();
		while (projectsIterator.hasNext()) {
			IProject project = (IProject) projectsIterator.next();
			Project uiProject = new Project(project);
			uiModel.addProject(uiProject);
			replaceAnalysisResult(analysis, project, uiProject);
			Iterator namespacesIterator = project.getNamespaces().values().iterator();
			while (namespacesIterator.hasNext()) {
				INamespace namespace = (INamespace) namespacesIterator.next();
				Namespace uiNamespace = new Namespace(namespace);
				uiProject.addNamespace(uiNamespace);
				replaceAnalysisResult(analysis, namespace, uiNamespace);
				Iterator programUnitsIterator = namespace.getProgramUnits().values().iterator();
				while (programUnitsIterator.hasNext()) {
					IProgramUnit programUnit = (IProgramUnit) programUnitsIterator.next();
					ProgramUnit uiProgramUnit = new ProgramUnit(programUnit);
					uiNamespace.addProgramUnit(uiProgramUnit);
					replaceAnalysisResult(analysis, programUnit, uiProgramUnit);
					Iterator methodsIterator = programUnit.getMethods().iterator();
					while (methodsIterator.hasNext()) {
						IMethod method = (IMethod) methodsIterator.next();
						Method uiMethod = new Method(method);
						uiProgramUnit.addMethod(uiMethod);
						replaceAnalysisResult(analysis, method, uiMethod);
					}
				}
			}
		}
		return uiModel;
	}

}
```

Пакет org.bog.lachesis.eclipse.plugin.jobs

```
package org.bog.lachesis.eclipse.plugin.jobs;

import org.eclipse.core.resources.WorkspaceJob;

public abstract class ALachesisWorkspaceJob extends WorkspaceJob {
		private AnalysisRequest analysisRequest;

		public ALachesisWorkspaceJob(String name, AnalysisRequest analysisRequest) {
				super(name);
				this.analysisRequest = analysisRequest;
		}

		/**
		 * @return Returns the identifier.
		 */
		public AnalysisRequest getAnalysisRequest() {
				return analysisRequest;
		}

}


package org.bog.lachesis.eclipse.plugin.jobs;

import org.bog.lachesis.analysis.tools.GlobalAnalysis;
import org.bog.lachesis.eclipse.plugin.LachesisModelPool;
import org.bog.lachesis.eclipse.plugin.preferences.PreferenceConstants;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.model.IModelSource;
import org.eclipse.core.resources.IResource;

public class AnalysisRequest {
		private String name;

		private Object report;

		private IResource analysisTargetRoot;

		private IModelSource modelSource;

		private boolean analyzeJars = true;

		private int createMarkers = PreferenceConstants.V_CHOICE_ASK;

		public AnalysisRequest() {
				super();
		}

		/**
		 * @return Returns the identifier.
		 */
		public Object getReport() {
				return report;
		}

		/**
		 * @param identifier
		 *		The identifier to set.
		 */
		public void setReport(Object report) {
				this.report = report;
		}

		/**
		 * @return Returns the name.
		 */
		public String getName() {
				return name;
		}

		/**
		 * @param name
		 *		The name to set.
		 */
		public void setName(String name) {
				this.name = name;
		}

		/**
		 * @return Returns the analyzeJars.
		 */
		public boolean isAnalyzeJars() {
				return analyzeJars;
		}

		/**
		 * @param analyzeJars
		 *		The analyzeJars to set.
		 */
```

```
	public void setAnalyzeJars(boolean analyzeJars) {
		this.analyzeJars = analyzeJars;
	}

	/**
	 * @return Returns the globalAnalysis.
	 */
	public GlobalAnalysis getGlobalAnalysis() {
		return LachesisModelPool.getInstance().getGlobalAnalysis(this);
	}

	/**
	 * @param globalAnalysis
	 *            The globalAnalysis to set.
	 */
	public void setGlobalAnalysis(GlobalAnalysis globalAnalysis) {
		LachesisModelPool.getInstance().addGlobalAnalysis(this, globalAnalysis);
	}

	/**
	 * @return Returns the model.
	 */
	public IModel getModel() {
		return LachesisModelPool.getInstance().getModel(this);
	}

	/**
	 * @param model
	 *            The model to set.
	 */
	public void setModel(IModel model) {
		LachesisModelPool.getInstance().addModel(this, model);
	}

	/**
	 * @return Returns the modelSource.
	 */
	public IModelSource getModelSource() {
		return modelSource;
	}

	/**
	 * @param modelSource
	 *            The modelSource to set.
	 */
	public void setModelSource(IModelSource modelSource) {
		this.modelSource = modelSource;
	}

	public void cleanModelAndGlobalAnalysis() {
		LachesisModelPool.getInstance().removeModel(this);
		LachesisModelPool.getInstance().removeGlobalAnalysis(this);
	}

	/**
	 * @return Returns the createMarkers.
	 */
	public int isCreateMarkers() {
		return createMarkers;
	}

	/**
	 * @param createMarkers The createMarkers to set.
	 */
	public void setCreateMarkers(int createMarkers) {
		this.createMarkers = createMarkers;
	}

	/**
	 * @return Returns the analysisTarget.
	 */
	public IResource getAnalysisTargetRoot() {
		return analysisTargetRoot;
	}

	/**
	 * @param analysisTarget The analysisTarget to set.
	 */
	public void setAnalysisTargetRoot(IResource analysisTarget) {
		this.analysisTargetRoot = analysisTarget;
	}
}


package org.bog.lachesis.eclipse.plugin.jobs;

import java.lang.reflect.InvocationTargetException;
import java.util.Iterator;

import org.apache.log4j.Logger;
```

```
import org.bog.lachesis.analysis.AbstractAnalysisMonitor;
import org.bog.lachesis.analysis.IAnalysisPackage;
import org.bog.lachesis.analysis.Status;
import org.bog.lachesis.analysis.StatusMessage;
import org.bog.lachesis.analysis.tools.GlobalAnalysis;
import org.bog.lachesis.eclipse.plugin.LachesisPlugin;
import org.bog.lachesis.eclipse.plugin.Messages;
import org.bog.lachesis.eclipse.plugin.editors.LachesisEditor;
import org.bog.lachesis.eclipse.plugin.editors.LachesisInput;
import org.bog.lachesis.eclipse.plugin.editors.LachesisStorage;
import org.bog.lachesis.eclipse.plugin.preferences.LachesisPreferences;
import org.bog.lachesis.eclipse.plugin.preferences.PreferenceConstants;
import org.bog.lachesis.eclipse.plugin.utils.ResourceUtils;
import org.bog.lachesis.eclipse.plugin.views.AnalysisMonitor;
import org.bog.lachesis.metamodel.ILinesOfCode;
import org.bog.lachesis.metamodel.IMethod;
import org.bog.lachesis.metamodel.IModel;
import org.bog.lachesis.metamodel.INamed;
import org.bog.lachesis.metamodel.INamespace;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IProject;
import org.bog.lachesis.profiler.Profiler;
import org.eclipse.core.resources.IMarker;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.resources.IStorage;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.IStorageEditorInput;
import org.eclipse.ui.IWorkbenchPage;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;

public class AnalyzeModelJob extends ALachesisWorkspaceJob {
	private static Logger log = Logger.getLogger(AnalyzeModelJob.class);

	private final static String PROFILED_TASK = "Analyze Model Job"; //$NON-NLS-1$

	public static final String LACHESIS_MARKER_ID = LachesisPlugin.ID + ".problemmarker"; //$NON-NLS-1$

	public AnalyzeModelJob(String name, AnalysisRequest identifier) {
		super(name, identifier);
	}

	private void openLachesisEditor(AnalysisRequest identifier) {
		final AnalysisRequest _ident = identifier;
		Display.getDefault().syncExec(new Runnable() {

			public void run() {
				IWorkbenchWindow window = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
				IStorage storage = new LachesisStorage(_ident);
				IStorageEditorInput input = new LachesisInput(storage);
				IWorkbenchPage page = window.getActivePage();
				if (page != null)
					try {
						LachesisEditor lachesisEditor = (LachesisEditor) page.openEditor(input,
LachesisEditor.ID);
					} catch (PartInitException e) {
						e.printStackTrace();
					}

			}
		});

	}

	public IStatus runInWorkspace(IProgressMonitor monitor) throws CoreException {
		final IProgressMonitor progressMonitor = monitor;
		Profiler.startProfile(PROFILED_TASK);
		try {
			GlobalAnalysis globalAnalysis = analyzeMetrics(getAnalysisRequest().getModel(), new AnalysisMonitor(progressMonitor));
			getAnalysisRequest().setGlobalAnalysis(globalAnalysis);
			Profiler.stopProfile(PROFILED_TASK);
			openLachesisEditor(getAnalysisRequest());
		} catch (InterruptedException e) {
			Profiler.stopProfile(PROFILED_TASK);
			return org.eclipse.core.runtime.Status.CANCEL_STATUS;
		} catch (Exception e) {
			Profiler.stopProfile(PROFILED_TASK);
			log.error("Analyze metrics failed", e); //$NON-NLS-1$
			e.printStackTrace();
			getAnalysisRequest().cleanModelAndGlobalAnalysis();
			LachesisPlugin.showErrorMessage(Messages.getString("AnalyzeModelJob.1")); //$NON-NLS-1$
			return org.eclipse.core.runtime.Status.CANCEL_STATUS;
		}
		// setFinishedSuccessfully(true);
		Profiler.logStatus();
		return org.eclipse.core.runtime.Status.OK_STATUS;
```

```
	}

	public GlobalAnalysis analyzeMetrics(IModel model, AbstractAnalysisMonitor analysisMonitor) throws InterruptedException,
			InvocationTargetException {
		GlobalAnalysis globalAnalysis = new GlobalAnalysis();
		globalAnalysis.analyzeMetrics(model, analysisMonitor);

		boolean createMarkers = (LachesisPreferences.getCreateMarkers() == PreferenceConstants.V_CHOICE_ENABLED);

		analysisMonitor.setTask(Messages.getString("AnalyzeModelJob.2")); //$NON-NLS-1$
		int statusMessages = countAllStatusMessages(globalAnalysis, model);

		if (LachesisPreferences.getCreateMarkers() == PreferenceConstants.V_CHOICE_ASK) {
			createMarkers = LachesisPlugin
					.openQuestion(Messages.getString("AnalyzeModelJob.3") + statusMessages + Messages.getString
("AnalyzeModelJob.4")); //$NON-NLS-1$ //$NON-NLS-2$
		}

		if (createMarkers)
			analysisMonitor.setBeginTask(
					Messages.getString("AnalyzeModelJob.5"), globalAnalysis.getModelPreProcessor().getProgramUnits
().size()); //$NON-NLS-1$
		else
			analysisMonitor.setBeginTask(
					Messages.getString("AnalyzeModelJob.6"), globalAnalysis.getModelPreProcessor().getProgramUnits
().size()); //$NON-NLS-1$

		Iterator it = model.getProjects().values().iterator();
		while (it.hasNext()) {
			IProject project = (IProject) it.next();
			if (createMarkers)
				createMarkers(globalAnalysis, project);
			Iterator it1 = project.getNamespaces().values().iterator();
			while (it1.hasNext()) {
				INamespace namespace = (INamespace) it1.next();
				if (createMarkers)
					createMarkers(globalAnalysis, namespace);
				Iterator it2 = namespace.getProgramUnits().values().iterator();
				while (it2.hasNext()) {
					IProgramUnit programUnit = (IProgramUnit) it2.next();
					analysisMonitor.processingResource(programUnit.getName(), 1);
					if (analysisMonitor.isCanceled())
						throw new InterruptedException("canceled"); //$NON-NLS-1$
					Iterator it3 = null;
					String resourceLocation = null;
					it3 = programUnit.getMethods().iterator();
					resourceLocation = programUnit.getAbsoluteLocation().getName();
					if (createMarkers)
						createMarkers(globalAnalysis, programUnit, resourceLocation);

					while (it3.hasNext()) {
						IMethod method = (IMethod) it3.next();
						if (createMarkers)
							createMarkers(globalAnalysis, method, resourceLocation);
					}
				}
			}
		}
		return globalAnalysis;
	}

	private IMarker createMarker(IResource resource, StatusMessage statusMessage, Object origin) {
		try {
			if (resource.exists()) {
				IMarker marker = resource.createMarker(LACHESIS_MARKER_ID);
				int markerSeverity = IMarker.SEVERITY_INFO;
				if (statusMessage.isError())
					markerSeverity = IMarker.SEVERITY_ERROR;
				else if (statusMessage.isFatal())
					markerSeverity = IMarker.SEVERITY_ERROR;
				else if (statusMessage.isWarning())
					markerSeverity = IMarker.SEVERITY_WARNING;

				marker.setAttribute(IMarker.MESSAGE, statusMessage.getMessage()
						+ Messages.getString("AnalyzeModelJob.7") + ((INamed) origin).getName()); //
$NON-NLS-1$
				marker.setAttribute(IMarker.SEVERITY, markerSeverity);
				if (origin instanceof ILinesOfCode) {
					marker.setAttribute(IMarker.LINE_NUMBER, ((ILinesOfCode) origin).getFirstLineNumber());
				}
				return marker;
			}
		} catch (CoreException e) {
			e.printStackTrace();
		}
		return null;
	}

	private void createMarkers(GlobalAnalysis globalAnalysis, Object element) {
		createMarkers(globalAnalysis, element, null);
```

```
	}

	private void createMarkers(GlobalAnalysis globalAnalysis, Object element, String resourcePath) {

		try {
			IResource resourceLocal = ResourceUtils.getResource(resourcePath);
			if (resourceLocal == null)
				return;

			for (Iterator iter = globalAnalysis.getAnalysisRecord(element).getAnalysisPackages().values().iterator(); iter.hasNext();) {

				IAnalysisPackage analysisPackage = (IAnalysisPackage) iter.next();
				Status status = analysisPackage.getStatus();
				for (int i = 0; i < status.getMessages().size(); i++) {
					StatusMessage statusMessage = (StatusMessage) status.getMessages().get(i);
					createMarker(resourceLocal, statusMessage, element);
				}
			}
		} catch (Throwable t) {
			t.printStackTrace();
		}

	}

	private int countStatusMessagesForElement(GlobalAnalysis globalAnalysis, Object element) {
		int result = 0;

		try {
			if ((globalAnalysis != null) && (globalAnalysis.getAnalysisRecord(element) != null)
					&& (globalAnalysis.getAnalysisRecord(element).getAnalysisPackages() != null))
				for (Iterator iter = globalAnalysis.getAnalysisRecord(element).getAnalysisPackages().values().iterator(); iter.hasNext
();) {
					IAnalysisPackage analysisPackage = (IAnalysisPackage) iter.next();
					if (analysisPackage != null)
						result += analysisPackage.getStatus().getMessages().size();
				}
		} catch (Throwable t) {
			t.printStackTrace();
		}
		return result;
	}

	private int countAllStatusMessages(GlobalAnalysis globalAnalysis, IModel model) {
		int result = 0;
		Iterator it = model.getProjects().values().iterator();
		while (it.hasNext()) {
			IProject project = (IProject) it.next();

			if (!GlobalAnalysis.isIncludedInAnalysis(project))
				continue;

			result += countStatusMessagesForElement(globalAnalysis, project);
			Iterator it1 = project.getNamespaces().values().iterator();
			while (it1.hasNext()) {
				INamespace namespace = (INamespace) it1.next();
				result += countStatusMessagesForElement(globalAnalysis, namespace);
				Iterator it2 = namespace.getProgramUnits().values().iterator();
				while (it2.hasNext()) {
					IProgramUnit programUnit = (IProgramUnit) it2.next();
					Iterator it3 = null;
					String resourceLocation = null;
					it3 = programUnit.getMethods().iterator();
					resourceLocation = programUnit.getAbsoluteLocation().getName();
					result += countStatusMessagesForElement(globalAnalysis, programUnit);

					while (it3.hasNext()) {
						IMethod method = (IMethod) it3.next();
						result += countStatusMessagesForElement(globalAnalysis, method);
					}
				}
			}
		}
		return result;
	}

}


package org.bog.lachesis.eclipse.plugin.jobs;

import java.io.File;

import org.apache.log4j.Logger;
import org.bog.lachesis.analysis.IAnalysisMonitor;
import org.bog.lachesis.eclipse.plugin.LachesisPlugin;
import org.bog.lachesis.eclipse.plugin.Messages;
import org.bog.lachesis.eclipse.plugin.views.AnalysisMonitor;
import org.bog.lachesis.model.readers.JavaModelReader;
import org.bog.lachesis.profiler.Profiler;
import org.bog.lachesis.reports.ModelReport;
```

```
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.core.runtime.Status;

public class ParseModelJob extends ALachesisWorkspaceJob {
		private static Logger log = Logger.getLogger(ParseModelJob.class);

		private final static String PROFILED_TASK = "Read Model Job"; //$NON-NLS-1$
		public ParseModelJob(String name, AnalysisRequest analysisRequest) {
			super(name, analysisRequest);
		}

		public IStatus runInWorkspace(IProgressMonitor monitor) throws CoreException {
			final IProgressMonitor progressMonitor = monitor;
			Profiler.startProfile(PROFILED_TASK);
			try {
				JavaModelReader javaPackageModelReader = new JavaModelReader();
				IAnalysisMonitor analysisMonitor = new AnalysisMonitor(progressMonitor);
				getAnalysisRequest().setModel(
						javaPackageModelReader.readModel(getAnalysisRequest().getModelSource(), analysisMonitor));
			} catch (InterruptedException e1) {
				Profiler.stopProfile(PROFILED_TASK);
				return Status.CANCEL_STATUS;
			} catch (Exception e) {
				Profiler.stopProfile(PROFILED_TASK);
				log.error("Java Model Read failed",e); //$NON-NLS-1$
				e.printStackTrace();
				LachesisPlugin.showErrorMessage(Messages.getString("ParseModelJob.1")); //$NON-NLS-1$
				return Status.CANCEL_STATUS;
			}
			File reportModelFile = (File) getAnalysisRequest().getReport();
			monitor.setTaskName(Messages.getString("ParseModelJob.2")); //$NON-NLS-1$
			Profiler.stopProfile(PROFILED_TASK);
			if (reportModelFile != null) {
				ModelReport.getInstance().printModelToFile(getAnalysisRequest().getModel(), reportModelFile);
				// setFinishedSuccessfully(true);
				ALachesisWorkspaceJob runJob = new AnalyzeModelJob(Messages.getString("ParseModelJob.3"), getAnalysisRequest()); //$NON-
NLS-1$

				runJob.setUser(true);
				runJob.schedule();
			}
			Profiler.logStatus();
			return Status.OK_STATUS;
		}


}
```

Пакет org.bog.lachesis.eclipse.plugin.metamodel

```
package org.bog.lachesis.eclipse.plugin.metamodel;

import org.bog.lachesis.metamodel.INamed;

public class AbstractNamed {
		String name;

		public AbstractNamed(INamed named) {
			setName(named.getName());
		}

		public String getName() {
			return name;
		}

		public void setName(String name) {
			this.name = name;
		}
}

package org.bog.lachesis.eclipse.plugin.metamodel;

import org.bog.lachesis.metamodel.ILinesOfCode;
import org.bog.lachesis.metamodel.INamed;

public class LinesOfCode extends AbstractNamed implements ILinesOfCode {

		private int firstLineNumber;

		private int lastLineNumber;

		public LinesOfCode(ILinesOfCode linesOfCode) {
			super((INamed)linesOfCode);
			firstLineNumber = linesOfCode.getFirstLineNumber();
			lastLineNumber = linesOfCode.getLastLineNumber();
		}
```

```
		public int getFirstLineNumber() {
			return firstLineNumber;
		}

		public int getLastLineNumber() {
			return lastLineNumber;
		}

		public void setFirstLineNumber(int firstLineNumber) {
			this.firstLineNumber = firstLineNumber;
		}

		public void setLastLineNumber(int lastLineNumber) {
			this.lastLineNumber = lastLineNumber;
		}

}

package org.bog.lachesis.eclipse.plugin.metamodel;

import org.bog.lachesis.metamodel.IMethod;


public class Method extends LinesOfCode {
		private ProgramUnit enclosingProgramUnit;


		public Method(IMethod method) {
			super(method);
		}

		public ProgramUnit getEnclosingProgramUnit() {
			return enclosingProgramUnit;
		}

		public void setEnclosingProgramUnit(ProgramUnit enclosingProgramUnit) {
			this.enclosingProgramUnit = enclosingProgramUnit;
		}

}


package org.bog.lachesis.eclipse.plugin.metamodel;

import java.util.ArrayList;
import java.util.List;

import org.bog.lachesis.metamodel.IModel;

public class Model extends AbstractNamed {
		public Model(IModel model) {
			super(model);
		}

		List projects = new ArrayList();

		public List getProjects() {
			return projects;
		}

		public void addProject(Project project) {
			projects.add(project);
		}


}

package org.bog.lachesis.eclipse.plugin.metamodel;

import java.util.ArrayList;
import java.util.List;

import org.bog.lachesis.metamodel.INamespace;

public class Namespace extends AbstractNamed {

		private Project enclosingProject;

		public Namespace(INamespace namespace) {
			super(namespace);
		}

		private List programUnits = new ArrayList();


		public List getProgramUnits() {
			return programUnits;
		}

		public void addProgramUnit(ProgramUnit programUnit) {
```

```
                programUnits.add(programUnit);
                programUnit.setEnclosingNamespace(this);
        }

        public Project getEnclosingProject() {
                return enclosingProject;
        }

        public void setEnclosingProject(Project enclosingProject) {
                this.enclosingProject = enclosingProject;
        }


}


package org.bog.lachesis.eclipse.plugin.metamodel;

import java.util.ArrayList;
import java.util.List;

import org.bog.lachesis.metamodel.ILocation;
import org.bog.lachesis.metamodel.IProgramUnit;
import org.bog.lachesis.metamodel.IResourceLocateable;

public class ProgramUnit extends LinesOfCode implements IResourceLocateable {
        private Namespace enclosingNamespace;
        private ILocation location;

        public ProgramUnit(IProgramUnit programUnit) {
                super(programUnit);
                setAbsoluteLocation(programUnit.getAbsoluteLocation());
        }

        private List methods = new ArrayList();


        public List getMethods() {
                return methods;
        }

        public void addMethod(Method method) {
                methods.add(method);
                method.setEnclosingProgramUnit(this);
        }

        public Namespace getEnclosingNamespace() {
                return enclosingNamespace;
        }

        public void setEnclosingNamespace(Namespace enclosingNamespace) {
                this.enclosingNamespace = enclosingNamespace;
        }

        public ILocation getAbsoluteLocation() {
                return location;
        }

        public void setAbsoluteLocation(ILocation location) {
                this.location = location;
        }

}


package org.bog.lachesis.eclipse.plugin.metamodel;

import java.util.ArrayList;
import java.util.List;

import org.bog.lachesis.metamodel.IProject;

public class Project extends AbstractNamed  {

        private boolean includedInReport;

        public Project(IProject project) {
                super(project);
                setIncludedInReport(project.isIncludedInReport());
        }

        List namespaces = new ArrayList();

        public List getNamespaces() {
                return namespaces;
        }

        public void addNamespace(Namespace namespace) {
                namespaces.add(namespace);
                namespace.setEnclosingProject(this);
```

```
		}
	public boolean isIncludedInReport() {
		return includedInReport;
	}
	public void setIncludedInReport(boolean includedInReport) {
		this.includedInReport = includedInReport;
	}

}
```

Пакет org.bog.lachesis.eclipse.plugin.popup.actions

```
package org.bog.lachesis.eclipse.plugin.popup.actions;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

import org.apache.log4j.Logger;
import org.bog.lachesis.eclipse.plugin.LachesisPlugin;
import org.bog.lachesis.eclipse.plugin.Messages;
import org.bog.lachesis.eclipse.plugin.jobs.AnalysisRequest;
import org.bog.lachesis.eclipse.plugin.jobs.ParseModelJob;
import org.bog.lachesis.eclipse.plugin.preferences.LachesisPreferences;
import org.bog.lachesis.eclipse.plugin.utils.ResourceUtils;
import org.bog.lachesis.model.IModelSource;
import org.bog.lachesis.model.sources.FileSystemModelSource;
import org.bog.lachesis.profiler.Profiler;
import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IFolder;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.resources.IWorkspace;
import org.eclipse.core.resources.WorkspaceJob;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.core.runtime.Status;
import org.eclipse.jdt.core.IClasspathEntry;
import org.eclipse.jdt.core.JavaCore;
import org.eclipse.jdt.internal.core.JavaProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.IActionDelegate;
import org.eclipse.ui.IObjectActionDelegate;
import org.eclipse.ui.IWorkbenchPart;
import org.eclipse.ui.dialogs.ResourceSelectionDialog;

public class ActionNewAnalysis implements IObjectActionDelegate {

		private static final Logger log = Logger.getLogger(ActionNewAnalysis.class);

		private IStructuredSelection selection;

		private AnalysisRequest analysisRequest = new AnalysisRequest();

		private List initialAnalysisResourceSelection;

		private List initialSupplementResourceSelection;

		/**
		 * Constructor for Action1.
		 */
		public ActionNewAnalysis() {
			super();
		}

		/**
		 * @see IObjectActionDelegate#setActivePart(IAction, IWorkbenchPart)
		 */
		public void setActivePart(IAction action, IWorkbenchPart targetPart) {

		}

		private boolean openProject(final org.eclipse.core.resources.IProject project) {
			WorkspaceJob job = new WorkspaceJob(Messages.getString("ActionNewAnalysis.0")) { //$NON-NLS-1$

				public IStatus runInWorkspace(IProgressMonitor monitor) throws CoreException {
					try {
						project.open(monitor);
					} catch (CoreException e) {
						e.printStackTrace();
						return Status.OK_STATUS;
```

```
                                                }
                                                return Status.OK_STATUS;
                                        }
                                };
                                job.setUser(true);
                                job.schedule();
                                while (job.getResult() == null)
                                        ;

                                return project.isOpen();
                }

                public void includeRecursivelyResource(File source, List resources) {
                                recurseInDirFrom(source.getAbsolutePath(), resources);
                }

                private void recurseInDirFrom(String dirItem, List resources) {
                                File file = new File(dirItem);
                                String list[];
                                if (file.isDirectory()) {
                                                list = file.list();
                                                for (int i = 0; i < list.length; i++)
                                                                recurseInDirFrom(dirItem + File.separatorChar + list[i], resources);
                                } else
                                                resources.add(file);

                }

                private Object[] convertToResources(List files) {
                                List resources = new ArrayList();
                                for (int i = 0; i < files.size(); i++) {
                                                IResource resource = ResourceUtils.getResource((File) files.get(i));
                                                if (resource != null)
                                                                resources.add(resource);
                                }
                                return resources.toArray();

                }

                private Object[] convertToFiles(Object[] resources) {
                                List files = new ArrayList();
                                for (int i = 0; i < resources.length; i++) {
                                                IPath path = ((IResource) resources[i]).getRawLocation();
                                                if (path != null)
                                                                files.add(new File(path.toOSString()));

                                }
                                return files.toArray();
                }

                private void proposeAnalysisResources(IAdaptable analysisTarget /*, IProgressMonitor progressMonitor*/) {
                                IResource resource = null;
//                              progressMonitor.beginTask(Messages.getString("ActionNewAnalysis.1"), IProgressMonitor.UNKNOWN); //$NON-NLS-1$
                                try {
                                                String resourceFileName = null;
                                                String resourceTitle = null;
                                                File reportFile = null;
                                                initialAnalysisResourceSelection = new ArrayList();
                                                initialSupplementResourceSelection = new ArrayList();

                                                if (analysisTarget instanceof IFolder) {
                                                                resource = (IResource) analysisTarget;
                                                                resourceTitle = resource.getName();
                                                                resourceFileName = ((IFolder) resource).getLocation().toOSString();
                                                                includeRecursivelyResource(new File(resourceFileName), initialAnalysisResourceSelection);
                                                } else if (analysisTarget instanceof IFile) {
                                                                resource = (IResource) analysisTarget;
                                                                resourceTitle = resource.getName();
                                                                resourceFileName = ((IFile) resource).getLocation().toOSString();
                                                                includeRecursivelyResource(new File(resourceFileName), initialAnalysisResourceSelection);
                                                } else if (analysisTarget instanceof JavaProject) {
                                                                JavaProject javaProject = (JavaProject) analysisTarget;
                                                                org.eclipse.core.resources.IProject project = javaProject.getProject();

                                                                resource = (IResource) project;
                                                                resourceTitle = project.getName();

                                                                if (!project.isOpen())
                                                                                if (!openProject(project)) {
                                                                                                LachesisPlugin.showInfoMessage(Messages.getString("ActionNewAnalysis.2")); //
$NON-NLS-1$
                                                                                                return;
                                                                                }
                                                                resourceFileName = project.getLocation().toOSString();

                                                                IClasspathEntry[] classpathEntries = javaProject.getExpandedClasspath(true);
                                                                for (int i = 0; i < classpathEntries.length; i++) {
                                                                                IClasspathEntry entry = JavaCore.getResolvedClasspathEntry(classpathEntries[i]);
                                                                                IPath path = entry.getPath();
                                                                                IWorkspace workspace = project.getWorkspace();
```

```
                                                IResource javaResource = null;

                                                if (entry.getEntryKind() == IClasspathEntry.CPE_PROJECT) {
                                                        log.error("ClassPathEntry PROJECT is not supported yet. (" + path + ")"); //$NON-
NLS-1$ //$NON-NLS-2$
                                                        continue;
                                                }
                                                try {
                                                        javaResource = workspace.getRoot().getFile(path);
                                                } catch (Exception e) {
                                                        log.error("CPE INVALID: " + path); //$NON-NLS-1$
                                                        e.printStackTrace();
                                                }
//                                              progressMonitor.subTask(resourceFileName);
//                                              progressMonitor.worked(1);
                                                switch (entry.getEntryKind()) {
                                                case IClasspathEntry.CPE_SOURCE:
                                                        if (javaResource != null) {
                                                                log.debug("CPE SOURCE: adding source folder as AnalysisResource:
" + javaResource.getLocation()); //$NON-NLS-1$
                                                                includeRecursivelyResource(new File(javaResource.getLocation().
toOSString()), initialAnalysisResourceSelection);
                                                        } else
                                                                log.error("CPE SOURCE: source folder could not be located : " +
path.toOSString()); //$NON-NLS-1$

                                                        break;
                                                case IClasspathEntry.CPE_CONTAINER:
                                                        log.debug("CPE CONTAINER"); //$NON-NLS-1$
                                                        break;
                                                case IClasspathEntry.CPE_LIBRARY:
                                                        if ((path.getFileExtension() != null) && (path.getFileExtension().toLowerCase().equals
("jar"))) { //$NON-NLS-1$
                                                                if (entry.isExported()) {
                                                                        log.debug("CPE EXPORTED LIBRARY: adding JAR
as AnalysisResource: " //$NON-NLS-1$
                                                                                        +
javaResource.getRawLocation().toOSString());
                                                                        includeRecursivelyResource(new File
(javaResource.getRawLocation().toOSString()),
initialAnalysisResourceSelection);
                                                                } else {
                                                                        log.debug("CPE LIBRARY: adding JAR as
SupplementResource: " + path.toOSString()); //$NON-NLS-1$
                                                                        includeRecursivelyResource(new File(path.toOSString
()), initialSupplementResourceSelection);
                                                                }
                                                        }
                                                        break;
                                                }
                                        }
                                } else if (analysisTarget instanceof org.eclipse.core.resources.IProject) {
                                        resource = (IResource) analysisTarget;
                                        org.eclipse.core.resources.IProject project = (org.eclipse.core.resources.IProject) resource;
                                        resourceTitle = project.getName();
                                        if (!project.isOpen())
                                                if (!openProject(project)) {
                                                        LachesisPlugin.showInfoMessage(Messages.getString("ActionNewAnalysis.12")); //
$NON-NLS-1$
                                                        return;
                                                }
                                        resourceFileName = project.getLocation().toOSString();
                                        includeRecursivelyResource(new File(resourceFileName), initialAnalysisResourceSelection);
                                }
                                try {
                                        reportFile = new File(resourceFileName + ".lachesisReport.xml"); //$NON-NLS-1$
                                        reportFile.createNewFile();
                                } catch (Exception e1) {
                                        reportFile = null;
                                        e1.printStackTrace();
                                }

                                analysisRequest.setReport(reportFile);
                                analysisRequest.setName(reportFile.getName());
                                analysisRequest.setAnalysisTargetRoot(resource);

                        } catch (OutOfMemoryError e) {
                                LachesisPlugin
                                                .showInfoMessage(Messages.getString("ActionNewAnalysis.14") //$NON-NLS-1$
                                                                + Runtime.getRuntime().freeMemory()
                                                                + Messages.getString("ActionNewAnalysis.15") + Runtime.getRuntime
().maxMemory() / 1048576 + Messages.getString("ActionNewAnalysis.16")); //$NON-NLS-1$ //$NON-NLS-2$

                                e.printStackTrace();
                                return;
                        } catch (Throwable e) {
                                e.printStackTrace();
                        }
                }
```

```
	private void processAnalysisResources(Object[] analysisResources, Object[] supplementResources) {
		try {
			log.debug("Java Model Reader is started ..."); //$NON-NLS-1$
			IModelSource modelSource = new FileSystemModelSource();

			modelSource.setName(analysisRequest.getName());

			for (int i = 0; i < analysisResources.length; i++)
				if (analysisResources[i] != null)
					modelSource.getAnalysisResources().add((File) analysisResources[i]);
			for (int i = 0; i < supplementResources.length; i++)
				if (supplementResources[i] != null)
					modelSource.getSupplementResources().add((File) supplementResources[i]);

			log.debug("Adding JAVA_HOME to ModelSource");
			modelSource.includeSupplementResource(new File(System.getProperty("java.home") + "/lib"));

			analysisRequest.setModelSource(modelSource);
		} catch (OutOfMemoryError e) {
			LachesisPlugin
					.showInfoMessage(Messages.getString("ActionNewAnalysis.14") //$NON-NLS-1$
							+ Runtime.getRuntime().freeMemory()
							+ Messages.getString("ActionNewAnalysis.15") + Runtime.getRuntime
().maxMemory() / 1048576 + Messages.getString("ActionNewAnalysis.16")); //$NON-NLS-1$ //$NON-NLS-2$

			e.printStackTrace();
			return;
		} catch (Throwable e) {
			e.printStackTrace();
		}
	}

	/**
	 * @see IActionDelegate#run(IAction)
	 */
	public void run(IAction action) {
		try {
			Profiler.logStatus();
			final IAdaptable adaptable = (IAdaptable) selection.getFirstElement();

			WorkspaceJob workspaceJob =  new WorkspaceJob("") {

				public IStatus runInWorkspace(IProgressMonitor monitor) throws CoreException {
					final IProgressMonitor innerMonitor = monitor;
					monitor.beginTask(Messages.getString("CollectResources"), 2); //$NON-NLS-1$
					proposeAnalysisResources(adaptable);

					Display.getDefault().syncExec(new Runnable() {

						public void run() {
							innerMonitor.subTask(Messages.getString
("SelectAnalysisResources")); //$NON-NLS-1$
							ResourceSelectionDialog analysisResourcesDialog = new
ResourceSelectionDialog(new Shell(),
									analysisRequest.getAnalysisTargetRoot
(), Messages.getString("ActionNewAnalysis.17")); //$NON-NLS-1$

							analysisResourcesDialog.setMessage(Messages.getString
("ActionNewAnalysis.18")); //$NON-NLS-1$
							analysisResourcesDialog.setTitle(Messages.getString
("ActionNewAnalysis.19")); //$NON-NLS-1$
							analysisResourcesDialog.setInitialSelections(convertToResources
(initialAnalysisResourceSelection));
							analysisResourcesDialog.setBlockOnOpen(true);

							if (analysisResourcesDialog.open() == Dialog.OK) {
								innerMonitor.worked(1);

								innerMonitor.subTask(Messages.getString
("SelectSupplementResources")); //$NON-NLS-1$

								ResourceSelectionDialog supplementResourcesDialog
= new ResourceSelectionDialog(new Shell(),
analysisRequest.getAnalysisTargetRoot(), Messages.getString("ActionNewAnalysis.17b")); //$NON-NLS-1$
								supplementResourcesDialog.setMessage
(Messages.getString("ActionNewAnalysis.18b")); //$NON-NLS-1$
								supplementResourcesDialog.setTitle
(Messages.getString("ActionNewAnalysis.19b")); //$NON-NLS-1$
								supplementResourcesDialog.setInitialSelections
(convertToResources(initialSupplementResourceSelection));
								supplementResourcesDialog.setBlockOnOpen(true);

								if (supplementResourcesDialog.open() == Dialog.OK) {
									innerMonitor.done();
									Object[] confirmedAnalysisResources =
analysisResourcesDialog.getResult();
									Object[]
confirmedSupplementResources = supplementResourcesDialog.getResult();
```

```
processAnalysisResources
(convertToFiles(confirmedAnalysisResources),
convertToFiles(confirmedSupplementResources));
analysisRequest.setCreateMarkers
(LachesisPreferences.getCreateMarkers());
ParseModelJob job = new
ParseModelJob(Messages.getString("ActionNewAnalysis.20"), analysisRequest); //$NON-NLS-1$
job.setUser(true);
job.schedule();
}
}

}

});
return Status.OK_STATUS;
}

};
workspaceJob.setName(Messages.getString("SelectResources")); //$NON-NLS-1$
workspaceJob.setUser(true);
workspaceJob.schedule();

Profiler.logStatus();

} catch (RuntimeException e) {
e.printStackTrace();
LachesisPlugin.showInfoMessage(Messages.getString("ActionNewAnalysis.21")); //$NON-NLS-1$
}
}

/**
 * @see IActionDelegate#selectionChanged(IAction, ISelection)
 */
public void selectionChanged(IAction action, ISelection selection) {
this.selection = (IStructuredSelection) selection;
}

}
```

Пакет org.bog.lachesis.eclipse.plugin.preferences

```
package org.bog.lachesis.eclipse.plugin.preferences;

import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.tools.AnalysisRecord;
import org.bog.lachesis.eclipse.plugin.LachesisPlugin;
import org.bog.lachesis.eclipse.plugin.Messages;
import org.eclipse.jface.preference.BooleanFieldEditor;
import org.eclipse.jface.preference.ColorFieldEditor;
import org.eclipse.jface.preference.FieldEditorPreferencePage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.TabFolder;
import org.eclipse.swt.widgets.TabItem;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPreferencePage;

/**
 * This class represents a preference page that is contributed to the
 * Preferences dialog. By subclassing <samp>FieldEditorPreferencePage</samp>,
 * we can use the field support built into JFace that allows us to create a page
 * that is small and knows how to save, restore and apply itself.
 * <p>
 * This page is used to modify preferences only. They are stored in the
 * preference store that belongs to the main plug-in class. That way,
 * preferences can be accessed directly via the preference store.
 */

public class AnalysisPackagesPage extends FieldEditorPreferencePage implements IWorkbenchPreferencePage {

	public AnalysisPackagesPage() {
		super(GRID);
		setPreferenceStore(LachesisPlugin.getDefault().getPreferenceStore());
		setDescription(Messages.getString("AnalysisPackagesPage.0")); //$NON-NLS-1$
	}

	/**
	 * Creates the field editors. Field editors are abstractions of the common
	 * GUI blocks needed to manipulate various types of preferences. Each field
	 * editor knows how to save and restore itself.
```

```
		 */
		public void createFieldEditors() {

				TabFolder folder = new TabFolder(getFieldEditorParent(), SWT.NONE);
				GridData gd = new GridData(GridData.FILL_BOTH);
				gd.horizontalSpan = 3;
				folder.setLayoutData(gd);

				List analysisPackageDescriptors = AnalysisRecord.getAnalysisPackageDescriptors();
				for (Iterator iter = analysisPackageDescriptors.iterator(); iter.hasNext();) {
					AnalysisPackageDescriptor packageDescriptor = (AnalysisPackageDescriptor) iter.next();
			createAnalysisPackagePreferencesPage(packageDescriptor,folder);
				}


//				fPluginsBlock.initialize();


		}

		private Control createAnalysisPackagePreferencesPage(AnalysisPackageDescriptor packageDescriptor, TabFolder folder) {
				Composite container = new Composite(folder, SWT.NONE);
				GridLayout layout = new GridLayout();
				layout.numColumns = 2;
				//layout.marginHeight = 0;
				//layout.marginWidth = 0;
				container.setLayout(layout);

				//container.setLayoutData(new GridData(GridData.FILL_BOTH));

				List analysisDescriptors = packageDescriptor.getAnalysisDescriptors();
				for (Iterator iterator = analysisDescriptors.iterator(); iterator.hasNext();) {
					AnalysisDescriptor descriptor = (AnalysisDescriptor) iterator.next();
					Composite descriptorContainer = new Composite(container, SWT.LEFT);
					Composite descriptorColorContainer = new Composite(container, SWT.RIGHT);
					GridLayout descriptorLayout = new GridLayout();
					layout.marginHeight = 5;
					layout.marginWidth = 5;
					descriptorContainer.setLayout(descriptorLayout);
					descriptorColorContainer.setLayout(descriptorLayout);

					addField(new BooleanFieldEditor(LachesisPreferences.getDescriptorPreferenceKey(descriptor), descriptor.getName(),
descriptorContainer));
					addField(new ColorFieldEditor(LachesisPreferences.getDescriptorColorPreferenceKey(descriptor) , "", descriptorColorContainer)); //
$NON-NLS-1$
				}

				TabItem tab = new TabItem(folder, SWT.NONE);
				tab.setText(packageDescriptor.getName()); //$NON-NLS-1$
				tab.setControl(container);

				return container;
		}
		/*
		 * (non-Javadoc)
		 *
		 * @see org.eclipse.ui.IWorkbenchPreferencePage#init(org.eclipse.ui.IWorkbench)
		 */
		public void init(IWorkbench workbench) {
		}

}

package org.bog.lachesis.eclipse.plugin.preferences;

import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.tools.AnalysisRecord;
import org.bog.lachesis.eclipse.plugin.LachesisPlugin;
import org.eclipse.jface.preference.PreferenceConverter;
import org.eclipse.swt.graphics.RGB;

public class LachesisPreferences {
		public static int getCreateMarkers() {
				return LachesisPlugin.getDefault().getPluginPreferences().getInt(PreferenceConstants.P_CHOICE_CREATE_MARKERS);
		}

		public static boolean getAnalysisResult(AnalysisDescriptor analysisDescriptor) {
				return LachesisPlugin.getDefault().getPluginPreferences().getBoolean(getDescriptorPreferenceKey(analysisDescriptor));
		}

		public static void composeListOfEnabledAnalysisResults(List list) {
				list.clear();
				List analysisPackageDescriptors = AnalysisRecord.getAnalysisPackageDescriptors();
				for (Iterator iter = analysisPackageDescriptors.listIterator(); iter.hasNext();) {
					AnalysisPackageDescriptor packageDescriptor = (AnalysisPackageDescriptor) iter.next();
					List analysisDescriptors = packageDescriptor.getAnalysisDescriptors();
```

```
                                        for (Iterator iterator = analysisDescriptors.listIterator(); iterator.hasNext();) {
                                                AnalysisDescriptor descriptor = (AnalysisDescriptor) iterator.next();
                                                if (LachesisPreferences.getAnalysisResult(descriptor))
                                                        list.add(descriptor);
                                        }
                                }
                }

                public static RGB getAnalysisDescriptorColumnColor(AnalysisDescriptor analysisDescriptor) {
                        return PreferenceConverter.getColor(LachesisPlugin.getDefault().getPreferenceStore(),
                                        getDescriptorColorPreferenceKey(analysisDescriptor));
                }

                public static String getDescriptorPreferenceKey(AnalysisDescriptor analysisDescriptor) {
                        return analysisDescriptor.getAnalysisPackageDescriptor().getName() + "." + analysisDescriptor.getName(); //$NON-NLS-1$
                }

                public static String getDescriptorColorPreferenceKey(AnalysisDescriptor analysisDescriptor) {
                        return getDescriptorPreferenceKey(analysisDescriptor) + ".column_background"; //$NON-NLS-1$
                }
}

package org.bog.lachesis.eclipse.plugin.preferences;

import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.tools.AnalysisRecord;
import org.bog.lachesis.eclipse.plugin.LachesisPlugin;
import org.eclipse.core.runtime.preferences.AbstractPreferenceInitializer;
import org.eclipse.core.runtime.preferences.DefaultScope;
import org.eclipse.core.runtime.preferences.IEclipsePreferences;
import org.eclipse.jface.preference.IPreferenceStore;
import org.eclipse.jface.preference.PreferenceConverter;
import org.eclipse.swt.graphics.RGB;

public class LachesisPreferencesInitializer extends AbstractPreferenceInitializer {
                public LachesisPreferencesInitializer() {
                        super();
                }

                public void initializeDefaultPreferences() {
                        IEclipsePreferences prefs = new DefaultScope().getNode(LachesisPlugin.ID);
                        prefs.putInt(PreferenceConstants.P_CHOICE_CREATE_MARKERS, PreferenceConstants.V_CHOICE_ASK);
                        prefs.putInt(PreferenceConstants.P_CHOICE_SAVE_MODEL, PreferenceConstants.V_CHOICE_ASK);
                        prefs.putInt(PreferenceConstants.P_CHOICE_SAVE_ANALYSIS, PreferenceConstants.V_CHOICE_ASK);

                        IPreferenceStore store = LachesisPlugin.getDefault().getPreferenceStore();
                        List analysisPackageDescriptors = AnalysisRecord.getAnalysisPackageDescriptors();
                        for (Iterator iter = analysisPackageDescriptors.listIterator(); iter.hasNext();) {
                                AnalysisPackageDescriptor packageDescriptor = (AnalysisPackageDescriptor) iter.next();
                                List analysisDescriptors = packageDescriptor.getAnalysisDescriptors();
                                for (Iterator iterator = analysisDescriptors.listIterator(); iterator.hasNext();) {
                                        AnalysisDescriptor descriptor = (AnalysisDescriptor) iterator.next();
                                        PreferenceConverter.setDefault(store, LachesisPreferences.getDescriptorColorPreferenceKey(descriptor),
                                                        new RGB(255, 255, 255));
                                }
                        }

                }

}


package org.bog.lachesis.eclipse.plugin.preferences;

import org.bog.lachesis.eclipse.plugin.LachesisPlugin;
import org.bog.lachesis.eclipse.plugin.Messages;
import org.eclipse.jface.preference.FieldEditorPreferencePage;
import org.eclipse.jface.preference.RadioGroupFieldEditor;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPreferencePage;

/**
 * This class represents a preference page that is contributed to the
 * Preferences dialog. By subclassing <samp>FieldEditorPreferencePage</samp>,
 * we can use the field support built into JFace that allows us to create a page
 * that is small and knows how to save, restore and apply itself.
 * <p>
 * This page is used to modify preferences only. They are stored in the
 * preference store that belongs to the main plug-in class. That way,
 * preferences can be accessed directly via the preference store.
 */

public class MainPage extends FieldEditorPreferencePage implements IWorkbenchPreferencePage {

                public MainPage() {
                        super(GRID);
```

```
                setPreferenceStore(LachesisPlugin.getDefault().getPreferenceStore());
                setDescription(Messages.getString("MainPage.0")); //$NON-NLS-1$
        }

        /**
         * Creates the field editors. Field editors are abstractions of the common
         * GUI blocks needed to manipulate various types of preferences. Each field
         * editor knows how to save and restore itself.
         */
        public void createFieldEditors() {
                addField(new RadioGroupFieldEditor(PreferenceConstants.P_CHOICE_CREATE_MARKERS, Messages.getString("MainPage.1"), 1, //$NON-
NLS-1$
                                new String[][] { { Messages.getString("MainPage.2"), String.valueOf
(PreferenceConstants.V_CHOICE_ENABLED) }, //$NON-NLS-1$
                                                { Messages.getString("MainPage.3"), String.valueOf
(PreferenceConstants.V_CHOICE_DISABLED) }, //$NON-NLS-1$
                                                { Messages.getString("MainPage.4"), String.valueOf
(PreferenceConstants.V_CHOICE_ASK) } }, getFieldEditorParent(), true)); //$NON-NLS-1$

                addField(new RadioGroupFieldEditor(PreferenceConstants.P_CHOICE_SAVE_MODEL, Messages.getString("MainPage.5"), 1, //$NON-NLS-1$
                                new String[][] { { Messages.getString("MainPage.6"), String.valueOf
(PreferenceConstants.V_CHOICE_ENABLED) }, //$NON-NLS-1$
                                                { Messages.getString("MainPage.7"), String.valueOf
(PreferenceConstants.V_CHOICE_DISABLED) }, //$NON-NLS-1$
                                                { Messages.getString("MainPage.8"), String.valueOf
(PreferenceConstants.V_CHOICE_ASK) } }, getFieldEditorParent(), true)); //$NON-NLS-1$

                addField(new RadioGroupFieldEditor(PreferenceConstants.P_CHOICE_SAVE_MODEL, Messages.getString("MainPage.9"), 1, //$NON-NLS-1$
                                new String[][] { { Messages.getString("MainPage.6"), String.valueOf
(PreferenceConstants.V_CHOICE_ENABLED) }, //$NON-NLS-1$
                                                { Messages.getString("MainPage.7"), String.valueOf
(PreferenceConstants.V_CHOICE_DISABLED) }, //$NON-NLS-1$
                                                { Messages.getString("MainPage.8"), String.valueOf
(PreferenceConstants.V_CHOICE_ASK) } }, getFieldEditorParent(), true)); //$NON-NLS-1$

        }

        /*
         * (non-Javadoc)
         *
         * @see org.eclipse.ui.IWorkbenchPreferencePage#init(org.eclipse.ui.IWorkbench)
         */
        public void init(IWorkbench workbench) {
        }

}

package org.bog.lachesis.eclipse.plugin.preferences;


/**
 * Constant definitions for plug-in preferences
 */
public class PreferenceConstants {

        public static final String P_PATH = "pathPreference"; //$NON-NLS-1$

        public static final String P_CHOICE_CREATE_MARKERS = "choiceCreateMarkersPreference"; //$NON-NLS-1$

        public static final String P_CHOICE_SAVE_MODEL = "choiceSaveModelPreference"; //$NON-NLS-1$
        public static final int V_CHOICE_ENABLED = 1;
        public static final int V_CHOICE_DISABLED = 2;
        public static final int V_CHOICE_ASK = 3;

        public static final String P_CHOICE_SAVE_ANALYSIS = "choiceSaveAnalysisPreference"; //$NON-NLS-1$

        public static final String P_CHOICE = "choicePreference"; //$NON-NLS-1$

        public static final String P_STRING = "stringPreference"; //$NON-NLS-1$

}


Пакет org.bog.lachesis.eclipse.plugin.utils

package org.bog.lachesis.eclipse.plugin.utils;

import java.io.File;

import org.apache.log4j.Logger;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.resources.IWorkspaceRoot;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.runtime.Path;

public class ResourceUtils {
        private static final Logger log = Logger.getLogger(ResourceUtils.class);

        public static IResource getResource(File file) {
```

```
                return getResource(file.getAbsolutePath());
        }
        public static IResource getResource(String resourceLocation) {
                IResource result = null;
                try {
                        if (resourceLocation != null) {
                                IWorkspaceRoot workspaceRoot = ResourcesPlugin.getWorkspace().getRoot();
                                IPath path = new Path(resourceLocation);
                                result = workspaceRoot.getFileForLocation(path);
                                if (result == null)
                                        result = workspaceRoot.getFile(path);
                                if (result == null)
                                        result = workspaceRoot.findMember(path);
                                if (result == null)
                                        result = workspaceRoot.findFilesForLocation(path)[0];
                                if (result != null)
                                        return result;
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
                log.error("Cannot map file: " + resourceLocation); //$NON-NLS-1$
                return null;
        }

}
```

Пакет org.bog.lachesis.eclipse.plugin.views

```
package org.bog.lachesis.eclipse.plugin.views;

import org.bog.lachesis.analysis.AbstractAnalysisMonitor;
import org.eclipse.core.runtime.IProgressMonitor;


public class AnalysisMonitor extends AbstractAnalysisMonitor {

        private IProgressMonitor progressMonitor;

        private String subTaskName;

        public AnalysisMonitor(IProgressMonitor progressMonitor) {
                this.progressMonitor = progressMonitor;
        }

        /*
         *  /* (non-Javadoc)
         *
         * @see org.bog.lachesis.model.readers.IParseMonitor#processingPart(int)
         */
        public void processingPart(int worked) {
                progressMonitor.worked(worked);
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.model.readers.IParseMonitor#setBeginTask(java.lang.String,
         *      int)
         */
        protected void setBeginTaskImpl(String taskName, int count) {
                progressMonitor.beginTask(taskName, count);
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.AbstractAnalysisMonitor#setTask(java.lang.String)
         */
        protected void setTaskImpl(String taskName) {
                progressMonitor.setTaskName(taskName);
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.AbstractAnalysisMonitor#setSubTask(java.lang.String)
         */
        public void setSubTask(String taskName) {
                subTaskName = taskName;
                progressMonitor.subTask(taskName);
        }

        /*
         * (non-Javadoc)
         *
         * @see org.bog.lachesis.AbstractAnalysisMonitor#isCanceled()
         */
```

```
		public boolean isCanceled() {
			return progressMonitor.isCanceled();
		}

		public void setSubTaskAddition(String addition) {
			progressMonitor.subTask(subTaskName+addition);
		}

}


package org.bog.lachesis.eclipse.plugin.views;

import java.util.Iterator;
import java.util.List;

import org.bog.lachesis.analysis.AnalysisDescriptor;
import org.bog.lachesis.analysis.AnalysisPackageDescriptor;
import org.bog.lachesis.analysis.IAnalysisPackage;
import org.bog.lachesis.analysis.impl.DependenciesAnalysis;
import org.bog.lachesis.analysis.tools.AnalysisRecord;
import org.bog.lachesis.eclipse.plugin.Messages;
import org.eclipse.ui.views.properties.IPropertyDescriptor;
import org.eclipse.ui.views.properties.IPropertySource;
import org.eclipse.ui.views.properties.PropertyDescriptor;
import org.eclipse.ui.views.properties.TextPropertyDescriptor;

public class AnalysisPackagePropertySource implements IPropertySource {

		AnalysisRecord analysisRecord;

		List enabledAnalysisResults;

		public AnalysisPackagePropertySource(AnalysisRecord analysisRecord, List enabledAnalysisResults) {
			this.analysisRecord = analysisRecord;
			this.enabledAnalysisResults = enabledAnalysisResults;
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see org.eclipse.ui.views.properties.IPropertySource#getEditableValue()
		 */
		public Object getEditableValue() {
			return null;
		}

		private class PropertyIdentifier {
			private AnalysisDescriptor analysisDescriptor;

			private Object item;

			/**
			 * @return Returns the analysisDescriptor.
			 */
			public AnalysisDescriptor getAnalysisDescriptor() {
				return analysisDescriptor;
			}

			/**
			 * @param analysisDescriptor
			 *            The analysisDescriptor to set.
			 */
			public void setAnalysisDescriptor(AnalysisDescriptor analysisDescriptor) {
				this.analysisDescriptor = analysisDescriptor;
			}

			/**
			 * @return Returns the item.
			 */
			public Object getItem() {
				return item;
			}

			/**
			 * @param item
			 *            The item to set.
			 */
			public void setItem(Object item) {
				this.item = item;
			}
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see org.eclipse.ui.views.properties.IPropertySource#getPropertyDescriptors()
		 */
		public IPropertyDescriptor[] getPropertyDescriptors() {
			// COUNT ALL ANALYSIS RESULTS FOR ALL ANALYSIS BUNDLES
```

```
                int length = 0;

                for (Iterator iter = enabledAnalysisResults.iterator(); iter.hasNext();) {
                        AnalysisDescriptor descriptor = (AnalysisDescriptor) iter.next();
                        AnalysisPackageDescriptor packageDescriptor = descriptor.getAnalysisPackageDescriptor();

                        IAnalysisPackage analysisPackage = analysisRecord.getAnalysisPackage(packageDescriptor);
                        if (analysisPackage != null) {
                                Object analysisResult = analysisPackage.getAnalysisResult(descriptor);
                                if (analysisResult != null) {
                                        if (packageDescriptor == DependenciesAnalysis.PACKAGE_DESCRIPTOR) {
                                                List list = (List) analysisResult;
                                                length += list.size();
                                        } else
                                                length++;
                                }
                        }
                }

                // CREATE PropertyDescriptor BASED ON THE COUNT CALCULATED
                PropertyDescriptor[] result = new PropertyDescriptor[length];
                int offset = 0;

                for (Iterator iter = enabledAnalysisResults.iterator(); iter.hasNext();) {
                        AnalysisDescriptor descriptor = (AnalysisDescriptor) iter.next();
                        AnalysisPackageDescriptor packageDescriptor = descriptor.getAnalysisPackageDescriptor();

                        IAnalysisPackage analysisPackage = analysisRecord.getAnalysisPackage(packageDescriptor);
                        if (analysisPackage != null) {
                                Object analysisResult = analysisPackage.getAnalysisResult(descriptor);
                                if (analysisResult != null) {
                                        if (packageDescriptor == DependenciesAnalysis.PACKAGE_DESCRIPTOR) {
                                                List list = (List) analysisResult;
                                                for (Iterator iterator = list.iterator(); iterator.hasNext();) {
                                                        Object item = iterator.next();
                                                        PropertyIdentifier propertyIdentifier = new PropertyIdentifier();
                                                        propertyIdentifier.setAnalysisDescriptor(descriptor);
                                                        propertyIdentifier.setItem(item);
                                                        result[offset] = new PropertyDescriptor(propertyIdentifier, (String)
(((DependenciesAnalysis.Dependency) item)
                                                                        .isResolved() ? Messages.getString
("AnalysisPackagePropertySource.0") : Messages.getString("AnalysisPackagePropertySource.1"))); //$NON-NLS-1$ //$NON-NLS-2$
                                                        result[offset].setCategory(descriptor.getName());
                                                        offset++;
                                                }
                                        } else {
                                                result[offset] = new TextPropertyDescriptor(descriptor, descriptor.getName());
                                                result[offset].setCategory(packageDescriptor.getName());
                                                offset++;
                                        }
                                }
                        }
                }

                return result;
        }

        public String convertToString(Object param) {
                if (param != null)
                        return param.toString();
                return null;
        }

        /*
         * (non-Javadoc)
         *
         * @see org.eclipse.ui.views.properties.IPropertySource#getPropertyValue(java.lang.Object)
         */
        public Object getPropertyValue(Object id) {
                Object value = null;
                if (id instanceof PropertyIdentifier) {
                        AnalysisDescriptor descriptor = ((PropertyIdentifier) id).getAnalysisDescriptor();

                        value = analysisRecord.getAnalysisPackage(descriptor.getAnalysisPackageDescriptor()).getAnalysisResult(descriptor);
                        if (value != null)
                                value = ((DependenciesAnalysis.Dependency) ((PropertyIdentifier) id).getItem()).getName();
                } else {
                        value = analysisRecord.getAnalysisPackage(((AnalysisDescriptor) id).getAnalysisPackageDescriptor()).getAnalysisResult(
                                        ((AnalysisDescriptor) id));
                }
                return value;
        }

        /*
         * (non-Javadoc)
         *
         * @see org.eclipse.ui.views.properties.IPropertySource#isPropertySet(java.lang.Object)
         */
        public boolean isPropertySet(Object id) {
                return false;
```

```
		}

		/*
		 * (non-Javadoc)
		 *
		 * @see org.eclipse.ui.views.properties.IPropertySource#resetPropertyValue(java.lang.Object)
		 */
		public void resetPropertyValue(Object id) {

		}

		/*
		 * (non-Javadoc)
		 *
		 * @see org.eclipse.ui.views.properties.IPropertySource#setPropertyValue(java.lang.Object,
		 *      java.lang.Object)
		 */
		public void setPropertyValue(Object id, Object value) {

		}

}

package org.bog.lachesis.eclipse.plugin.views;

import org.eclipse.jface.operation.IRunnableWithProgress;


public abstract class ARunnableWithProgressWithStatus implements
				IRunnableWithProgress {
		private boolean finishedSuccessfully = false;

		/**
		 * @return Returns the finishedSuccessfully.
		 */
		public boolean isFinishedSuccessfully() {
				return finishedSuccessfully;
		}

		/**
		 * @param finishedSuccessfully
		 *            The finishedSuccessfully to set.
		 */
		public void setFinishedSuccessfully(boolean finishedSuccessfully) {
				this.finishedSuccessfully = finishedSuccessfully;
		}
}

plugin.xml

<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
  id="org.bog.lachesis.eclipse.plugin"
  name="Lachesis Plug-in"
  version="1.1.3"
  provider-name="Bogdan Vatkov"
  class="org.bog.lachesis.eclipse.plugin.LachesisPlugin">

  <runtime>
    <library name="lib/lachesis.core.jar">
      <export name="*"/>
    </library>
    <library name="lachesis.ecp.jar">
      <export name="*"/>
    </library>
    <library name="lib/log4j-1.2.8.jar">
      <export name="*"/>
    </library>
    <library name="lib/asm-2.1.jar">
      <export name="*"/>
    </library>
    <library name="lib/asm-tree-2.1.jar">
      <export name="*"/>
    </library>
  </runtime>

  <requires>
    <import plugin="org.eclipse.ui"/>
    <import plugin="org.eclipse.core.runtime"/>
    <import plugin="org.eclipse.jface.text"/>
    <import plugin="org.eclipse.ui.editors"/>
    <import plugin="org.eclipse.help"/>
    <import plugin="org.eclipse.core.resources"/>
    <import plugin="org.eclipse.ui.ide"/>
    <import plugin="org.eclipse.ui.views" version="3.1.0"/>
    <import plugin="org.eclipse.jdt.ui"/>
    <import plugin="org.eclipse.ui.workbench.texteditor"/>
    <import plugin="org.eclipse.jdt.core"/>
  </requires>
```

```
<extension
      point="org.eclipse.ui.editors">
   <editor
         name="Lachesis Metrics Models"
         icon="icons/sample.gif"
         class="org.bog.lachesis.eclipse.plugin.editors.LachesisEditor"
         id="org.bog.lachesis.eclipse.plugin.editors.LachesisEditor">
   </editor>
</extension>
<extension
      point="org.eclipse.help.toc">
   <toc
         file="toc.xml">
   </toc>
   <toc
         file="testToc.xml"
         primary="true">
   </toc>
</extension>
<extension
      point="org.eclipse.ui.popupMenus">
   <objectContribution
         objectClass="org.eclipse.core.resources.IFolder"
         id="org.bog.lachesis.eclipse.plugin.contribution1">
      <menu
            label="%lachesis"
            path="additions"
            id="org.bog.lachesis.eclipse.plugin.menu1">
         <separator
               name="group1">
         </separator>
      </menu>
      <action
            label="%analyze.with.lachesis"
            class="org.bog.lachesis.eclipse.plugin.popup.actions.ActionNewAnalysis"
            menubarPath="org.bog.lachesis.eclipse.plugin.menu1/group1"
            enablesFor="1"
            id="org.bog.lachesis.eclipse.plugin.ActionNewAnalysis">
      </action>
   </objectContribution>
</extension>
<extension
      point="org.eclipse.ui.popupMenus">
   <objectContribution
         objectClass="org.eclipse.core.resources.IProject"
         id="org.bog.lachesis.eclipse.plugin.contribution1">
      <menu
            label="%lachesis"
            path="additions"
            id="org.bog.lachesis.eclipse.plugin.menu1">
         <separator
               name="group1">
         </separator>
      </menu>
      <action
            label="%analyze.with.lachesis"
            class="org.bog.lachesis.eclipse.plugin.popup.actions.ActionNewAnalysis"
            menubarPath="org.bog.lachesis.eclipse.plugin.menu1/group1"
            enablesFor="1"
            id="org.bog.lachesis.eclipse.plugin.ActionNewAnalysis">
      </action>
   </objectContribution>
</extension>
<extension
      point="org.eclipse.ui.popupMenus">
   <objectContribution
         objectClass="org.eclipse.jdt.core.IJavaProject"
         id="org.bog.lachesis.eclipse.plugin.contribution1">
      <menu
            label="%lachesis"
            path="additions"
            id="org.bog.lachesis.eclipse.plugin.menu1">
         <separator
               name="group1">
         </separator>
      </menu>
      <action
            label="%analyze.with.lachesis"
            class="org.bog.lachesis.eclipse.plugin.popup.actions.ActionNewAnalysis"
            menubarPath="org.bog.lachesis.eclipse.plugin.menu1/group1"
            enablesFor="1"
            id="org.bog.lachesis.eclipse.plugin.ActionNewAnalysis">
      </action>
   </objectContribution>
</extension>
<extension
      point="org.eclipse.ui.popupMenus">
   <objectContribution
         objectClass="org.eclipse.core.resources.IFile"
```

```
            nameFilter="*.*"
            id="org.bog.lachesis.eclipse.plugin.contribution1">
         <menu
               label="%lachesis"
               path="additions"
               id="org.bog.lachesis.eclipse.plugin.menu1">
            <separator
                  name="group1">
            </separator>
         </menu>
         <action
               label="%lachesis.analyze"
               class="org.bog.lachesis.eclipse.plugin.popup.actions.ActionNewAnalysis"
               menubarPath="org.bog.lachesis.eclipse.plugin.menu1/group1"
               enablesFor="1"
               id="org.bog.lachesis.eclipse.plugin.ActionNewAnalysis">
         </action>
      </objectContribution>
   </extension>
   <extension point="org.eclipse.core.runtime.preferences">
      <initializer
         class="org.bog.lachesis.eclipse.plugin.preferences.LachesisPreferencesInitializer"/>
   </extension>
   <extension
         point="org.eclipse.ui.preferencePages">
      <page
            name="%preferences"
            class="org.bog.lachesis.eclipse.plugin.preferences.MainPage"
            id="org.bog.lachesis.eclipse.plugin.preferences.MainPage">
      </page>
      <page
            name="%preferences.analysis.packages"
            category="org.bog.lachesis.eclipse.plugin.preferences.MainPage"
            class="org.bog.lachesis.eclipse.plugin.preferences.AnalysisPackagesPage"
            id="org.bog.lachesis.eclipse.plugin.preferences.AnalysisPackagesPage">
      </page>
   </extension>
   <extension
         point="org.eclipse.ui.propertyPages">
      <page
            objectClass="org.eclipse.core.resources.IFile"
            name="Sample Page"
            nameFilter="*.*"
            class="org.bog.lachesis.eclipse.plugin.properties.SamplePropertyPage"
            id="org.bog.lachesis.eclipse.plugin.properties.samplePropertyPage">
      </page>
   </extension>
   <extension
         point="org.eclipse.ui.views">
      <view
            name="Lachesis Metrics"
            icon="icons/sample.gif"
            category="TestPlugin"
            class="org.bog.lachesis.eclipse.plugin.views.LachesisView"
            id="org.bog.lachesis.eclipse.plugin.views.LachesisView">
      </view>
   </extension>
   <extension
         id="problemmarker"
         name="%problemName"
         point="org.eclipse.core.resources.markers">
      <super
            type="org.eclipse.core.resources.problemmarker">
      </super>
      <attribute
            name="severity">
      </attribute>
      <attribute
            name="message">
      </attribute>
      <attribute
            name="location">
      </attribute>
   </extension>

</plugin>
```

Пакет log4j.properties

```
# ------------------------------------------------------------------
#
# L O G G I N G
#
# ------------------------------------------------------------------
# We use Log4J for all the logging and we embed the log4j
# properties within our application configuration.
# ------------------------------------------------------------------


log4j.appender.stdout=org.apache.log4j.FileAppender
log4j.appender.stdout.File=./lachesis.log
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %m %100c{1}:%L %n
#log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

log4j.appender.errout=org.apache.log4j.ConsoleAppender
log4j.appender.errout.Target=System.err
log4j.appender.errout.layout=org.apache.log4j.PatternLayout
log4j.appender.errout.layout.ConversionPattern=%d{ABSOLUTE} %5p %m %100c{1}:%L %n

log4j.appender.MainDebugFile=org.apache.log4j.FileAppender
log4j.appender.MainDebugFile.File=logs/main.log.xml
log4j.appender.MainDebugFile.layout=org.apache.log4j.PatternLayout
log4j.appender.MainDebugFile.layout.ConversionPattern=%m%n


### set log levels - for more verbose logging change 'warn' or 'info' to 'debug' ###

## ROOT
log4j.rootLogger=error, stdout

## MAIN
log4j.logger.org.bog.lachesis.tools.Lachesis=error, stdout
log4j.additivity.org.bog.lachesis.tools.Lachesis=false


#log4j.logger.org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst=error,ParsingLogFile
#log4j.additivity.org.bog.lachesis.recognizers.sourcecode.java.sablecc.cst=false

#log4j.logger.org.bog.lachesis.recognizers.binarycode.javabytecode.ByteCodeModelExtractor = error, stdout
#log4j.additivity.org.bog.lachesis.recognizers.binarycode.javabytecode.ByteCodeModelExtractor = false

log4j.logger.org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor = all, stdout
log4j.additivity.org.bog.lachesis.recognizers.sourcecode.java.sablecc.ModelPreProcessor =false

log4j.logger.org.bog.lachesis.model.readers=error, stdout
log4j.additivity.org.bog.lachesis.model.readers=false

log4j.logger.org.bog.lachesis.eclipse.plugin.LachesisModelPool=error, stdout
log4j.additivity.org.bog.lachesis.eclipse.plugin.LachesisModelPool=false

log4j.logger.org.bog.lachesis.profiler.Profiler=debug, stdout
log4j.additivity.org.bog.lachesis.profiler.Profiler=false


## BINARY RECOGNIZER
log4j.logger.org.bog.lachesis.recognizers.binarycode=error, stdout
log4j.additivity.org.bog.lachesis.recognizers.binarycode=false
```