COPE User Guide

A guide on using the Component Adaptation Environment (COPE)

Table of Contents

ABBREVIATIONS **6**

1. COPE USER GUIDE **7**

1.1 INTRODUCTION 7

1.1.1 COPE: Purpose & Scope *7*

1.1.2 Before you start using COPE *7*

1.2 INSTALLING COPE 8

1.2.1 Minimum System Requirements *8*

1.2.2 Software Prerequisites *9*

1.2.3 Installation Process *9*

1.3 USING COPE 9

1.3.1 Preparing a F/LOSS project for COPE *10*

1.3.2 Exiting COPE *10*

1.3.3 Creating a new Reuse Project *10*

1.3.4 Opening an existing Reuse Project *12*

1.3.5 Analyzing a Reuse Project with COPE *13*

1.3.5.1 Performing Static Analysis *13*

1.3.5.2 Performing Source File Indexing *16*

1.3.5.3 Performing Documentation Generation *17*

1.3.5.4 Performing Dynamic Analysis *18*

1.3.5.5 Performing History Analysis *34*

1.3.5.6 Performing Pattern Analysis *34*

1.3.6 Cluster Recommendation *35*

1.3.6.1 Running the Dependencies Recommender *36*

1.3.6.2 Running the Pattern Recommender *37*

1.3.7 Component Making *38*

1.3.7.1 Running the Dependencies Maker *39*

1.4 COPE'S COMPONENTS STRUCTURE 41

1.4.1 Root Directory (Component's Name) *43*

1.4.2 Readme.txt 43
1.4.3 Source Directory (src/)..43
1.4.4 Libraries Directory (lib/)..43
1.4.5 Documentation Directory (doc/) 44
1.4.6 Test Directory (test/) 44
1.5 THE KNOWLEDGE MANAGER..44
1.5.1 Characterizing a Generated Component 46
1.5.2 Adding / Removing / Renaming Languages and Technologies 47
1.5.3 Classifying Components...49
1.5.3.1 Adding a new Concept 50
1.5.3.2 Adding a new Domain 50
1.5.3.3 Assigning Concepts to Domains 50
1.5.3.4 Creating the Metamodel 53
1.5.3.5 Classifying the Component 58
1.5.3.6 Selecting Tiers.. 60
1.5.4 Synchronizing with Compare Repository 60
1.6 KNOWN ISSUES & WORKAROUNDS...62
1.6.1 Documentation Generation 62
1.6.2 Dynamic Analysis 62
1.6.2.1 Coverage... 62
1.6.2.2 Validation 63
1.6.3 Component Makers...63
2. REFERENCES..64

List of Figures

Figure 1 - COPE’s main window 10
Figure 2 – New Reuse Project… dialog....11
Figure 3- Reuse Project successfully created....12
Figure 4- Open Reuse Project… dialog 13
Figure 5- COPE performing Static Analysis....15
Figure 6- COPE’s main window after a successful static analysis 15
Figure 7- COPE performing Source File Indexing....16
Figure 8- Searching for classes using COPE’s internal search engine 17
Figure 9-Documentation for the class Account of the JMoney Reuse Project 18
Figure 10- COPE’s Dynamic Analysis: Component Selection Tab....19
Figure 11- Selecting an execution scenario....21
Figure 12- Dynamic Analysis: Coverage Tab 21
Figure 13 – Coverage results....22
Figure 14 – Coverage information for Account’s setName method 23
Figure 15 – Control Flow Graph for Account’s setName method....23
Figure 16 – Validation: Class tab....24
Figure 17 - Validation: Interfaces tab 24
Figure 18 - Validation: Scope tab 25
Figure 19 - Validation: Tests tab....25
Figure 20 - Validation: AspectJ tab....26
Figure 21 – Validation process successfully completed 27
Figure 22 – ProM MXML file tab....27
Figure 23 – Dynamic Analysis: State Mappings....28
Figure 24 – Dynamic Analysis: Variable Mappings....29
Figure 25 – Dynamic Analysis: Visualize FSM tab....30
Figure 26 – Guidelines to use ModelJUnit as an external tool 31

Figure 27 – Dynamic Analysis: ModelJUnit generated code....................31
Figure 28 – Customized Dynamic Analysis HTML report....................32
Figure 29 – Dynamic Analysis HTML Report: Component Description....................33
Figure 30 - Dynamic Analysis HTML Report: Coverage Report (generic)....................33
Figure 31 - Dynamic Analysis HTML Report: Coverage Report (method view pt. 1)....................34
Figure 32 - Dynamic Analysis HTML Report: Coverage Report (method view pt.2)....................34
Figure 33- COPE's main window after Pattern Analysis....................35
Figure 34 – Introductory dialog to Dependencies Recommender....................36
Figure 35 – Dependencies Recommender main dialog....................37
Figure 36 – Introductory dialog to Pattern Recommender....................38
Figure 37 – Pattern Recommender Main Dialog....................38
Figure 38 – Extracting component from classes selected in the main window of COPE....................39
Figure 39 – Dependency Maker main dialog....................40
Figure 40 – Dependency Maker dialog after a component was successfully generated....................41
Figure 41: COPE generated components' packaging....................42
Figure 42: Project Folder....................43
Figure 43 – Knowledge Manager: Main dialog....................45
Figure 44 – Characterizing a component....................47
Figure 45 – Managing Languages and Technologies dialog....................48
Figure 46 – J2ME technology added....................48
Figure 47 – J2ME technology available at the Components Tab....................49
Figure 48 – The main dialog of the Component Classification Console....................49
Figure 49 – Assigning concept to domain....................51
Figure 50 – Select concept dialog....................51
Figure 51 – Concept successfully assigned to a domain....................52
Figure 52 – Domain successfully assigned to concept....................53
Figure 53- Component MetaModels Dialog....................54
Figure 54 – MetaModel group successfully created....................55
Figure 55 – MetaModel successfully created....................56
Figure 56 – Domain selection window....................57
Figure 57 – Metamodel successfully created....................57
Figure 58 – Classifying a component....................58

Figure 59 – MetaModel selection....59

Figure 60 – Component classification successfully finished....59

Figure 61 – Select tier dialog....60

Figure 62 – Synchronization process between COPE and COMPARE....61

Figure 63 – Successful synchronization message....61

Figure 64 – Knowledge Manager’s main dialog after the successful completion of the synchronization process....62

ABBREVIATIONS

OPEN-SME	Open Source Software Reuse Service for Small / Medium Enterprises
OCEAN	Open Source Search Engine
COPE	Component Adaptation Environment
COMPARE	Component Repository and Search Engine
F/LOSS	Free/Libre Open Source Software
UML	Unified Modeling Language
CPU	Central Processing Unit
RAM	Random Access Memory
LCSAJ	Linear Code Sequence And Jump
FSM	Finite State Machine
SME	Small and Medium Enterprise
SME AG	Small and Medium Enterprise Association Group
SVN	Subversion
E-Tier	Enterprise Tier
R-Tier	Resource Tier
U-Tier	User Interface Tier
W-Tier	Workspace Tier
XML	Extensible Markup Language
ProM	Process Mining

1. COPE USER GUIDE

1.1 INTRODUCTION

This document aims in providing a complete yet simple guide in using OPEN-SME Component Adaptation Environment (COPE). Using this document the Reuse Engineer will be able to reuse Free/Libre Open Source Software (F/LOSS) Projects to produce autonomous, software components.

This guide presents in detail the aforementioned process of extracting components from existing F/LOSS projects and provides real case studies that can be used as examples for the intended user.

1.1.1 COPE: PURPOSE & SCOPE

The OPEN-SME Component Adaptation Environment (COPE) tool-chain was designed to support in a holistic manner the different activities of the Domain Engineering Process. COPE is an application with clearly defined interfaces. Different component implementations result in different instantiations of COPE’s tool chain to support specific modeling languages, programming languages, target platforms etc.

For the time being COPE is 100% compatible with Linux based operating systems and supports the Unified Model Language (UML) and Java based F/LOSS Projects.

1.1.2 BEFORE YOU START USING COPE

Every F/LOSS Project selected to be analysed and reused with COPE, is being stored in the form of a “Reuse Project”.

A “Reuse Project” serves as an extended version of a regular F/LOSS project. It combines the source code related information (of the original F/LOSS project) with those resulted from the static analysis process. A Reuse Project’s lifecycle consists of the following phases:

- **Analysis phase:** the source code of the target F/LOSS project is being analysed and the results of this analysis are being stored in its “Reuse Project”. COPE currently supports:

    1. Static Analysis: Calculates different metrics and dependencies among classes (among other things) by statically analysing the source code (i.e. without executing the program).

    2. Source File Indexing: Creates an index of the source files suitable for free-text searching.

    1. Documentation Generation: Generates the documentation of the source code (Javadoc) with the addition of UML diagrams for each class and package.

    2. Dynamic Analysis: Analyses the program using dynamic analysis (i.e. by executing the program). The reuse engineer uses dynamic analysis after component extraction to understand the functioning of an extracted component, determine the coverage of the dynamic analysis, validate the component using model-based testing techniques etc.

    3. History Analysis: We store the changes that occurred in a project's development history for Subversion repositories. Currently this feature is not used for the recommendation of components but it may be used in the future.

    4. Pattern Analysis: We use pattern detection techniques (e.g. detection of the Adapter or Proxy design patterns [1]) to pinpoint classes that participate in design patterns. Based

on pattern participation we then extract components (e.g. extract a component of a subsystem behind a proxy as a component).

- **Cluster Recommendation:** in this phase COPE automatically suggests class clusters that could possibly serve as reusable components. Currently, the following recommenders are available in COPE:

    1. Dependencies Recommender: Extracts components based on an analysis of the dependencies of the classes in a project.

    2. Pattern Recommender: Extracts components based on the detected patterns of the project.

- **Component Making:** this set of functionalities allows the user to extract components from the reuse project by either using class clusters recommended in the Cluster Recommendation phase or by selecting a single class that along with its dependencies will form a class cluster and eventually the reusable component. Currently, four different component makers are available in COPE:

    1. Interface Maker: It uses a class as a starting point and creates a component that includes all class's dependencies (recursively).

    2. Dependency Maker: It uses the clusters produced by the dependencies recommender to create components.

    3. Adapter Pattern Maker: It creates components using as starting point classes that implement the Adapter design pattern.

    4. Proxy Pattern Maker: It creates components using as starting point classes that implement the Proxy design pattern.

- **Knowledge Management:** in this phase the user provides information for the generated components. Using the "Semantic Application" feature, the user can describe the functionality of each component. Moreover s/he can classify the resulting component to a specific domain and concept.

NOTE THAT: The aforementioned phases and options form COPE's process in its full form. However it is possible for some phases and / or functionalities to be omitted.

1.2 INSTALLING COPE

This section provides detailed guidelines on installing COPE to your machine. COPE was designed to be as autonomous as possible. Therefore the prerequisites and dependencies in pre-existing software and tools have been minimized.

In general, COPE uses a MySQL database to store all the information needed to support its processes. It also uses the graphviz program to produce figures of UML diagrams. There are available packages for most Linux distributions that can be used to install these requirements.

1.2.1 MINIMUM SYSTEM REQUIREMENTS

- **CPU:** Pentium IV, 3.2GHz, Multithreading

- **RAM:** 2GB

- **Free Disk Space:** 10GB

- **Operating System:** Linux

1.2.2 SOFTWARE PREREQUISITES

- **Graphviz** (http://www.graphviz.org)[1]
- **MySQL** (http://www.mysql.com)

1.2.3 INSTALLATION PROCESS

From the provided CD:

1. Create the database using the script "CopeDatabase.sql", located in "database script" folder
   a. As MySQL root user, use the command:
   create database dependencies;
   from the mysql command prompt.
   b. Then from the OS command line issue the command:
   mysql –u root –p dependencies < CopeDatabase.sql
2. Create a database user, with these credentials:
   - **username:** copeuser
   - **password:** opensme

   From the MySQL command prompt as a root user, execute the following command: CREATE USER 'copeuser'@'localhost' IDENTIFIED BY 'opensme';
3. Give all the privileges of the database ‘dependencies’ created above, in the new user:
   a. From the MySQL command prompt as a root user, execute the following command: GRANT ALL ON dependencies.* TO 'copeuser'@'localhost';
4. Run the application from the command line with the command:
   java -jar COPESwingApp.jar

1.3 USING COPE

The main window of COPE consists of three areas of focus (see Figure 1 - COPE’s main window).

- **Area A** (marked with red): COPE’s main menu.
- **Area B** (marked with blue): Search engine intended in discovering classes within a reuse project that match specific criteria.
- **Area C** (marked with green): Shows the results of the analyses for a specific Free/Libre Open Source Software Project.

[1] In case Graphviz is not available COPE will still be able to run but all UML diagrams won’t be available.

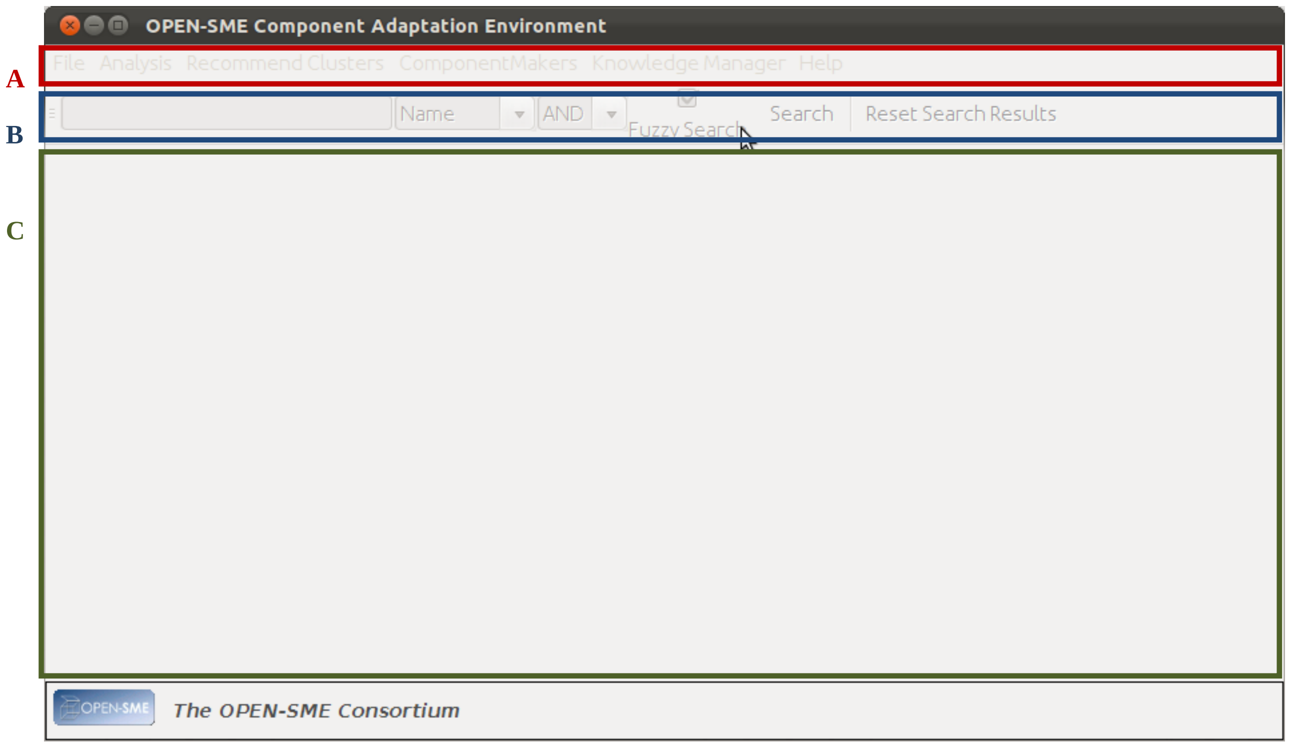

A
B
C
OPEN-SME Component Adaptation Environment
File Analysis Recommend Clusters ComponentMakers Knowledge Manager Help
Name
AND
Fuzzy Search
Search
Reset Search Results
OPEN-SME
The OPEN-SME Consortium

Figure 1 - COPE's main window

1.3.1 Preparing a F/LOSS project for COPE

In order for a Free/Libre Open Source Software Project to be able to serve as a Reuse Project for COPE the following information needs to be available:

- The binary (.jar) file of the project
- Any possible external libraries (dependencies, usually in .jar format)
- The source code of the project

CASE STUDY: For this part of our guide we will be using JMoney[2] as an example. Before we proceed in creating a new Reuse Project for JMoney in COPE we need to go through the aforementioned preparation process (see Figure 2 – New Reuse Project… dialog) to see the specific information of JMoney project.

1.3.2 Exiting COPE

To exit COPE:

1. Select **"Exit"** from the **File** menu.

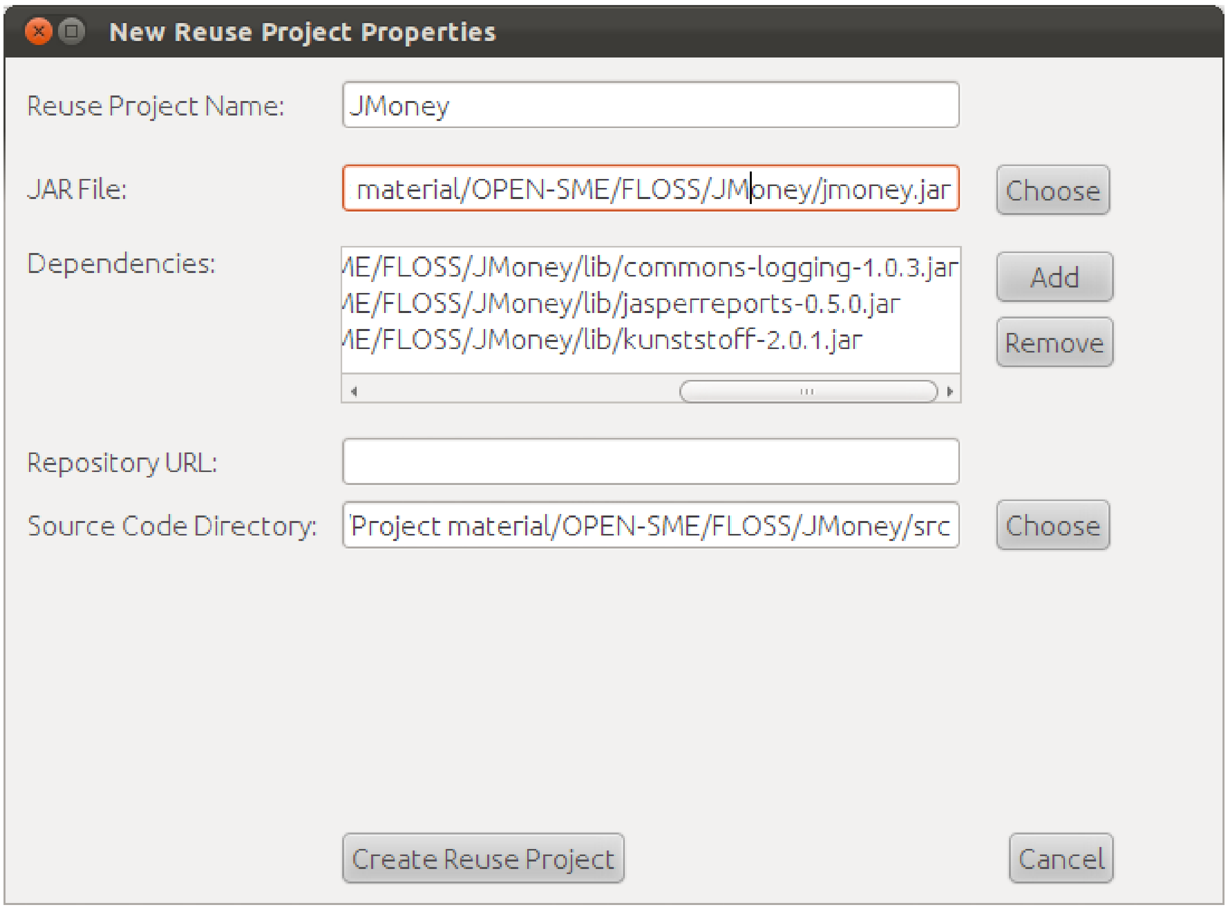
NOTE THAT: In case you were working on a Reuse Project, possible changes are automatically saved.

1.3.3 Creating a new Reuse Project

To create a new Reuse Project in COPE:

[2] http://sourceforge.net/projects/jmoney/

2. Select **“New Reuse Project…”** from the **File** menu.
3. Provide the appropriate data (see 1.3.1) to the dialog “New Reuse Project Properties” (see Figure 2 – New Reuse Project… dialog).
   - **Reuse Project Name (mandatory):** The name for the Reuse Project (free text – usually the name of the F/LOSS project to be added to COPE)
   - **JAR File (mandatory):** Path of the binary file of the F/LOSS project.
   - **Dependencies (optional):** Paths of the external libraries (.jar format) that serve as dependencies to the F/LOSS project. The reuse engineer can add or remove libraries at will using the buttons provided at the right of the form. Finally, mass addition of libraries is possible by using the CTRL+A hotkey.
   - **Repository URL (optional):** The URL to the SVN repository of the F/LOSS project.
   - **Source Code Directory (mandatory):** Path to the root folder of the source code of the F/LOSS project.


New Reuse Project Properties

Reuse Project Name: JMoney

JAR File: material/OPEN-SME/FLOSS/JMoney/jmoney.jar

Dependencies:
ME/FLOSS/JMoney/lib/commons-logging-1.0.3.jar
ME/FLOSS/JMoney/lib/jasperreports-0.5.0.jar
ME/FLOSS/JMoney/lib/kunststoff-2.0.1.jar

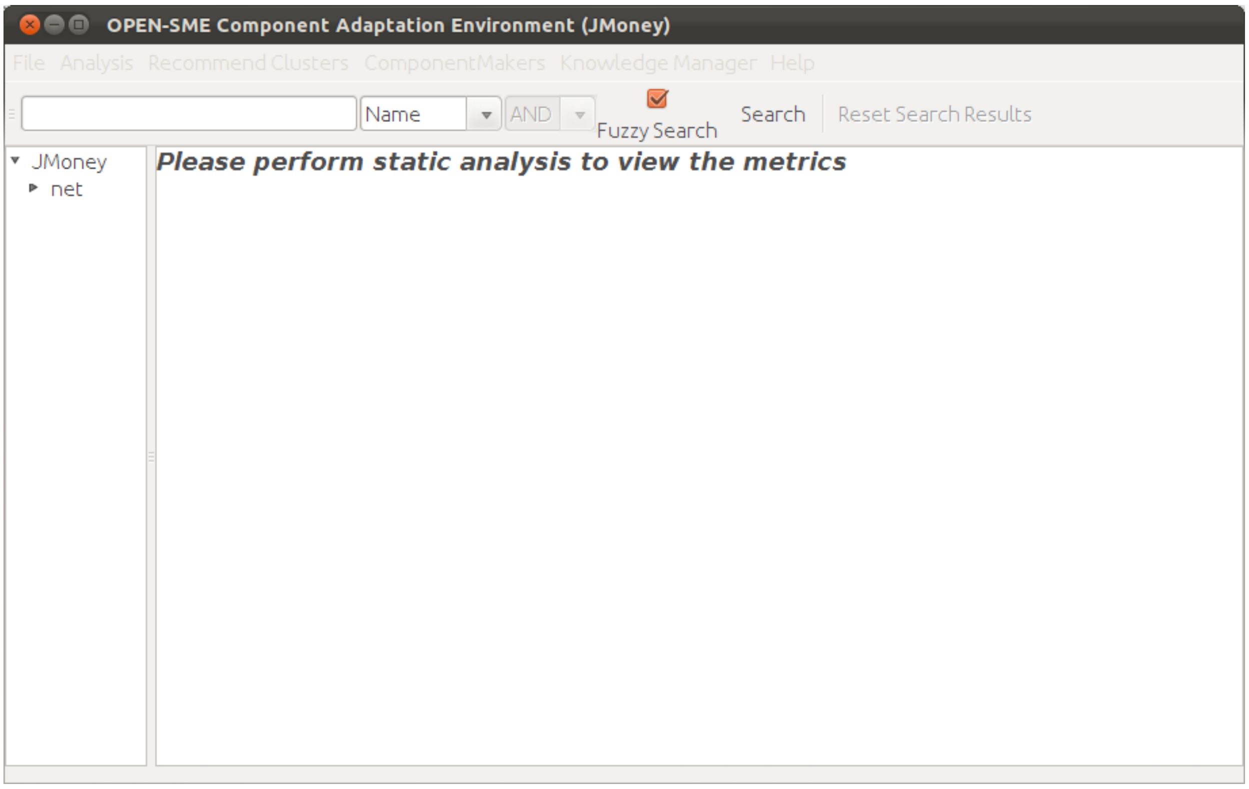
Repository URL:

Source Code Directory: Project material/OPEN-SME/FLOSS/JMoney/src

Figure 2 – New Reuse Project… dialog

NOTE THAT: The name of root folder of the source code should always be “src”.

4. Click on “Create Reuse Project”. If no errors occur, the following dialog should appear. The newly created project was successfully loaded and the given name appears to the window title (e.g. JMoney). The dialog instructs the Reuse Engineer to continue by performing Static Analysis (see 1.3.5.1).


OPEN-SME Component Adaptation Environment (JMoney)
File Analysis Recommend Clusters ComponentMakers Knowledge Manager Help
Name
AND
Fuzzy Search
Search
Reset Search Results
JMoney
net

Please perform static analysis to view the metrics

Figure 3- Reuse Project successfully created

The aforementioned process can be aborted at any step by clicking the “Cancel” button.

1.3.4 OPENING AN EXISTING REUSE PROJECT

To open an existing Reuse Project in COPE:

1. Select **“Open Reuse Project…”** from the **File** menu.
2. In the dialog that appears, select the Reuse Project.
3. Click “Open”.

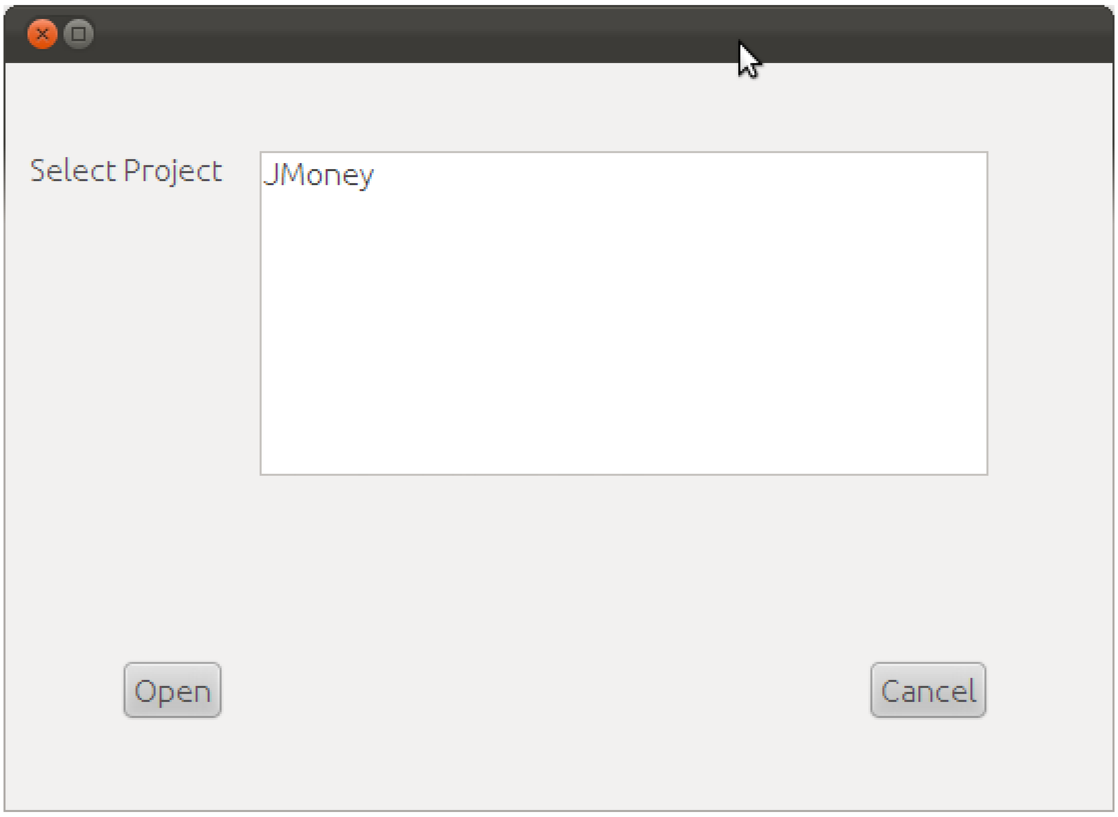


Figure 4- Open Reuse Project… dialog

If no errors occur the Reuse Project is loaded successfully in its latest known state. The aforementioned process can be aborted at any step by clicking the "Cancel" button.

CASE STUDY: If we choose to open the "JMoney" Reuse Project which we created in the previous paragraph, COPE will take us back to "Figure 3- Reuse Project successfully created".

1.3.5 ANALYZING A REUSE PROJECT WITH COPE

There are several analyzers available in COPE. These analyzers can be applied to any COPE Reuse Project and provide information such as metrics, source code documentation, pattern detection, etc. This information can assist the Reuse Engineer in the phases that follow to decide which components are promising for extraction.

1.3.5.1 PERFORMING STATIC ANALYSIS

Static Analysis results on a specific set of information for the source code of a Reuse Project. More specifically for each class of the F/LOSS Project participating in a Reuse Project in COPE, Static Analysis returns:

- **Class Name:** The fully qualified name of the specific class.
- **Type:** The type of the class (e.g. "Class", "Abstract Class", "Interface", etc.).
- **Size:** The size of the specific class (in bytes).
- **Used By:** Number of classes in the project that uses the specific class.
- **Uses (I):** Number of internal classes used by the specific class.

- **Uses (E):** Number of external classes used by the specific class.
- **Layer:** The layer of the class. The Classycle analyzer tool[3] is used internally to discover class dependencies and Directed Acyclic Graph (DAG) layers. The Classycle tool discovers strong dependencies between classes and packages, and creates a Strongly Connected Components (SCC) graph applying Tarjan's algorithm. Next, according to SCCs calls, the graph is condensated to an acyclic digraph of SCCs, from which the layers are extracted.
- **The Chidamber and Kemerer Java Metrics:** These metrics are calculated with the help of the CKJM tool[4,5]:
    - **WMC:** Weighted Methods per Class.
    - **DIT:** Depth of Inheritance Tree.
    - **NOC:** Number of Children.
    - **CBO:** Coupling between object classes.
    - **RFC:** Response for a Class.
    - **LCOM:** Lack of cohesion in methods.
    - **Ca:** Afferent couplings.
    - **NPM:** Number of Public Methods.
- **R (reusability index):** It is an estimation of the reusability of a class based on the Chidamber-Kemerer metrics. The larger this value the more reusable a class is. It can take values between 3 to -20 (approximately).
- **Pattern:** The design pattern to which a specific class is involved (if any).
- **Cluster Size:** The number of classes the specific class needs in order to form an autonomous, fully functional component. This is the cardinality of the dependencies' set of a class, which includes the class's dependencies, the dependencies of these dependencies and so on.

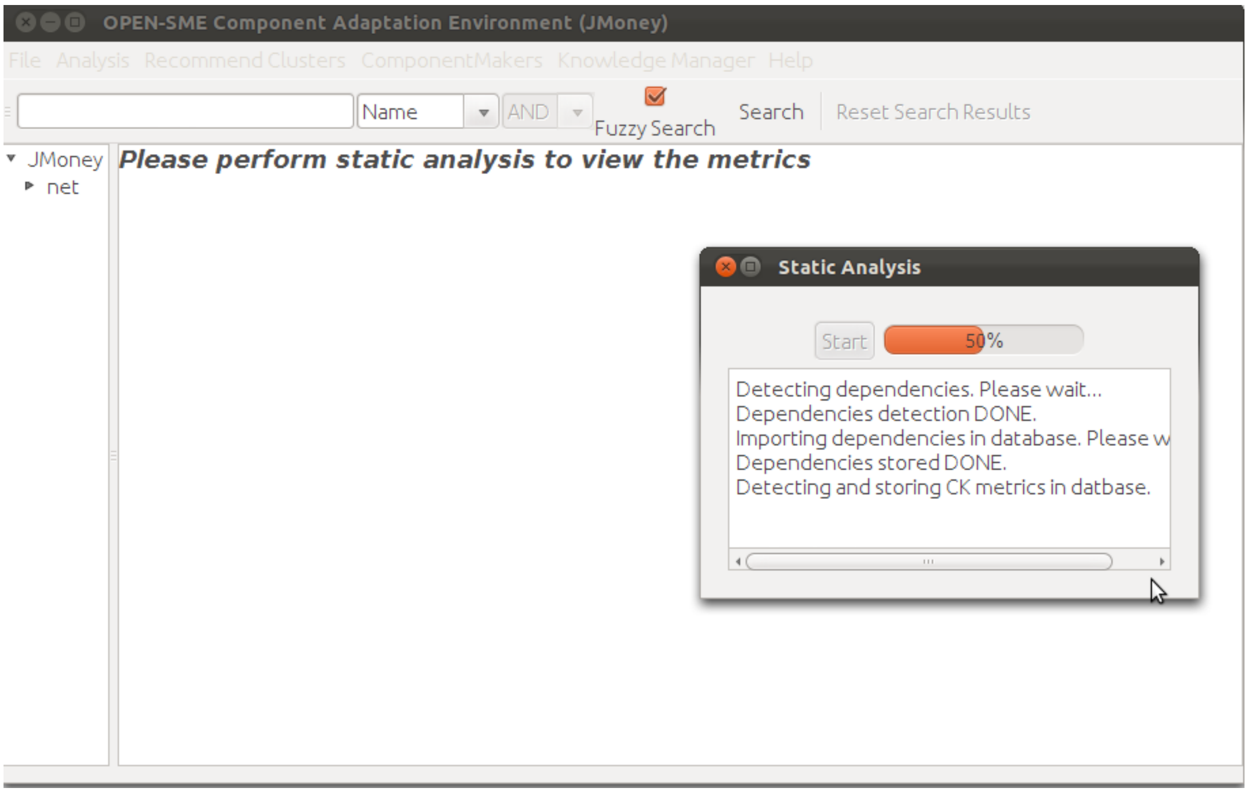
To perform static analysis for a Reuse Project in COPE:

1. Select **"Static Analysis"** from the **Analysis** menu.
2. In the dialog that appears (entitled "Static Analysis") click to the **Start** button.

[3] http://classycle.sourceforge.net/

[4] http://www.spinellis.gr/sw/ckjm/

[5] A more detailed analysis on the Chidamber and Kemerer Java Metrics can be found here


OPEN-SME Component Adaptation Environment (JMoney)
File Analysis Recommend Clusters ComponentMakers Knowledge Manager Help
Name
AND
Fuzzy Search
Search
Reset Search Results
JMoney
net
Please perform static analysis to view the metrics
Static Analysis
Start
50%

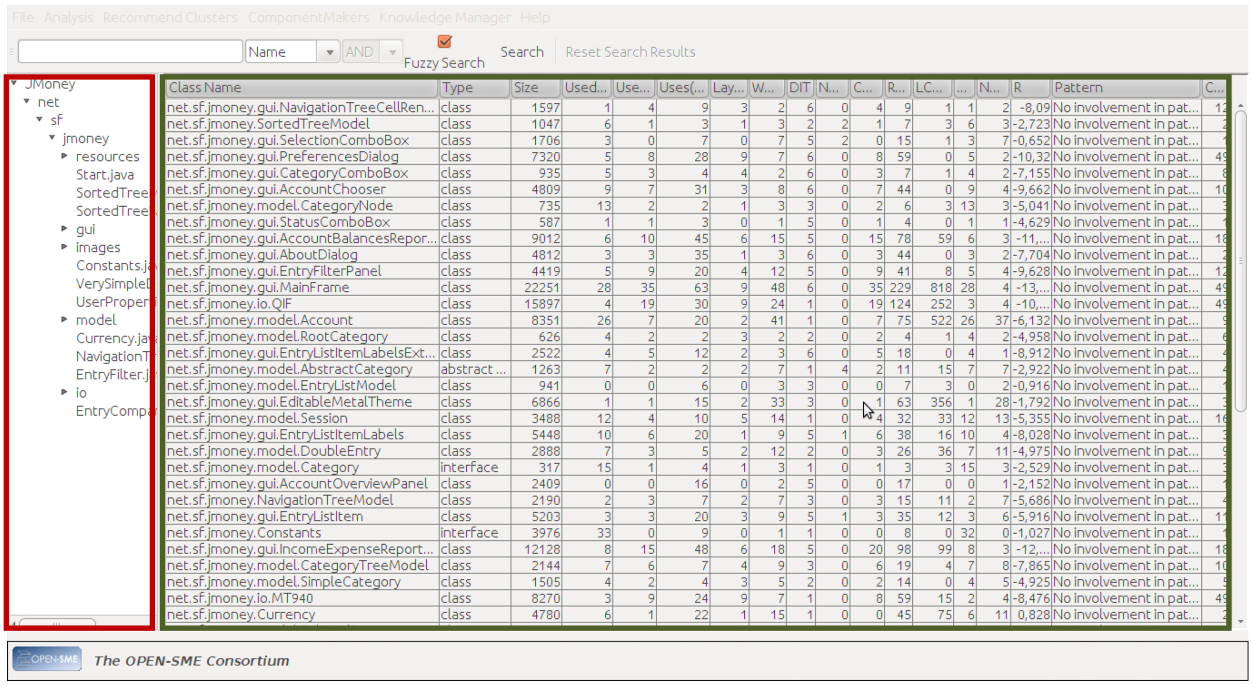
```
Detecting dependencies. Please wait...
Dependencies detection DONE.
Importing dependencies in database. Please w
Dependencies stored DONE.
Detecting and storing CK metrics in datbase.
```

Figure 5- COPE performing Static Analysis

3. When the analysis is complete (the progress bar has reached 100%) you can close the “Static Analysis” dialog using the close button of the dialog.

After the static analysis has ended, COPE’s main window contains a series of metrics and information produced by the static analysis.

<u>CASE STUDY</u>: Performing Static Analysis to the JMoney Reuse Project produces the following results



Class Name	Type	Size	Used...	Use...	Uses(...	Lay...	W...	DIT	N...	C...	R...	LC...	...	N...	R	Pattern	C...
net.sf.jmoney.gui.NavigationTreeCellRen...	class	1597	1	4	9	3	2	6	0	4	9	1	1	2	-8,09	No involvement in pat...	12
net.sf.jmoney.SortedTreeModel	class	1047	6	1	3	1	3	2	2	1	7	3	6	3	-2,723	No involvement in pat...	2
net.sf.jmoney.gui.SelectionComboBox	class	1706	3	0	7	0	7	5	2	0	15	1	3	7	-0,652	No involvement in pat...	1
net.sf.jmoney.gui.PreferencesDialog	class	7320	5	8	28	9	7	6	0	8	59	0	5	2	-10,32	No involvement in pat...	49
net.sf.jmoney.gui.CategoryComboBox	class	935	5	3	4	4	2	6	0	3	7	1	4	2	-7,155	No involvement in pat...	8
net.sf.jmoney.gui.AccountChooser	class	4809	9	7	31	3	8	6	0	7	44	0	9	4	-9,662	No involvement in pat...	10
net.sf.jmoney.model.CategoryNode	class	735	13	2	2	1	3	3	0	2	6	3	13	3	-5,041	No involvement in pat...	3
net.sf.jmoney.gui.StatusComboBox	class	587	1	1	3	0	1	5	0	1	4	0	1	1	-4,629	No involvement in pat...	1
net.sf.jmoney.gui.AccountBalancesRepor...	class	9012	6	10	45	6	15	5	0	15	78	59	6	3	-11,...	No involvement in pat...	18
net.sf.jmoney.gui.AboutDialog	class	4812	3	3	35	1	3	6	0	3	44	0	3	2	-7,704	No involvement in pat...	2
net.sf.jmoney.gui.EntryFilterPanel	class	4419	5	9	20	4	12	5	0	9	41	8	5	4	-9,628	No involvement in pat...	12
net.sf.jmoney.gui.MainFrame	class	22251	28	35	63	9	48	6	0	35	229	818	28	4	-13,...	No involvement in pat...	49
net.sf.jmoney.io.QIF	class	15897	4	19	30	9	24	1	0	19	124	252	3	4	-10,...	No involvement in pat...	49
net.sf.jmoney.model.Account	class	8351	26	7	20	2	41	1	0	7	75	522	26	37	-6,132	No involvement in pat...	9
net.sf.jmoney.model.RootCategory	class	626	4	2	2	3	2	2	0	2	4	1	4	2	-4,958	No involvement in pat...	6
net.sf.jmoney.gui.EntryListItemLabelsExt...	class	2522	4	5	12	2	3	6	0	5	18	0	4	1	-8,912	No involvement in pat...	4
net.sf.jmoney.model.AbstractCategory	abstract ...	1263	7	2	2	2	7	1	4	2	11	15	7	7	-2,922	No involvement in pat...	4
net.sf.jmoney.model.EntryListModel	class	941	0	0	6	0	3	3	0	0	7	3	0	2	-0,916	No involvement in pat...	1
net.sf.jmoney.gui.EditableMetalTheme	class	6866	1	1	15	2	33	3	0	1	63	356	1	28	-1,792	No involvement in pat...	3
net.sf.jmoney.model.Session	class	3488	12	4	10	5	14	1	0	4	32	33	12	13	-5,355	No involvement in pat...	16
net.sf.jmoney.gui.EntryListItemLabels	class	5448	10	6	20	1	9	5	1	6	38	16	10	4	-8,028	No involvement in pat...	3
net.sf.jmoney.model.DoubleEntry	class	2888	7	3	5	2	12	2	0	3	26	36	7	11	-4,975	No involvement in pat...	9
net.sf.jmoney.model.Category	interface	317	15	1	4	1	3	1	0	1	3	3	15	3	-2,529	No involvement in pat...	3
net.sf.jmoney.gui.AccountOverviewPanel	class	2409	0	0	16	0	2	5	0	0	17	0	0	1	-2,152	No involvement in pat...	1
net.sf.jmoney.NavigationTreeModel	class	2190	2	3	7	2	7	3	0	3	15	11	2	7	-5,686	No involvement in pat...	4
net.sf.jmoney.gui.EntryListItem	class	5203	3	3	20	3	9	5	1	3	35	12	3	6	-5,916	No involvement in pat...	11
net.sf.jmoney.Constants	interface	3976	33	0	9	0	1	1	0	0	8	0	32	0	-1,027	No involvement in pat...	1
net.sf.jmoney.gui.IncomeExpenseReport...	class	12128	8	15	48	6	18	5	0	20	98	99	8	3	-12,...	No involvement in pat...	18
net.sf.jmoney.model.CategoryTreeModel	class	2144	7	6	7	4	9	3	0	6	19	4	7	8	-7,865	No involvement in pat...	10
net.sf.jmoney.model.SimpleCategory	class	1505	4	2	4	3	5	2	0	2	14	0	4	5	-4,925	No involvement in pat...	5
net.sf.jmoney.io.MT940	class	8270	3	9	24	9	7	1	0	8	59	15	2	4	-8,476	No involvement in pat...	49
net.sf.jmoney.Currency	class	4780	6	1	22	1	15	1	0	0	45	75	6	11	0,828	No involvement in pat...	2

Figure 6- COPE’s main window after a successful static analysis

In the red area COPE provides file structure of the source code for the JMoney project (with nodes representing the packages and leaves representing the classes).

The green area is presenting the aforementioned data provided by the static analysis.

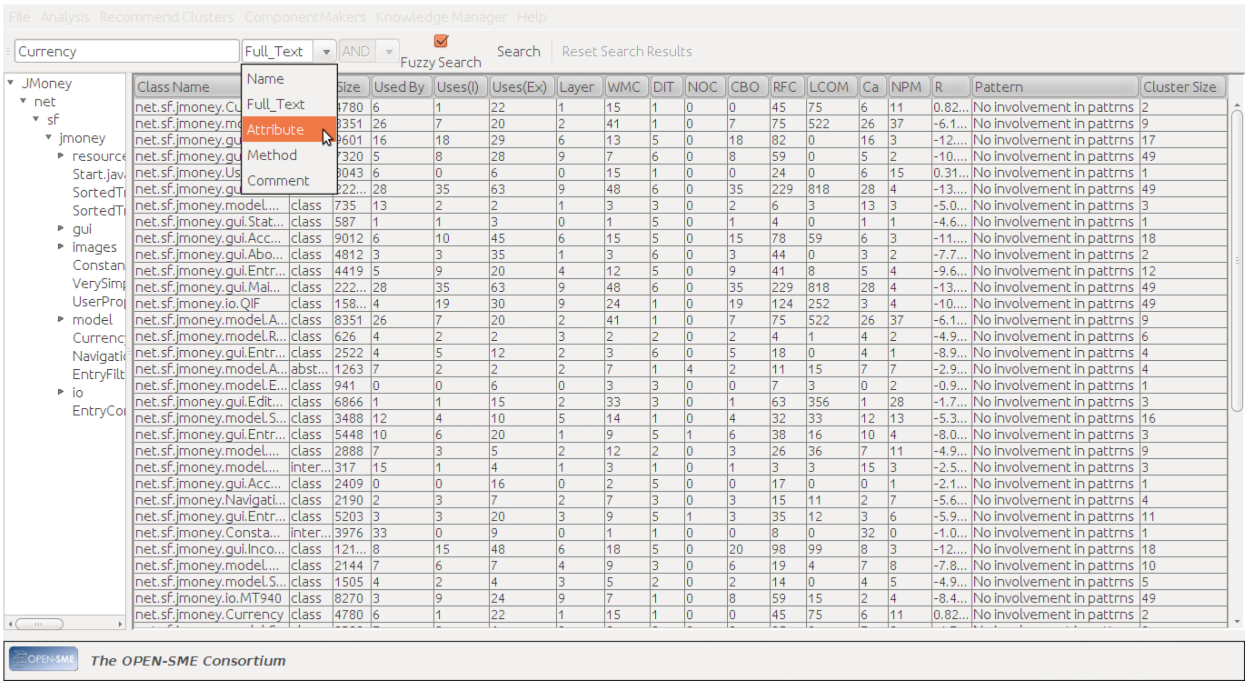
1.3.5.2 PERFORMING SOURCE FILE INDEXING

Source File Indexing is another analyser that creates an index from the source files of a Reuse Project. This index enables the feature of the internal search engine we mentioned in the beginning of section 2 and provides five different fields for targeted search:

- **Full Text:** performs search for the specified keywords to the whole class file
- **Name:** performs search for the specified keywords to the name of the class file
- **Attribute:** performs search for the specified keywords to the attributes of the class file
- **Method:** performs search for the specified keywords to the methods of the class file
- **Comment:** performs search for the specified keywords to the comments of the class file

To perform Source File Indexing for a Reuse Project in COPE:

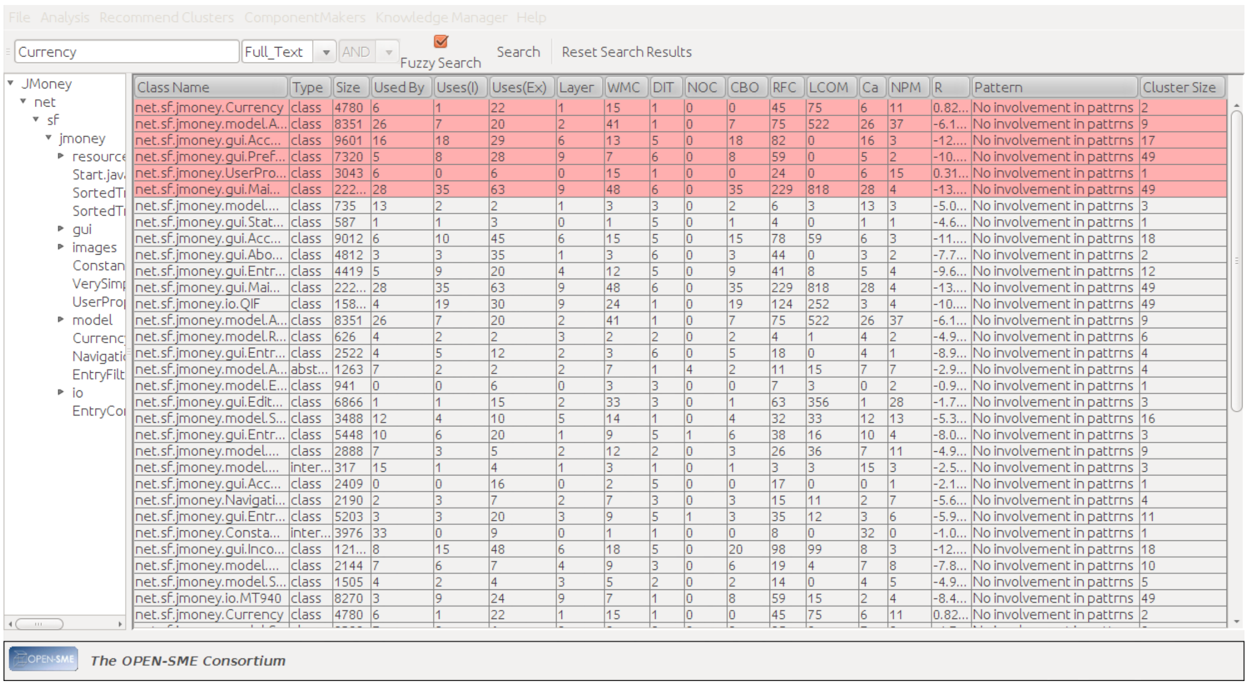
1. Select **"Source File Indexing…"** from the **Analysis** menu.
2. In the dialog that appears (entitled "Source File Indexing") click to the **Start** button.
3. When the analysis is complete (the progress bar has reached 100%) you can close the "Source File Indexing" dialog using the "X" button.



Class Name		Size	Used By	Uses(I)	Uses(Ex)	Layer	WMC	DIT	NOC	CBO	RFC	LCOM	Ca	NPM	R	Pattern	Cluster Size
net.sf.jmoney.Cu		4780	6	1	22	1	15	1	0	0	45	75	6	11	0.82...	No involvement in pattrns	2
net.sf.jmoney.mo		8351	26	7	20	2	41	1	0	7	75	522	26	37	-6.1...	No involvement in pattrns	9
net.sf.jmoney.gu		9601	16	18	29	6	13	5	0	18	82	0	16	3	-12....	No involvement in pattrns	17
net.sf.jmoney.gu		7320	5	8	28	9	7	6	0	8	59	0	5	2	-10....	No involvement in pattrns	49
net.sf.jmoney.Us		3043	6	0	6	0	15	1	0	0	24	0	6	15	0.31...	No involvement in pattrns	1
net.sf.jmoney.gu		222...	28	35	63	9	48	6	0	35	229	818	28	4	-13....	No involvement in pattrns	49
net.sf.jmoney.model....	class	735	13	2	2	1	3	3	0	2	6	3	13	3	-5.0...	No involvement in pattrns	3
net.sf.jmoney.gui.Stat...	class	587	1	1	3	0	1	5	0	1	4	0	1	1	-4.6...	No involvement in pattrns	1
net.sf.jmoney.gui.Acc...	class	9012	6	10	45	6	15	5	0	15	78	59	6	3	-11....	No involvement in pattrns	18
net.sf.jmoney.gui.Abo...	class	4812	3	3	35	1	3	6	0	3	44	0	3	2	-7.7...	No involvement in pattrns	2
net.sf.jmoney.gui.Entr...	class	4419	5	9	20	4	12	5	0	9	41	8	5	4	-9.6...	No involvement in pattrns	12
net.sf.jmoney.gui.Mai...	class	222...	28	35	63	9	48	6	0	35	229	818	28	4	-13....	No involvement in pattrns	49
net.sf.jmoney.io.QIF	class	158...	4	19	30	9	24	1	0	19	124	252	3	4	-10....	No involvement in pattrns	49
net.sf.jmoney.model.A...	class	8351	26	7	20	2	41	1	0	7	75	522	26	37	-6.1...	No involvement in pattrns	9
net.sf.jmoney.model.R...	class	626	4	2	2	3	2	2	0	2	4	1	4	2	-4.9...	No involvement in pattrns	6
net.sf.jmoney.gui.Entr...	class	2522	4	5	12	2	3	6	0	5	18	0	4	1	-8.9...	No involvement in pattrns	4
net.sf.jmoney.model.A...	abst...	1263	7	2	2	2	7	1	4	2	11	15	7	7	-2.9...	No involvement in pattrns	4
net.sf.jmoney.model.E...	class	941	0	0	6	0	3	3	0	0	7	3	0	2	-0.9...	No involvement in pattrns	1
net.sf.jmoney.gui.Edit...	class	6866	1	1	15	2	33	3	0	1	63	356	1	28	-1.7...	No involvement in pattrns	3
net.sf.jmoney.model.S...	class	3488	12	4	10	5	14	1	0	4	32	33	12	13	-5.3...	No involvement in pattrns	16
net.sf.jmoney.gui.Entr...	class	5448	10	6	20	1	9	5	1	6	38	16	10	4	-8.0...	No involvement in pattrns	3
net.sf.jmoney.model....	class	2888	7	3	5	2	12	2	0	3	26	36	7	11	-4.9...	No involvement in pattrns	9
net.sf.jmoney.model....	inter...	317	15	1	4	1	3	1	0	1	3	3	15	3	-2.5...	No involvement in pattrns	3
net.sf.jmoney.gui.Acc...	class	2409	0	0	16	0	2	5	0	0	17	0	0	1	-2.1...	No involvement in pattrns	1
net.sf.jmoney.Navigati...	class	2190	2	3	7	2	7	3	0	3	15	11	2	7	-5.6...	No involvement in pattrns	4
net.sf.jmoney.gui.Entr...	class	5203	3	3	20	3	9	5	1	3	35	12	3	6	-5.9...	No involvement in pattrns	11
net.sf.jmoney.Consta...	inter...	3976	33	0	9	0	1	1	0	0	8	0	32	0	-1.0...	No involvement in pattrns	1
net.sf.jmoney.gui.Inco...	class	121...	8	15	48	6	18	5	0	20	98	99	8	3	-12....	No involvement in pattrns	18
net.sf.jmoney.model....	class	2144	7	6	7	4	9	3	0	6	19	4	7	8	-7.8...	No involvement in pattrns	10
net.sf.jmoney.model.S...	class	1505	4	2	4	3	5	2	0	2	14	0	4	5	-4.9...	No involvement in pattrns	5
net.sf.jmoney.io.MT940	class	8270	3	9	24	9	7	1	0	8	59	15	2	4	-8.4...	No involvement in pattrns	49
net.sf.jmoney.Currency	class	4780	6	1	22	1	15	1	0	0	45	75	6	11	0.82...	No involvement in pattrns	2

Figure 7- COPE performing Source File Indexing

<u>CASE STUDY</u>: Having performed Source File Indexing for JMoney Reuse Project we can now use the internal search engine to find classes that match specific queries. For example a search for the term "Currency" using the field of "Full text" will return (in the form of highlighted rows) six classes of the JMoney Reuse Project (see Figure 8- Searching for classes using COPE's internal search engine*)*



Class Name	Type	Size	Used By	Uses(I)	Uses(Ex)	Layer	WMC	DIT	NOC	CBO	RFC	LCOM	Ca	NPM	R	Pattern	Cluster Size
net.sf.jmoney.Currency	class	4780	6	1	22	1	15	1	0	0	45	75	6	11	0.82...	No involvement in pattrns	2
net.sf.jmoney.model.A...	class	8351	26	7	20	2	41	1	0	7	75	522	26	37	-6.1...	No involvement in pattrns	9
net.sf.jmoney.gui.Acc...	class	9601	16	18	29	6	13	5	0	18	82	0	16	3	-12....	No involvement in pattrns	17
net.sf.jmoney.gui.Pref...	class	7320	5	8	28	9	7	6	0	8	59	0	5	2	-10....	No involvement in pattrns	49
net.sf.jmoney.UserPro...	class	3043	6	0	6	0	15	1	0	0	24	0	6	15	0.31...	No involvement in pattrns	1
net.sf.jmoney.gui.Mai...	class	222...	28	35	63	9	48	6	0	35	229	818	28	4	-13....	No involvement in pattrns	49
net.sf.jmoney.model....	class	735	13	2	2	1	3	3	0	2	6	3	13	3	-5.0...	No involvement in pattrns	3
net.sf.jmoney.gui.Stat...	class	587	1	1	3	0	1	5	0	1	4	0	1	1	-4.6...	No involvement in pattrns	1
net.sf.jmoney.gui.Acc...	class	9012	6	10	45	6	15	5	0	15	78	59	6	3	-11....	No involvement in pattrns	18
net.sf.jmoney.gui.Abo...	class	4812	3	3	35	1	3	6	0	3	44	0	3	2	-7.7...	No involvement in pattrns	2
net.sf.jmoney.gui.Entr...	class	4419	5	9	20	4	12	5	0	9	41	8	5	4	-9.6...	No involvement in pattrns	12
net.sf.jmoney.gui.Mai...	class	222...	28	35	63	9	48	6	0	35	229	818	28	4	-13....	No involvement in pattrns	49
net.sf.jmoney.io.QIF	class	158...	4	19	30	9	24	1	0	19	124	252	3	4	-10....	No involvement in pattrns	49
net.sf.jmoney.model.A...	class	8351	26	7	20	2	41	1	0	7	75	522	26	37	-6.1...	No involvement in pattrns	9
net.sf.jmoney.model.R...	class	626	4	2	2	3	2	2	0	2	4	1	4	2	-4.9...	No involvement in pattrns	6
net.sf.jmoney.gui.Entr...	class	2522	4	5	12	2	3	6	0	5	18	0	4	1	-8.9...	No involvement in pattrns	4
net.sf.jmoney.model.A...	abst...	1263	7	2	2	2	7	1	4	2	11	15	7	7	-2.9...	No involvement in pattrns	4
net.sf.jmoney.model.E...	class	941	0	0	6	0	3	3	0	0	7	3	0	2	-0.9...	No involvement in pattrns	1
net.sf.jmoney.gui.Edit...	class	6866	1	1	15	2	33	3	0	1	63	356	1	28	-1.7...	No involvement in pattrns	3
net.sf.jmoney.model.S...	class	3488	12	4	10	5	14	1	0	4	32	33	12	13	-5.3...	No involvement in pattrns	16
net.sf.jmoney.gui.Entr...	class	5448	10	6	20	1	9	5	1	6	38	16	10	4	-8.0...	No involvement in pattrns	3
net.sf.jmoney.model....	class	2888	7	3	5	2	12	2	0	3	26	36	7	11	-4.9...	No involvement in pattrns	9
net.sf.jmoney.model....	inter...	317	15	1	4	1	3	1	0	1	3	3	15	3	-2.5...	No involvement in pattrns	3
net.sf.jmoney.gui.Acc...	class	2409	0	0	16	0	2	5	0	0	17	0	0	1	-2.1...	No involvement in pattrns	1
net.sf.jmoney.Navigati...	class	2190	2	3	7	2	7	3	0	3	15	11	2	7	-5.6...	No involvement in pattrns	4
net.sf.jmoney.gui.Entr...	class	5203	3	3	20	3	9	5	1	3	35	12	3	6	-5.9...	No involvement in pattrns	11
net.sf.jmoney.Consta...	inter...	3976	33	0	9	0	1	1	0	0	8	0	32	0	-1.0...	No involvement in pattrns	1
net.sf.jmoney.gui.Inco...	class	121...	8	15	48	6	18	5	0	20	98	99	8	3	-12....	No involvement in pattrns	18
net.sf.jmoney.model....	class	2144	7	6	7	4	9	3	0	6	19	4	7	8	-7.8...	No involvement in pattrns	10
net.sf.jmoney.model.S...	class	1505	4	2	4	3	5	2	0	2	14	0	4	5	-4.9...	No involvement in pattrns	5
net.sf.jmoney.io.MT940	class	8270	3	9	24	9	7	1	0	8	59	15	2	4	-8.4...	No involvement in pattrns	49
net.sf.jmoney.Currency	class	4780	6	1	22	1	15	1	0	0	45	75	6	11	0.82...	No involvement in pattrns	2

Figure 8- Searching for classes using COPE's internal search engine

1.3.5.3 PERFORMING DOCUMENTATION GENERATION

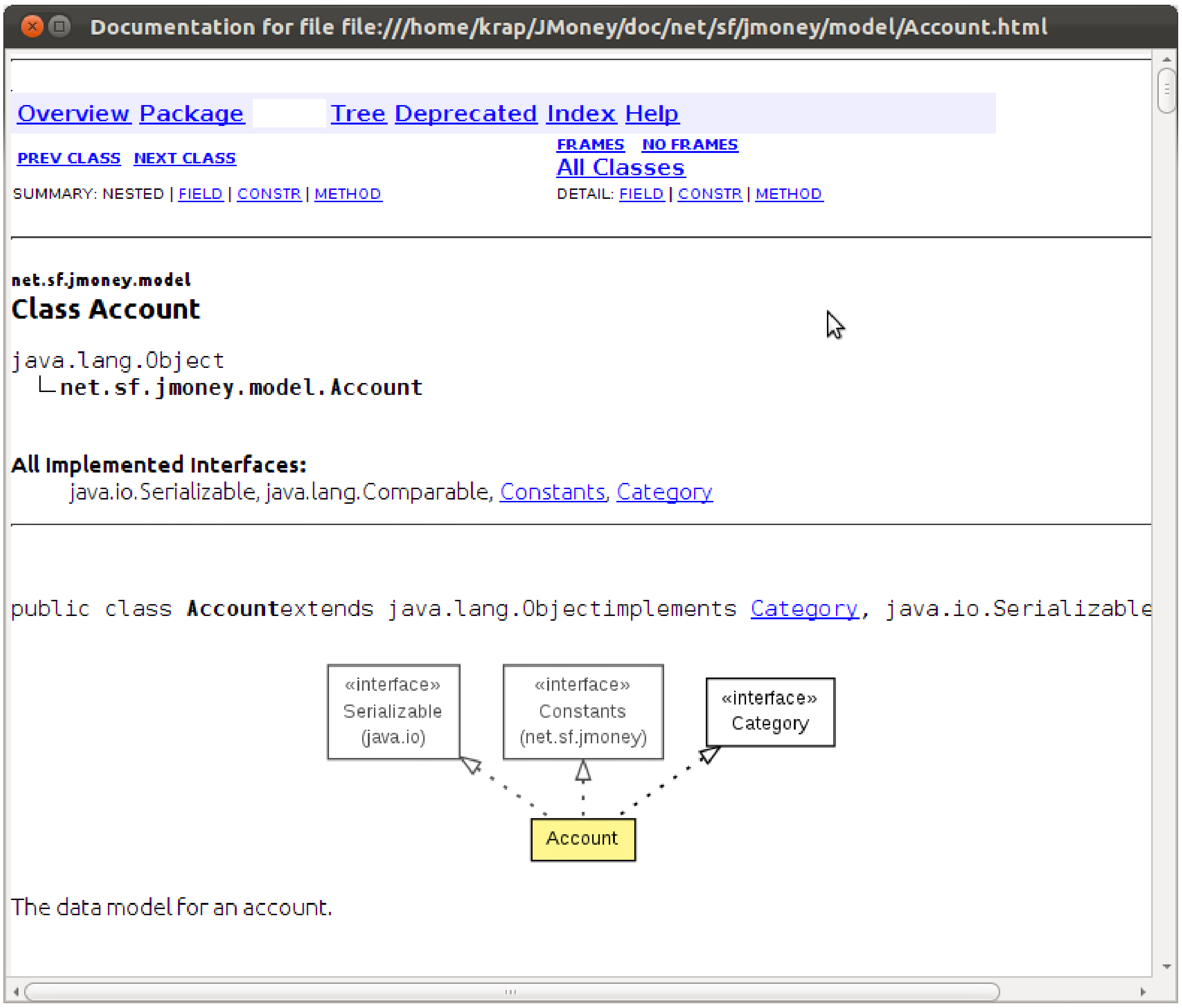
Documentation Generation creates the Javadoc documentation for the source code of the Reuse Project.

To perform Documentation Generation for a Reuse Project in COPE:

1. Select **"Documentation Generation…"** from the **Analysis** menu.
2. In the dialog that appears click to the **Start** button.
3. When the generation is complete (the progress bar has reached 100%) you can close the process dialog using the "X" button.

CASE STUDY: After the Documentation Generation process is finished the Reuse Engineer is able to see the documentation for a class of the Reuse Project by selecting this class from the tree view located in the left of COPE's main window (see Figure 6- COPE's main window after a successful static analysis*).*

Let's say, for example, that in the JMoney Reuse Project, we would like to see the documentation of the class Account. Selecting it with the aforementioned way would provide us with the following dialog:


Documentation for file file:///home/krap/JMoney/doc/net/sf/jmoney/model/Account.html

Overview Package Tree Deprecated Index Help

PREV CLASS NEXT CLASS

FRAMES NO FRAMES

All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

net.sf.jmoney.model

Class Account

```
java.lang.Object
  └net.sf.jmoney.model.Account
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable, Constants, Category

```
public class Accountextends java.lang.Objectimplements Category, java.io.Serializable
```

«interface» Serializable (java.io)
«interface» Constants (net.sf.jmoney)
«interface» Category
Account

The data model for an account.

Figure 9-Documentation for the class Account of the JMoney Reuse Project

NOTE THAT: besides the standard javadoc, uml diagrams are also included. COPE internally uses the apiviz doclet[6] which provides the generated documentation.

1.3.5.4 PERFORMING DYNAMIC ANALYSIS

Dynamic Analysis, gives the opportunity to the Reuse Engineer to test and validate the components he extracts from COPE's Reuse Projects. More specifically, Dynamic Analysis can provide the following information:

- Statement Coverage of the Component
- Statement Coverage per Method of the Component
- Linear Code Sequence and Jump (LCSAJ) coverage of the Component
- LCSAJ Coverage per Method of the Component
- Control Flow Graph per Method of the Component

[6] http://code.google.com/p/apiviz/

- Automatic Functional Test Generation

<u>NOTE THAT</u>:

- In order for the Dynamic Analysis to be performed at least one component should have been generated using one of the available Component Makers (see 1.3.7).
- For the needs of this section we are going to introduce and work with the Account component extracted from the JMoney Reuse Project. If you want to know more about the Component Making Process please see Section 1.3.7).

To perform Dynamic Analysis for a Reuse Project in COPE:

1. Select **"Dynamic Analysis"** from the **Analysis** menu.

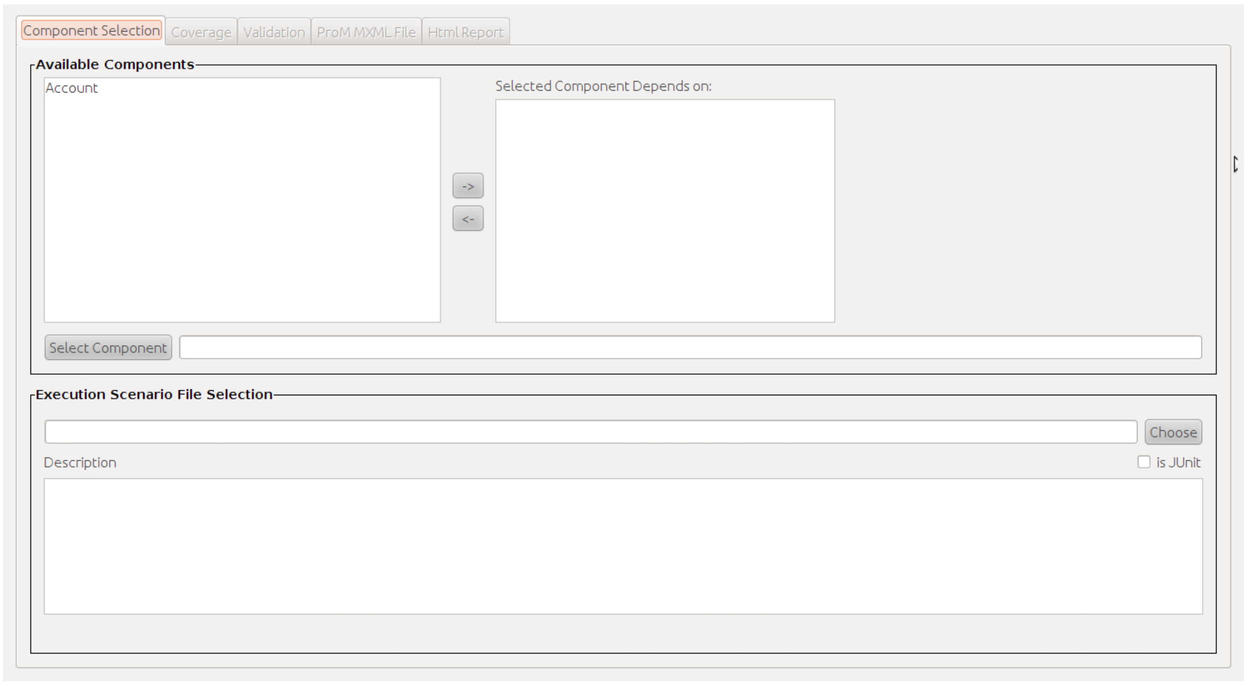


Figure 10- COPE's Dynamic Analysis: Component Selection Tab

2. In the first tab (**Component Selection**)
   a. Select the component to perform dynamic analysis and click to the "Select Component" button. The available components extracted so far from the open project will be visible in the 'Available Components' list.
   b. If the selected component depends on other components, select those components using the arrow buttons provided to the upper right area of the dialog.
   c. Define the execution scenario for the component to be tested. An execution scenario consists of a fully functional class that uses the main functionality of a specific component.

<u>NOTICE THAT</u>: Many different execution scenarios can be developed and provided here for the same component.

<u>CASE STUDY</u>: For the Account component of the JMoney Reuse project a possible execution scenario could be the following:

```
public class JMoneyAccountExample {

      public static void main(String[] args) {

            Account account = new Account();

            account.setMinBalance(0L);

            account.setAccountNumber("100000GKB");

            account.setCurrencyCode("EUR");

            account.setName("Joan Doe");

            account.setBank("GKBank");

            account.setComment("An artificial account");

            //add some entries in the account

            Entry entry1 = new Entry();

            entry1.setAmount(2000);

            entry1.setDescription("Salary deposit");

            entry1.setDate(new Date());

            account.addEntry(entry1);


            Entry entry2 = new Entry();

            entry2.setAmount(-30);

            entry2.setDescription("Withdrawal of 30 euros");

            entry2.setDate(new Date());

            account.addEntry(entry2);


            Vector v =account.getEntries();

            Iterator i = v.iterator();

            while (i.hasNext()) {

                  Entry e = (Entry) i.next();
                  System.out.println(
                    "Entry date:" + e.getDate() + " Entry amount: " +
                        e.getAmount() + " <"+e.getDescription() + ">");

            }

      }

}
```

In this case, the execution scenario is a class (JAccountExample) which exercises the main functionality of the Account class in its main method.

d. Click the “Choose” button and browse to the class representing the execution scenario *(e.g.*

for the Account Component, JAccountExample.java)

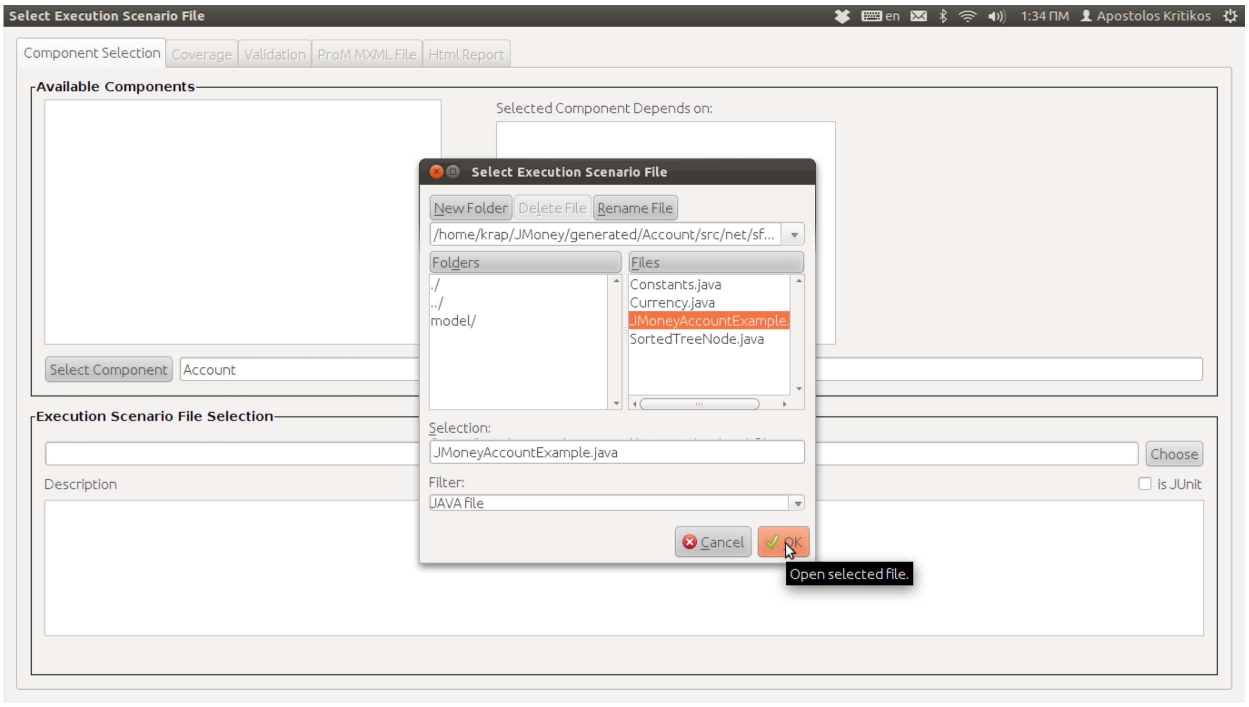


Figure 11- Selecting an execution scenario

e. If the execution scenario class is a JUnit test, check the checkbox “is Junit”.

f. Provide a description for the selected component in the “Description” text area. This documentation, if provided, will be included in the HTML generated documentation for the component tests (see Section Performing Dynamic Analysis).

3. In the second tab (**Coverage**)

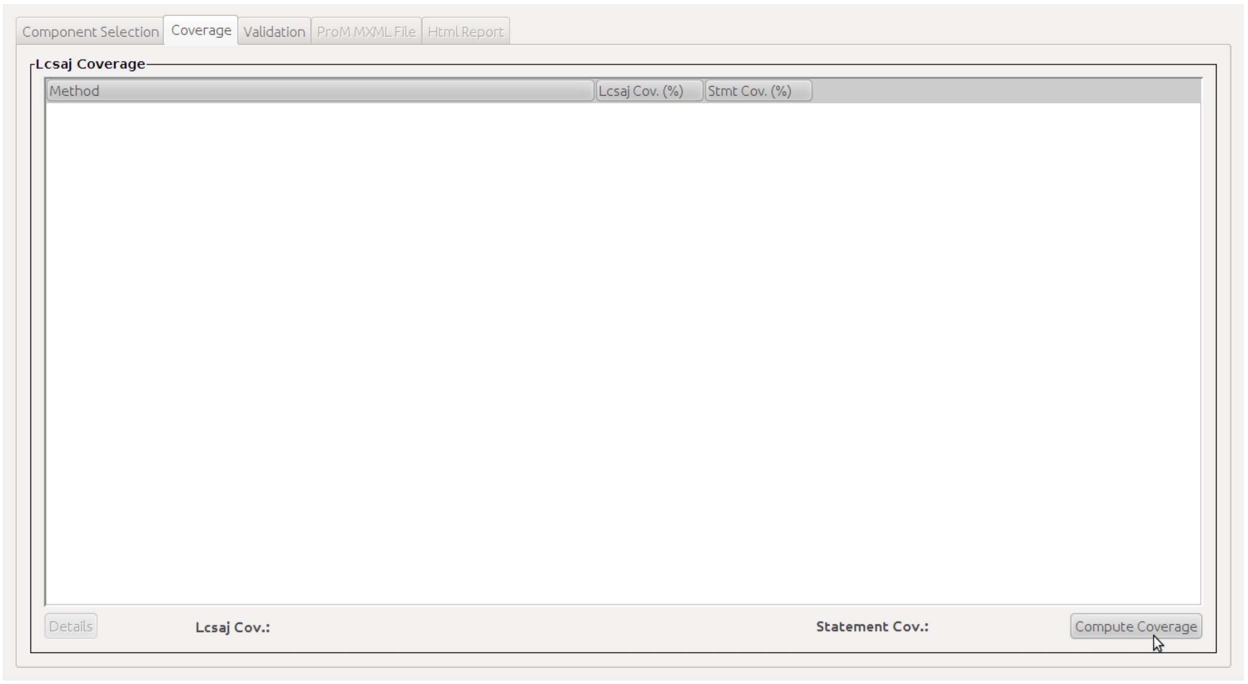
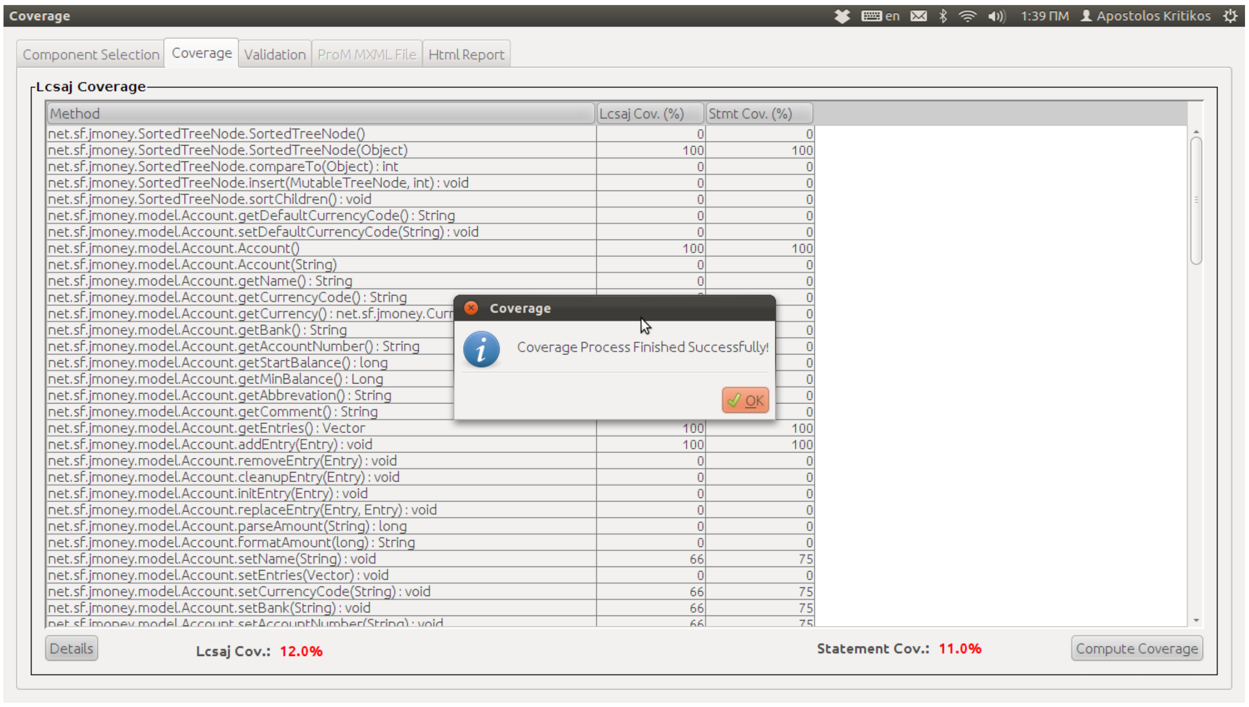


Figure 12- Dynamic Analysis: Coverage Tab

a. Click to the “Compute Coverage” button.



Method	Lcsaj Cov. (%)	Stmt Cov. (%)
net.sf.jmoney.SortedTreeNode.SortedTreeNode()	0	0
net.sf.jmoney.SortedTreeNode.SortedTreeNode(Object)	100	100
net.sf.jmoney.SortedTreeNode.compareTo(Object) : int	0	0
net.sf.jmoney.SortedTreeNode.insert(MutableTreeNode, int) : void	0	0
net.sf.jmoney.SortedTreeNode.sortChildren() : void	0	0
net.sf.jmoney.model.Account.getDefaultCurrencyCode() : String	0	0
net.sf.jmoney.model.Account.setDefaultCurrencyCode(String) : void	0	0
net.sf.jmoney.model.Account.Account()	100	100
net.sf.jmoney.model.Account.Account(String)	0	0
net.sf.jmoney.model.Account.getName() : String	0	0
net.sf.jmoney.model.Account.getCurrencyCode() : String		0
net.sf.jmoney.model.Account.getCurrency() : net.sf.jmoney.Curr		0
net.sf.jmoney.model.Account.getBank() : String		0
net.sf.jmoney.model.Account.getAccountNumber() : String		0
net.sf.jmoney.model.Account.getStartBalance() : long		0
net.sf.jmoney.model.Account.getMinBalance() : Long		0
net.sf.jmoney.model.Account.getAbbrevation() : String		0
net.sf.jmoney.model.Account.getComment() : String		0
net.sf.jmoney.model.Account.getEntries() : Vector	100	100
net.sf.jmoney.model.Account.addEntry(Entry) : void	100	100
net.sf.jmoney.model.Account.removeEntry(Entry) : void	0	0
net.sf.jmoney.model.Account.cleanupEntry(Entry) : void	0	0
net.sf.jmoney.model.Account.initEntry(Entry) : void	0	0
net.sf.jmoney.model.Account.replaceEntry(Entry, Entry) : void	0	0
net.sf.jmoney.model.Account.parseAmount(String) : long	0	0
net.sf.jmoney.model.Account.formatAmount(long) : String	0	0
net.sf.jmoney.model.Account.setName(String) : void	66	75
net.sf.jmoney.model.Account.setEntries(Vector) : void	0	0
net.sf.jmoney.model.Account.setCurrencyCode(String) : void	66	75
net.sf.jmoney.model.Account.setBank(String) : void	66	75
net.sf.jmoney.model.Account.setAccountNumber(String) : void	66	75

Coverage Process Finished Successfully!

Lcsaj Cov.: 12.0%

Statement Cov.: 11.0%

Figure 13 – Coverage results

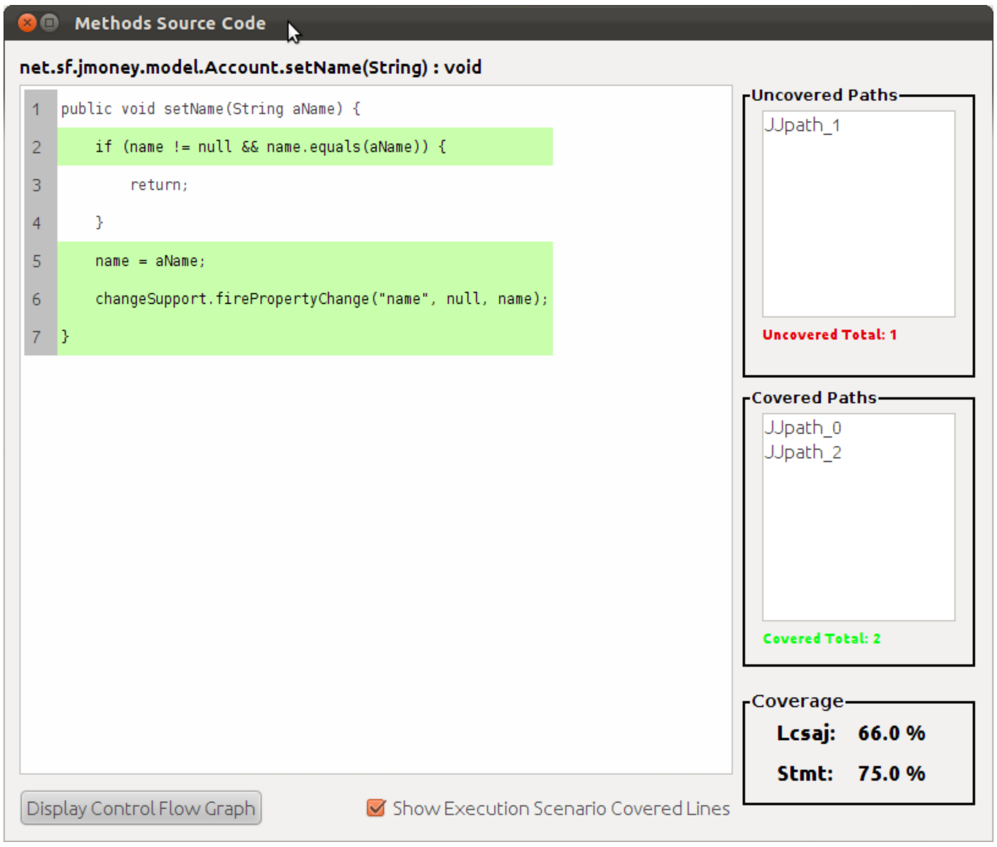
After the coverage computation has finished, the estimated **LCSAJ** and **Statement coverage** metrics appear in red in the bottom of the coverage results dialog (see Figure 13 – Coverage results).

Moreover by selecting a specific method from the table in the centre of the dialog and clicking to the “Details” button, the Reuse Engineer is prompted with a new dialog which presents new coverage information, specifically for the selected method.

As you can see in “Figure 14 – Coverage information for Account’s setName method” this window provides not only the LCSAJ and statement coverage metrics for the selected method but also it visualizes the paths that were covered by the execution scenario we provided (text highlighted in green).

The same dialog provides the option of displaying the Control Flow Graph for a specific method. By clicking the “Display Control Flow Graph” button, the control flow graph opens in a new dialog (see Figure 15 – Control Flow Graph for Account’s setName method)

NOTE THAT: Control Flow Graph contains an extra statement “System.currentTimeMillis()_”. This statement is placed automatically to support the tracing procedure (using AspectJ) and should be ignored.



Methods Source Code

net.sf.jmoney.model.Account.setName(String) : void

```
public void setName(String aName) {
    if (name != null && name.equals(aName)) {
        return;
    }
    name = aName;
    changeSupport.firePropertyChange("name", null, name);
}
```

Uncovered Paths
JJpath_1
Uncovered Total: 1

Covered Paths
JJpath_0
JJpath_2
Covered Total: 2

Coverage
Lcsaj: 66.0 %
Stmt: 75.0 %

Display Control Flow Graph
Show Execution Scenario Covered Lines

Figure 14 – Coverage information for Account's setName method

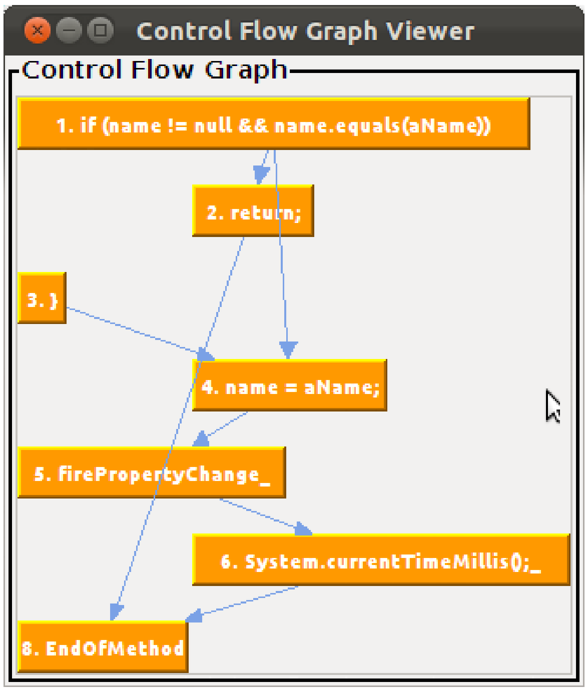
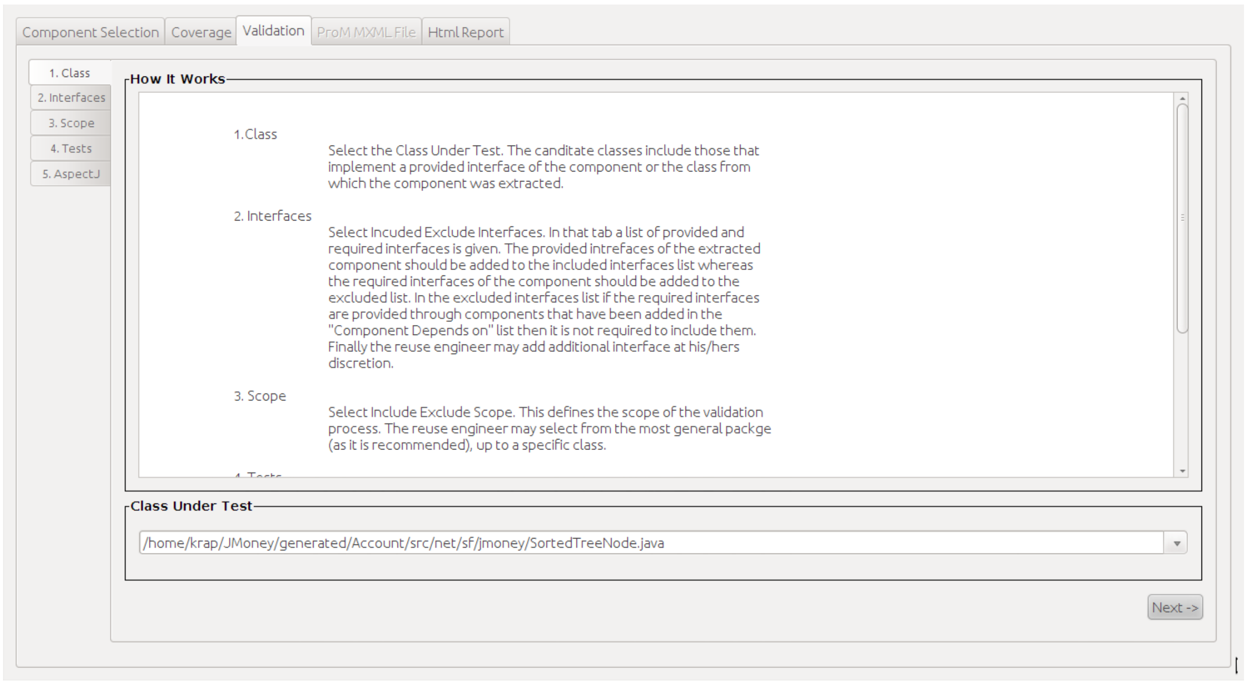

Control Flow Graph Viewer
Control Flow Graph
1. if (name != null && name.equals(aName))
2. return;
3. }
4. name = aName;
5. firePropertyChange_
6. System.currentTimeMillis();_
8. EndOfMethod

Figure 15 – Control Flow Graph for Account's setName method

4. In the third tab (**Validation**) a five step process (given in the form of tabs) guides the Reuse Engineer through the validation phase.

   a. Step 1: **Class**. The candidate classes include those that implement a provided interface of the component or the class from which the component was extracted.



1.Class

Select the Class Under Test. The canditate classes include those that implement a provided interface of the component or the class from which the component was extracted.

2. Interfaces

Select Incuded Exclude Interfaces. In that tab a list of provided and required interfaces is given. The provided intrefaces of the extracted component should be added to the included interfaces list whereas the required interfaces of the component should be added to the excluded list. In the excluded interfaces list if the required interfaces are provided through components that have been added in the "Component Depends on" list then it is not required to include them. Finally the reuse engineer may add additional interface at his/hers discretion.

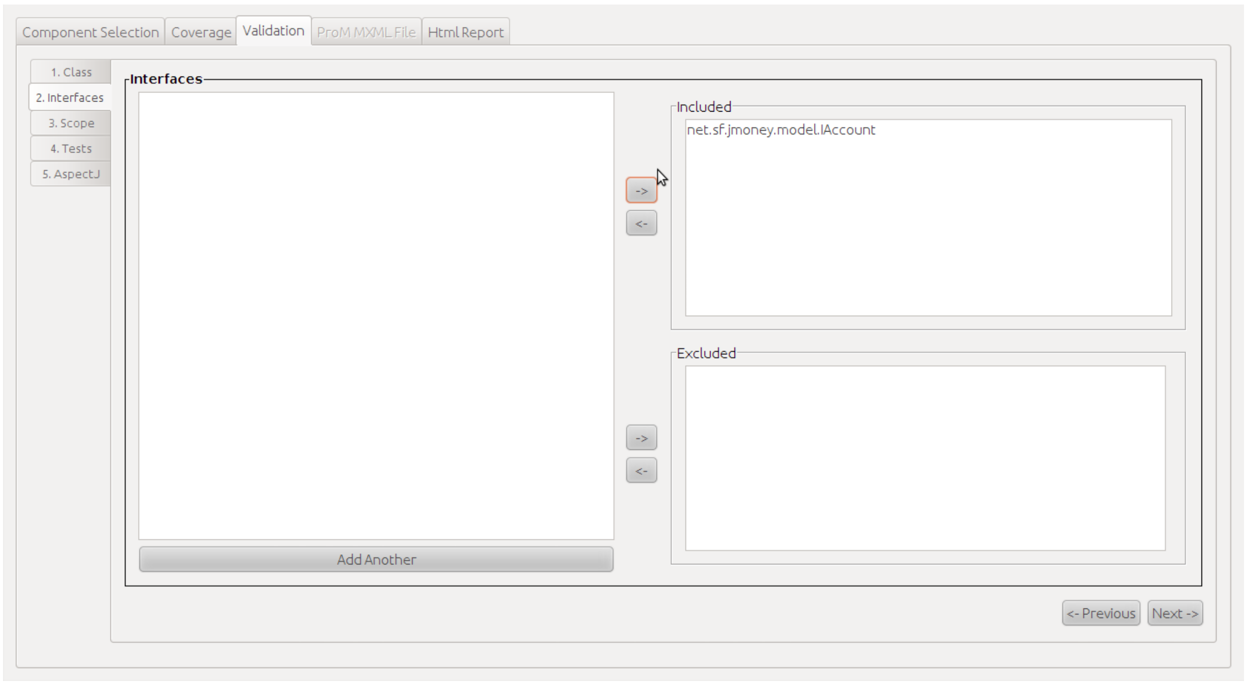
3. Scope

Select Include Exclude Scope. This defines the scope of the validation process. The reuse engineer may select from the most general packge (as it is recommended), up to a specific class.

/home/krap/JMoney/generated/Account/src/net/sf/jmoney/SortedTreeNode.java

Figure 16 – Validation: Class tab

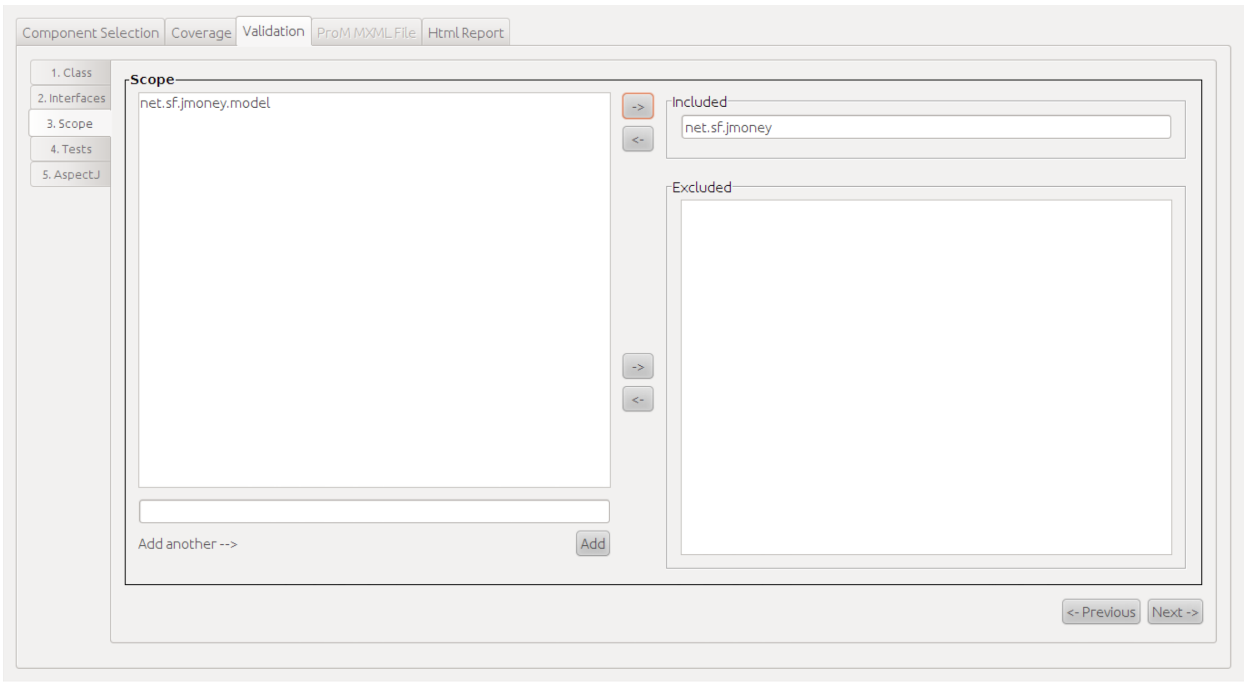
b. Step 2: **Interfaces**. Select Included / Exclude Interfaces. In that tab a list of provided and required interfaces is given. The provided interfaces of the extracted component should be added to the included interfaces list whereas the required interfaces of the component should be added to the excluded list. In the excluded interfaces list if the required interfaces are provided through components that have been added in the “Component Depends on” list then it is not required to include them. Finally the reuse engineer may add additional interface at his/hers discretion.



net.sf.jmoney.model.IAccount

Figure 17 - Validation: Interfaces tab

c. Step 3: **Scope**. Select Include Exclude Scope. This defines the scope of the validation process. The reuse engineer may select from the most general package (as it is recommended), up to a specific class.

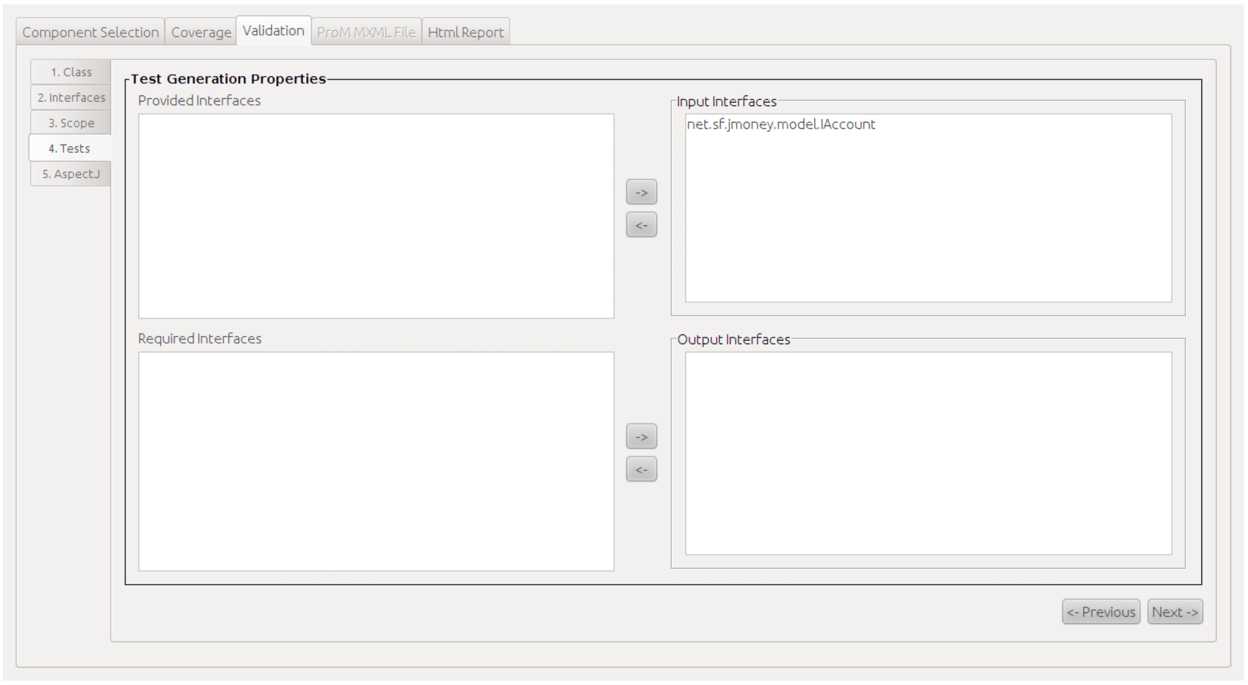


net.sf.jmoney.model

Included: net.sf.jmoney

Figure 18 - Validation: Scope tab

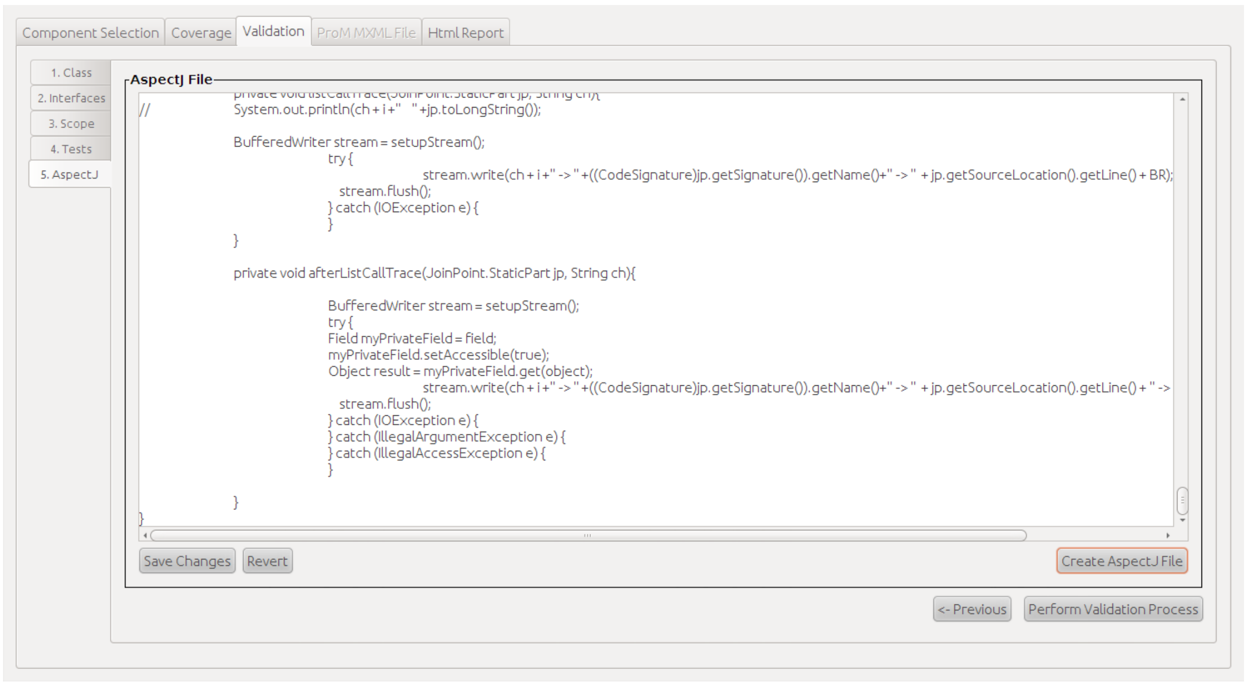
d. **Tests**. Select Test Generation Properties. In this tab the reuse engineer must define the input and output interfaces. Input interfaces are those that are provided by the component and are going to be tested based on the execution scenario. In the same manner, Output interfaces are those that are required by the component and are going to be tested based on the execution scenario. Required interfaces should be implemented to include them in Output list.



Input Interfaces: net.sf.jmoney.model.IAccount

Figure 19 - Validation: Tests tab

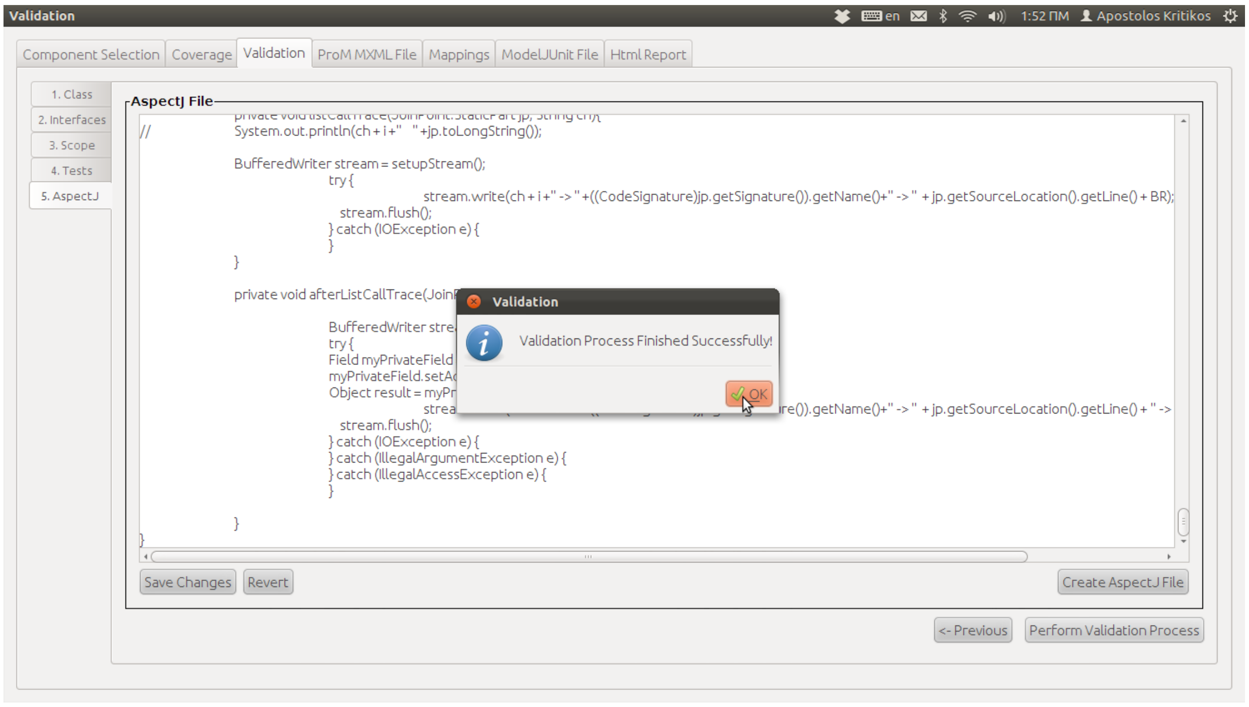
e. **AspectJ**. Aspect Oriented Programming [2] is used to trace component's behaviour as it is executed. An AspectJ[7] file is generated according to the fields of 1, 2 and 3 tabs. Reuse Engineer may make any change in the displayed code and save the changes.



```
            private void listCallTrace(JoinPoint.StaticPart jp, String ch){
//          System.out.println(ch + i + "   " +jp.toLongString());

            BufferedWriter stream = setupStream();
                        try {
                                    stream.write(ch + i + " -> " +((CodeSignature)jp.getSignature()).getName()+" -> " + jp.getSourceLocation().getLine() + BR);
                          stream.flush();
                        } catch (IOException e) {
                        }
            }

            private void afterListCallTrace(JoinPoint.StaticPart jp, String ch){

                        BufferedWriter stream = setupStream();
                        try {
                        Field myPrivateField = field;
                        myPrivateField.setAccessible(true);
                        Object result = myPrivateField.get(object);
                                    stream.write(ch + i + " -> " +((CodeSignature)jp.getSignature()).getName()+" -> " + jp.getSourceLocation().getLine() + " ->
                          stream.flush();
                        } catch (IOException e) {
                        } catch (IllegalArgumentException e) {
                        } catch (IllegalAccessException e) {
                        }

            }
}
```

Figure 20 - Validation: AspectJ tab

f. Once the AspectJ file has been generated the Reuse Engineer can proceed in performing validation process by clicking to "Perform Validation Process" button. If no errors occur a success message is prompted after a while.



Validation Process Finished Successfully!

```
            private void listCallTrace(JoinPoint.StaticPart jp, String ch){
//          System.out.println(ch + i + "   " +jp.toLongString());

            BufferedWriter stream = setupStream();
                        try {
                                    stream.write(ch + i + " -> " +((CodeSignature)jp.getSignature()).getName()+" -> " + jp.getSourceLocation().getLine() + BR);
                          stream.flush();
                        } catch (IOException e) {
                        }
            }

            private void afterListCallTrace(JoinP

                        BufferedWriter stre
                        try {
                        Field myPrivateField
                        myPrivateField.setA
                        Object result = myPr
                                    strea                                  re()).getName()+" -> " + jp.getSourceLocation().getLine() + " ->
                          stream.flush();
                        } catch (IOException e) {
                        } catch (IllegalArgumentException e) {
                        } catch (IllegalAccessException e) {
                        }

            }
}
```

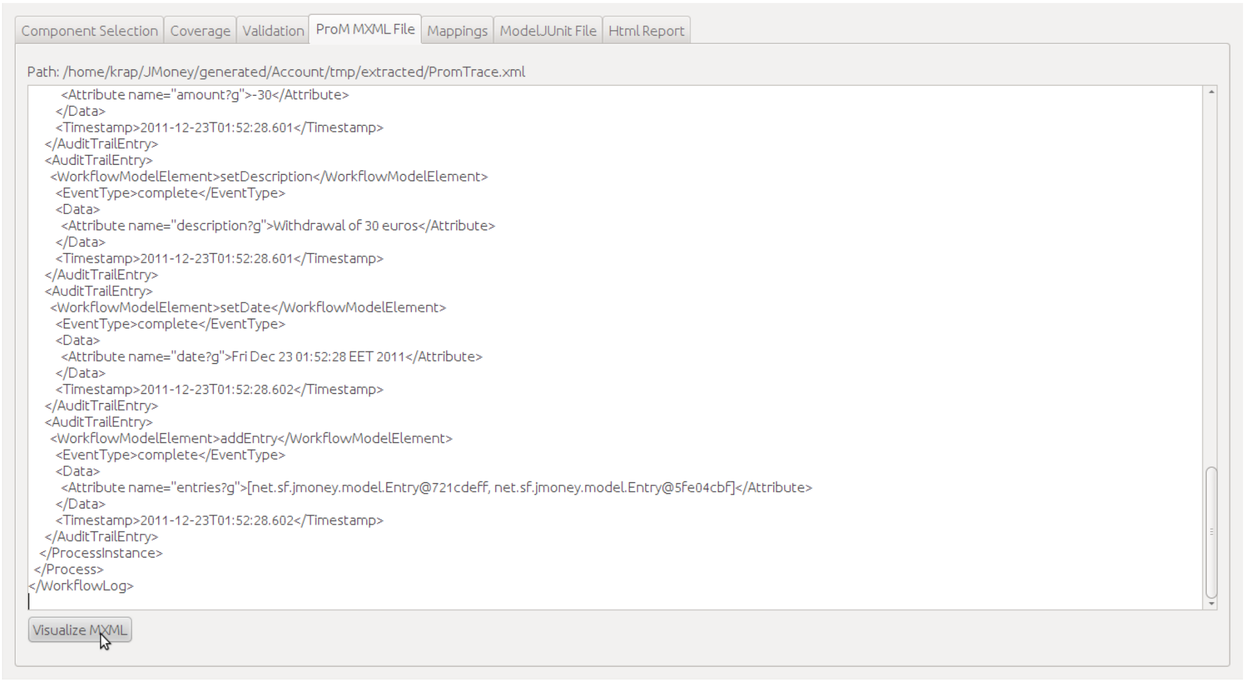
7 http://www.eclipse.org/aspectj/

Figure 21 – Validation process successfully completed

<u>NOTE THAT</u>: Nodes entitled "System.currentTimeMillis()_" serve tracing needs for the AspectJ part of the validation. The Reuse Engineer may simply ignore them.

5. In the fourth tab (**ProM MXML File**)

This tab displays an automatically generated XML file after validation process completed successfully. This XML file represents a Finite State Machine (FSM) automaton. The Reuse Engineer can choose to visualize MXML by clicking to the “Visualize MXML” button to the bottom left corner of the tab. By doing so, the aforementioned FSM is being transformed from its XML version to a user friendlier, graph like version (which is available in a newly created tab named **“FSM Visualization”**).



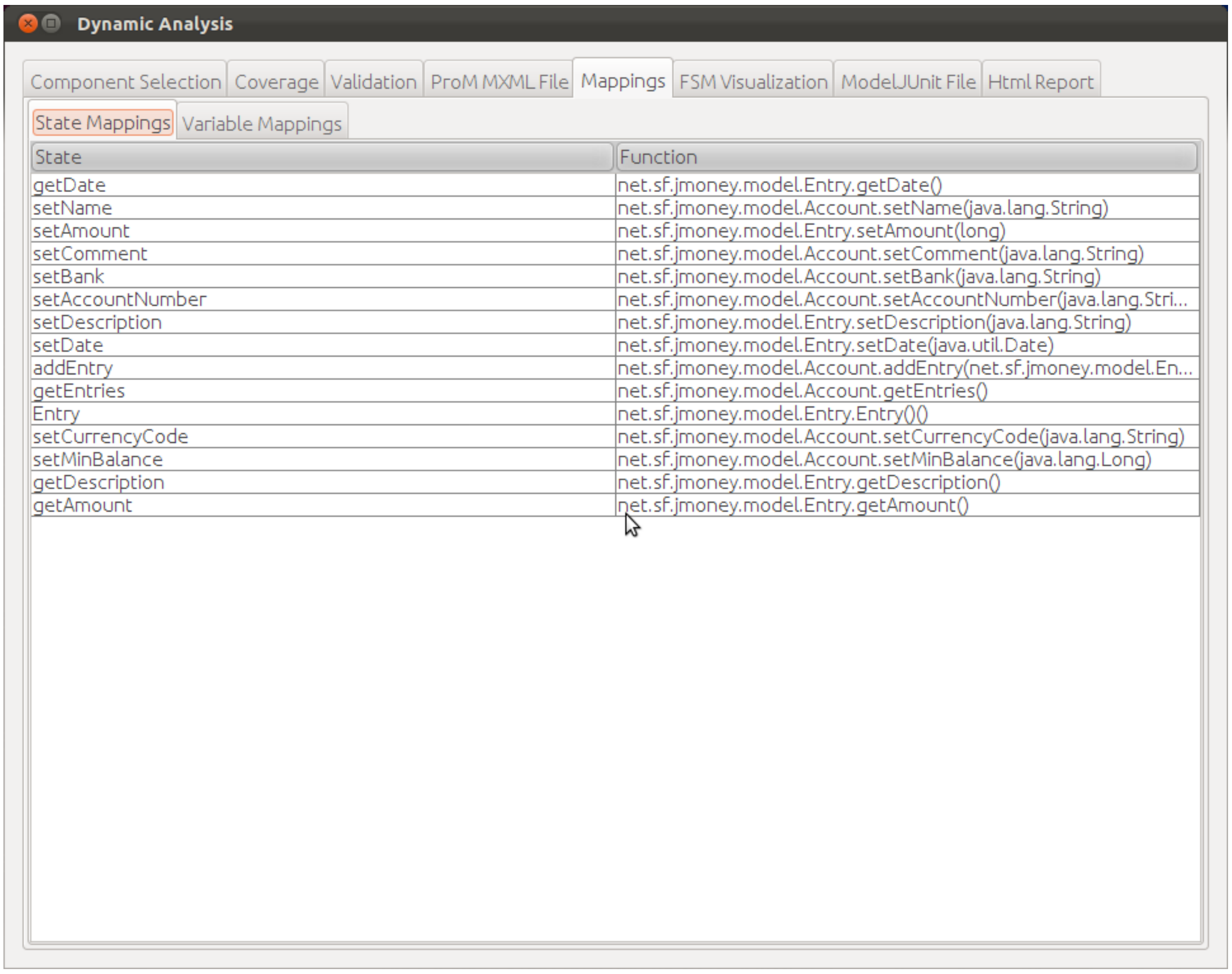
Path: /home/krap/JMoney/generated/Account/tmp/extracted/PromTrace.xml

```
      <Attribute name="amount?g">-30</Attribute>
     </Data>
     <Timestamp>2011-12-23T01:52:28.601</Timestamp>
   </AuditTrailEntry>
   <AuditTrailEntry>
    <WorkflowModelElement>setDescription</WorkflowModelElement>
     <EventType>complete</EventType>
     <Data>
      <Attribute name="description?g">Withdrawal of 30 euros</Attribute>
     </Data>
     <Timestamp>2011-12-23T01:52:28.601</Timestamp>
   </AuditTrailEntry>
   <AuditTrailEntry>
    <WorkflowModelElement>setDate</WorkflowModelElement>
     <EventType>complete</EventType>
     <Data>
      <Attribute name="date?g">Fri Dec 23 01:52:28 EET 2011</Attribute>
     </Data>
     <Timestamp>2011-12-23T01:52:28.602</Timestamp>
   </AuditTrailEntry>
   <AuditTrailEntry>
    <WorkflowModelElement>addEntry</WorkflowModelElement>
     <EventType>complete</EventType>
     <Data>
      <Attribute name="entries?g">[net.sf.jmoney.model.Entry@721cdeff, net.sf.jmoney.model.Entry@5fe04cbf]</Attribute>
     </Data>
     <Timestamp>2011-12-23T01:52:28.602</Timestamp>
   </AuditTrailEntry>
  </ProcessInstance>
 </Process>
</WorkflowLog>
```

Figure 22 – ProM MXML file tab

6. In the fifth tab (**Mappings**)

Maps the states of the FSM automaton produced in the previous step with methods of the selected component and presents them in a tabulated form.

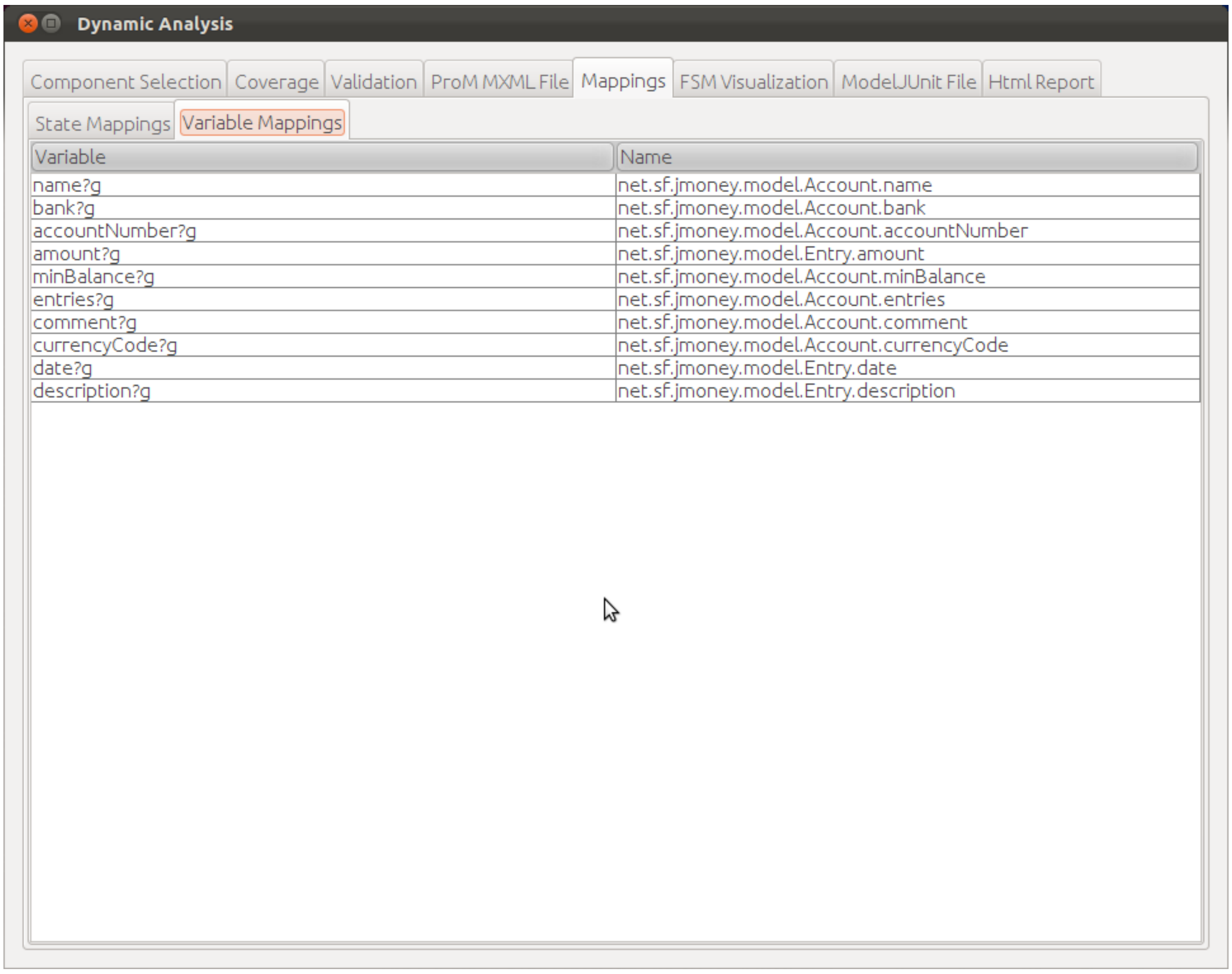


State	Function
getDate	net.sf.jmoney.model.Entry.getDate()
setName	net.sf.jmoney.model.Account.setName(java.lang.String)
setAmount	net.sf.jmoney.model.Entry.setAmount(long)
setComment	net.sf.jmoney.model.Account.setComment(java.lang.String)
setBank	net.sf.jmoney.model.Account.setBank(java.lang.String)
setAccountNumber	net.sf.jmoney.model.Account.setAccountNumber(java.lang.Stri...
setDescription	net.sf.jmoney.model.Entry.setDescription(java.lang.String)
setDate	net.sf.jmoney.model.Entry.setDate(java.util.Date)
addEntry	net.sf.jmoney.model.Account.addEntry(net.sf.jmoney.model.En...
getEntries	net.sf.jmoney.model.Account.getEntries()
Entry	net.sf.jmoney.model.Entry.Entry()()
setCurrencyCode	net.sf.jmoney.model.Account.setCurrencyCode(java.lang.String)
setMinBalance	net.sf.jmoney.model.Account.setMinBalance(java.lang.Long)
getDescription	net.sf.jmoney.model.Entry.getDescription()
getAmount	net.sf.jmoney.model.Entry.getAmount()

Figure 23 – Dynamic Analysis: State Mappings

<u>NOTE THAT</u>: Special characters (in the Variable Mappings section) stand for:

- **?g –** The specific variable is global, therefore it affects component's behaviour
- **?0, ?1**, **?2, …, ?N** – Applies to the first, second, …, Nth parameter



Variable	Name
name?g	net.sf.jmoney.model.Account.name
bank?g	net.sf.jmoney.model.Account.bank
accountNumber?g	net.sf.jmoney.model.Account.accountNumber
amount?g	net.sf.jmoney.model.Entry.amount
minBalance?g	net.sf.jmoney.model.Account.minBalance
entries?g	net.sf.jmoney.model.Account.entries
comment?g	net.sf.jmoney.model.Account.comment
currencyCode?g	net.sf.jmoney.model.Account.currencyCode
date?g	net.sf.jmoney.model.Entry.date
description?g	net.sf.jmoney.model.Entry.description

Figure 24 – Dynamic Analysis: Variable Mappings

7. In the sixth tab (**FSM Visualization)**

In this tab the Reuse Engineer can view the contents of the **ProM MXML File** in a user friendlier version (a graph). Since the resulting FSM can be complex, the tab offers zoom in and zoom out capabilities in order to make it easier for the Reuse Engineer to focus on specific parts of the automaton (if needed).

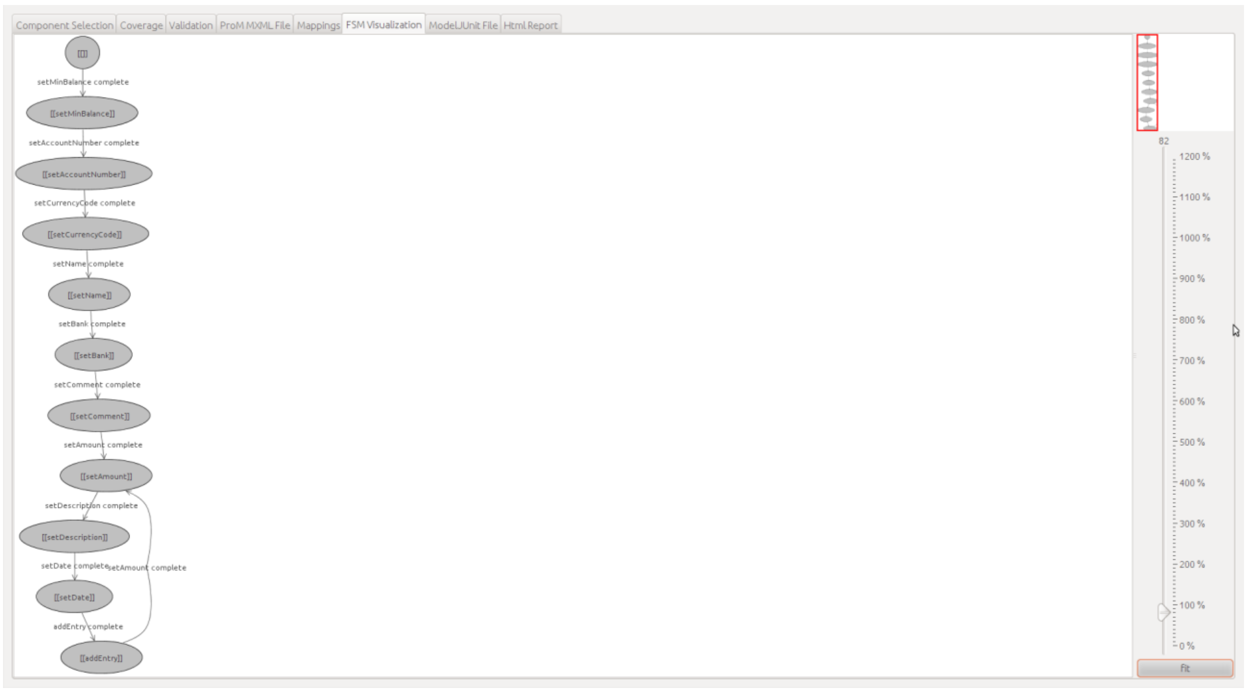
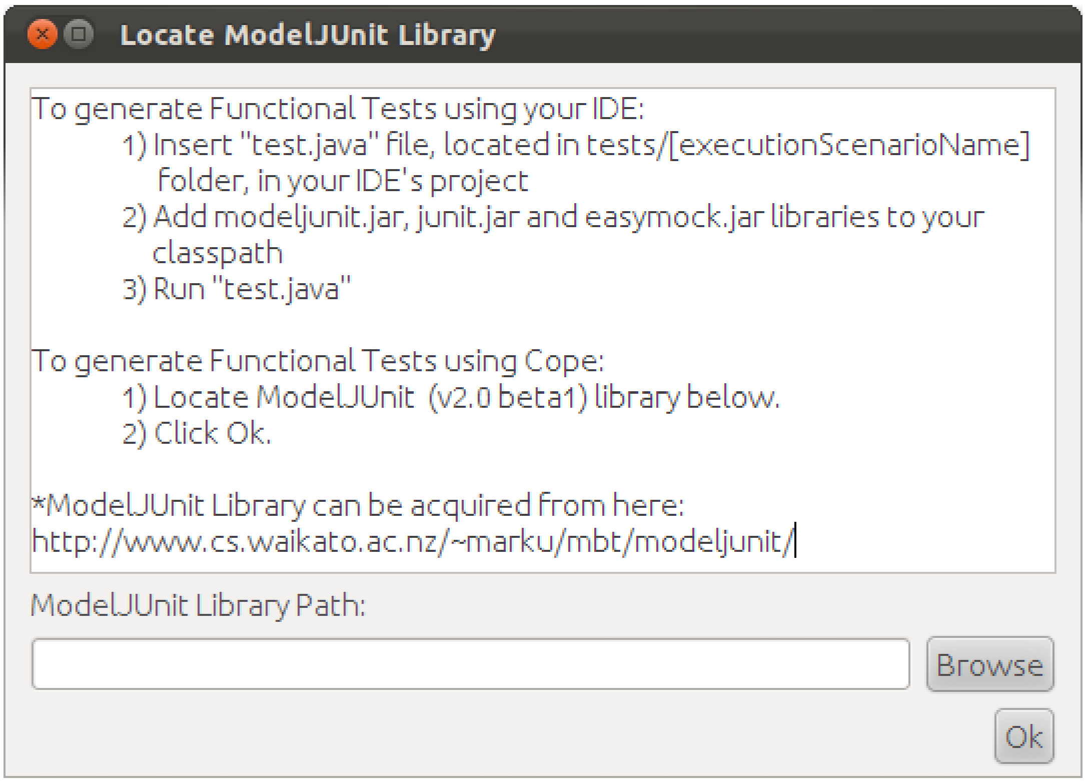


Figure 25 – Dynamic Analysis: Visualize FSM tab

8. In the seventh tab (**ModelJUnitFile**)

This tab uses an external tool, ModelJUnit, to automatically generate functional tests for the selected component. By clicking “Generate Functional Tests” for the first time, the Reuse Engineer is prompted with a dialog that contains detailed guidelines on how to download and use the ModelJUnit external tool.



To generate Functional Tests using your IDE:
1) Insert "test.java" file, located in tests/[executionScenarioName] folder, in your IDE's project
2) Add modeljunit.jar, junit.jar and easymock.jar libraries to your classpath
3) Run "test.java"

To generate Functional Tests using Cope:
1) Locate ModelJUnit (v2.0 beta1) library below.
2) Click Ok.

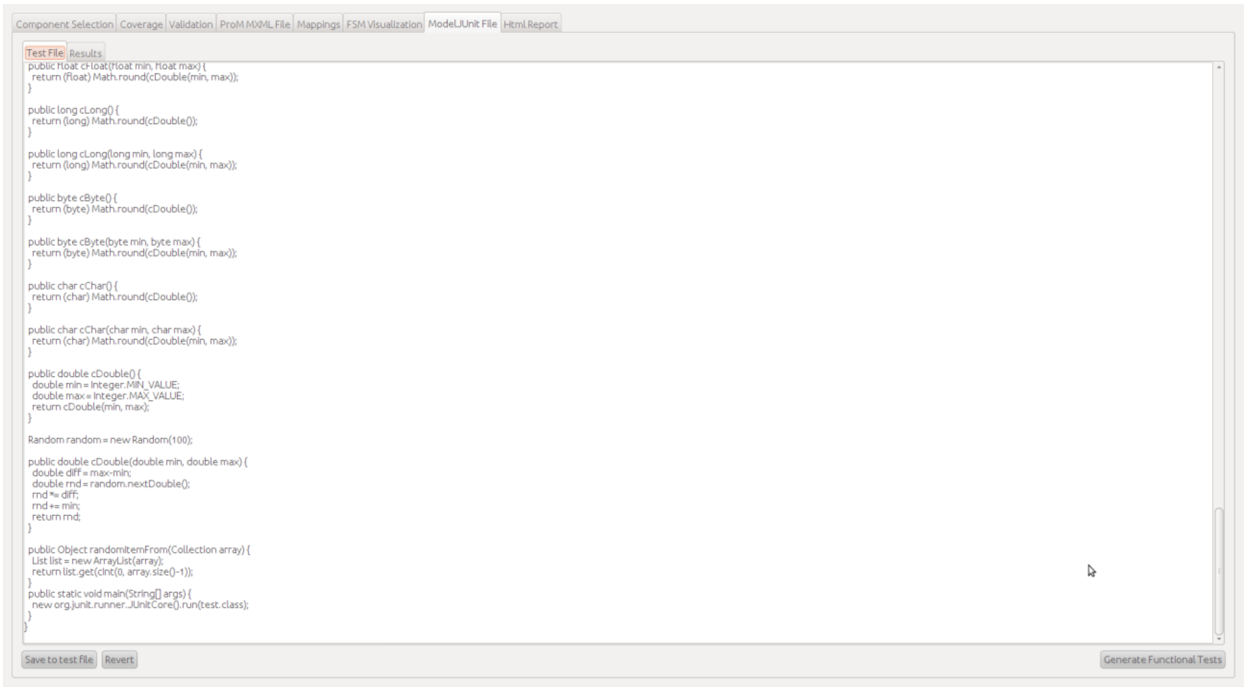
*ModelJUnit Library can be acquired from here:
http://www.cs.waikato.ac.nz/~marku/mbt/modeljunit/

ModelJUnit Library Path:

Figure 26 – Guidelines to use ModelJUnit as an external tool

<u>NOTE THAT</u>: After the ModelJUnit code is created, some methods must be implemented manually by the Reuse Engineer in order for the Functional test to be correct. More specifically, in the generated code the Reuse Engineer should find the following line and implement the methods (which are grouped) after that line:

//---------- TODO IMPLEMENT METHODS IN THIS SECTION TO GENERATE OBJECTS -----------



```
public float cFloat(float min, float max) {
 return (float) Math.round(cDouble(min, max));
}

public long cLong() {
 return (long) Math.round(cDouble());
}

public long cLong(long min, long max) {
 return (long) Math.round(cDouble(min, max));
}

public byte cByte() {
 return (byte) Math.round(cDouble());
}

public byte cByte(byte min, byte max) {
 return (byte) Math.round(cDouble(min, max));
}

public char cChar() {
 return (char) Math.round(cDouble());
}

public char cChar(char min, char max) {
 return (char) Math.round(cDouble(min, max));
}

public double cDouble() {
 double min = Integer.MIN_VALUE;
 double max = Integer.MAX_VALUE;
 return cDouble(min, max);
}

Random random = new Random(100);

public double cDouble(double min, double max) {
 double diff = max-min;
 double rnd = random.nextDouble();
 rnd *= diff;
 rnd += min;
 return rnd;
}

public Object randomItemFrom(Collection array) {
 List list = new ArrayList(array);
 return list.get(cInt(0, array.size()-1));
}
public static void main(String[] args) {
 new org.junit.runner.JUnitCore().run(test.class);
}
}
```

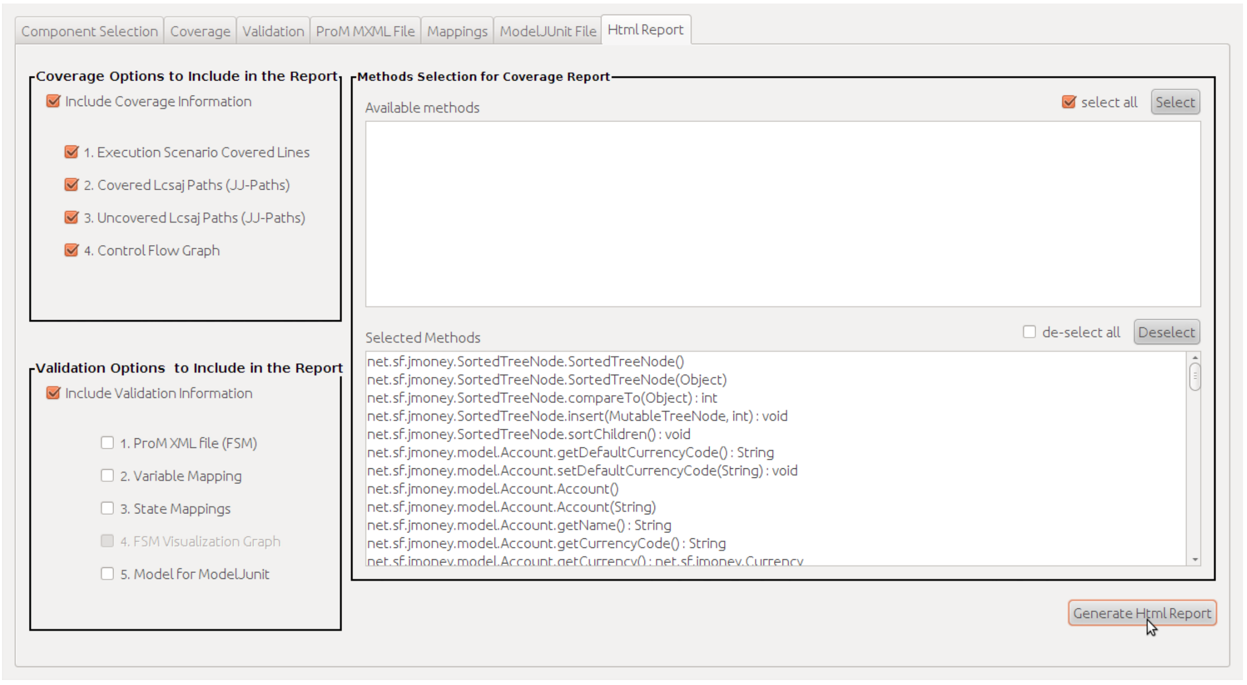
Figure 27 – Dynamic Analysis: ModelJUnit generated code

9. In the eighth tab (**HTML Report**)

Reuse engineer has the option of generating an HTML report with all the information provided by the Dynamic Analysis Process.

When this option is first visited, all available options are automatically selected to be generated. The Reuse Engineer however maintains the option of customizing the report depending on its needs.

CASE STUDY: In the following image, the Reuse Engineer has selected to generate a report containing coverage information for all the available classes of the JMoney Reuse Project.



Coverage Options to Include in the Report

[✓] Include Coverage Information

[✓] 1. Execution Scenario Covered Lines

[✓] 2. Covered Lcsaj Paths (JJ-Paths)

[✓] 3. Uncovered Lcsaj Paths (JJ-Paths)

[✓] 4. Control Flow Graph

Validation Options to Include in the Report

[✓] Include Validation Information

[ ] 1. ProM XML file (FSM)

[ ] 2. Variable Mapping

[ ] 3. State Mappings

[ ] 4. FSM Visualization Graph

[ ] 5. Model For ModelJunit

Methods Selection for Coverage Report

Available methods [✓] select all

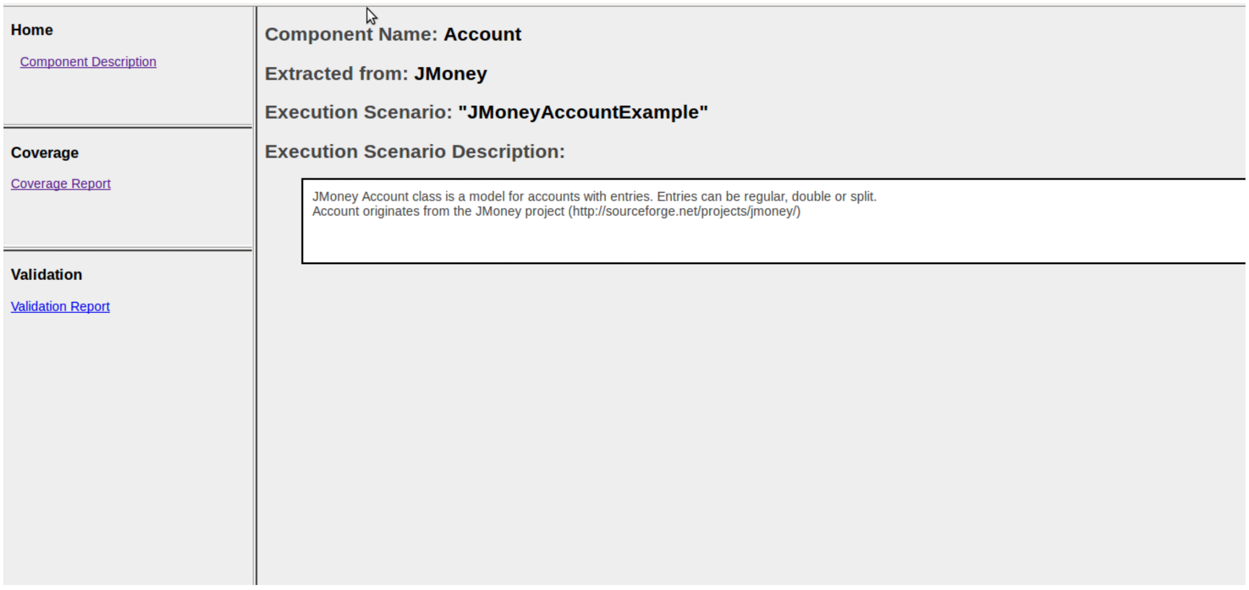
Selected Methods [ ] de-select all

```
net.sf.jmoney.SortedTreeNode.SortedTreeNode()
net.sf.jmoney.SortedTreeNode.SortedTreeNode(Object)
net.sf.jmoney.SortedTreeNode.compareTo(Object) : int
net.sf.jmoney.SortedTreeNode.insert(MutableTreeNode, int) : void
net.sf.jmoney.SortedTreeNode.sortChildren() : void
net.sf.jmoney.model.Account.getDefaultCurrencyCode() : String
net.sf.jmoney.model.Account.setDefaultCurrencyCode(String) : void
net.sf.jmoney.model.Account.Account()
net.sf.jmoney.model.Account.Account(String)
net.sf.jmoney.model.Account.getName() : String
net.sf.jmoney.model.Account.getCurrencyCode() : String
net.sf.jmoney.model.Account.getCurrency() : net.sf.jmoney.Currency
```

Figure 28 – Customized Dynamic Analysis HTML report

The generated report is located in the “tests” of the generated component under a directory entitled “doc”. It presents the aforementioned information as a set of HTML pages.

*CASE STUDY: In the following images, you can see the Dynamic Analysis HTML report generated using the options of “*Figure 28 – Customized Dynamic Analysis HTML report*”*



Component Name: Account

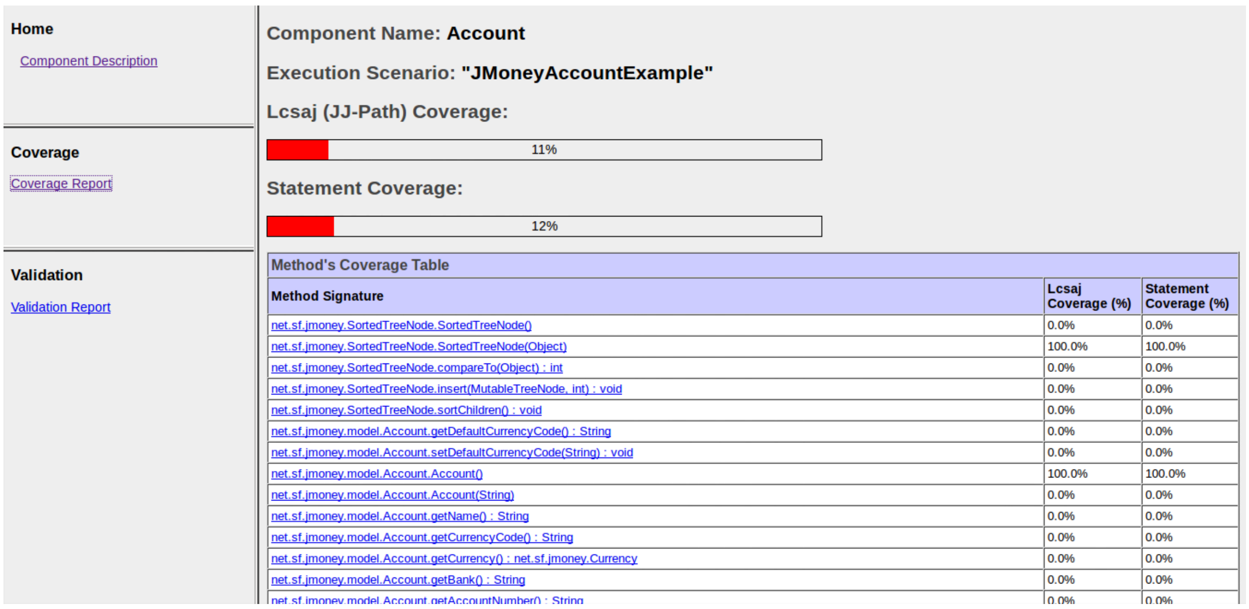
Extracted from: JMoney

Execution Scenario: "JMoneyAccountExample"

Execution Scenario Description:

JMoney Account class is a model for accounts with entries. Entries can be regular, double or split.
Account originates from the JMoney project (http://sourceforge.net/projects/jmoney/)

Figure 29 – Dynamic Analysis HTML Report: Component Description



Component Name: Account

Execution Scenario: "JMoneyAccountExample"

Lcsaj (JJ-Path) Coverage:

11%

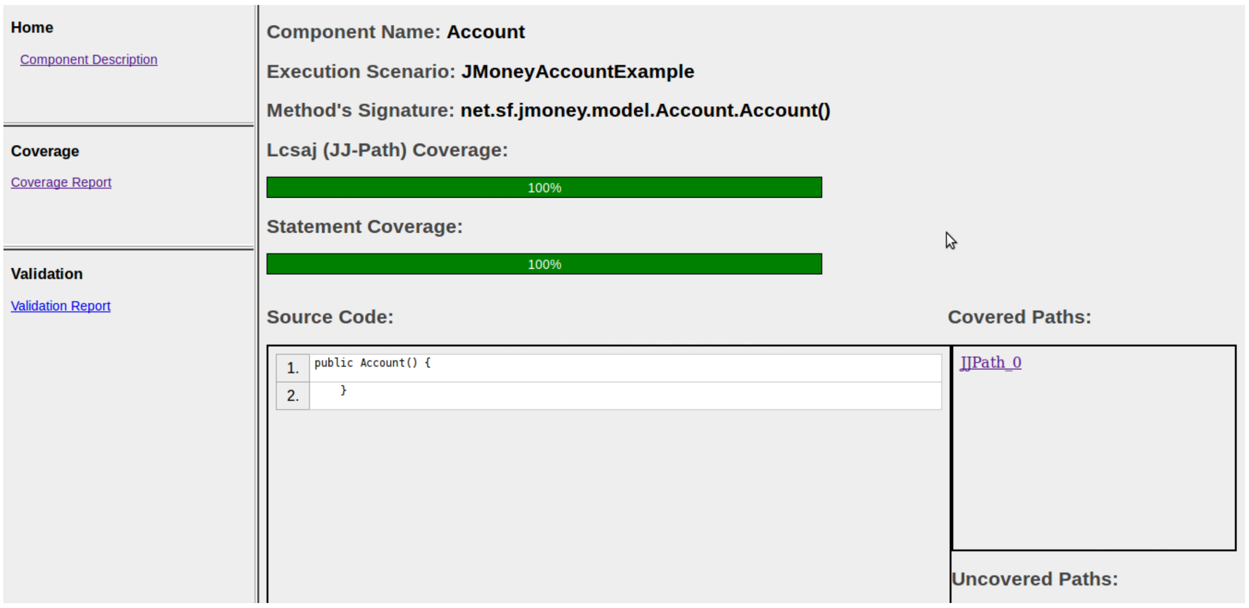
Statement Coverage:

12%

Method's Coverage Table

Method Signature	Lcsaj Coverage (%)	Statement Coverage (%)
net.sf.jmoney.SortedTreeNode.SortedTreeNode()	0.0%	0.0%
net.sf.jmoney.SortedTreeNode.SortedTreeNode(Object)	100.0%	100.0%
net.sf.jmoney.SortedTreeNode.compareTo(Object) : int	0.0%	0.0%
net.sf.jmoney.SortedTreeNode.insert(MutableTreeNode, int) : void	0.0%	0.0%
net.sf.jmoney.SortedTreeNode.sortChildren() : void	0.0%	0.0%
net.sf.jmoney.model.Account.getDefaultCurrencyCode() : String	0.0%	0.0%
net.sf.jmoney.model.Account.setDefaultCurrencyCode(String) : void	0.0%	0.0%
net.sf.jmoney.model.Account.Account()	100.0%	100.0%
net.sf.jmoney.model.Account.Account(String)	0.0%	0.0%
net.sf.jmoney.model.Account.getName() : String	0.0%	0.0%
net.sf.jmoney.model.Account.getCurrencyCode() : String	0.0%	0.0%
net.sf.jmoney.model.Account.getCurrency() : net.sf.jmoney.Currency	0.0%	0.0%
net.sf.jmoney.model.Account.getBank() : String	0.0%	0.0%
net.sf.jmoney.model.Account.getAccountNumber() : String	0.0%	0.0%

Figure 30 - Dynamic Analysis HTML Report: Coverage Report (generic)



Component Name: Account

Execution Scenario: JMoneyAccountExample

Method's Signature: net.sf.jmoney.model.Account.Account()

Lcsaj (JJ-Path) Coverage:

100%

Statement Coverage:

100%

Source Code:

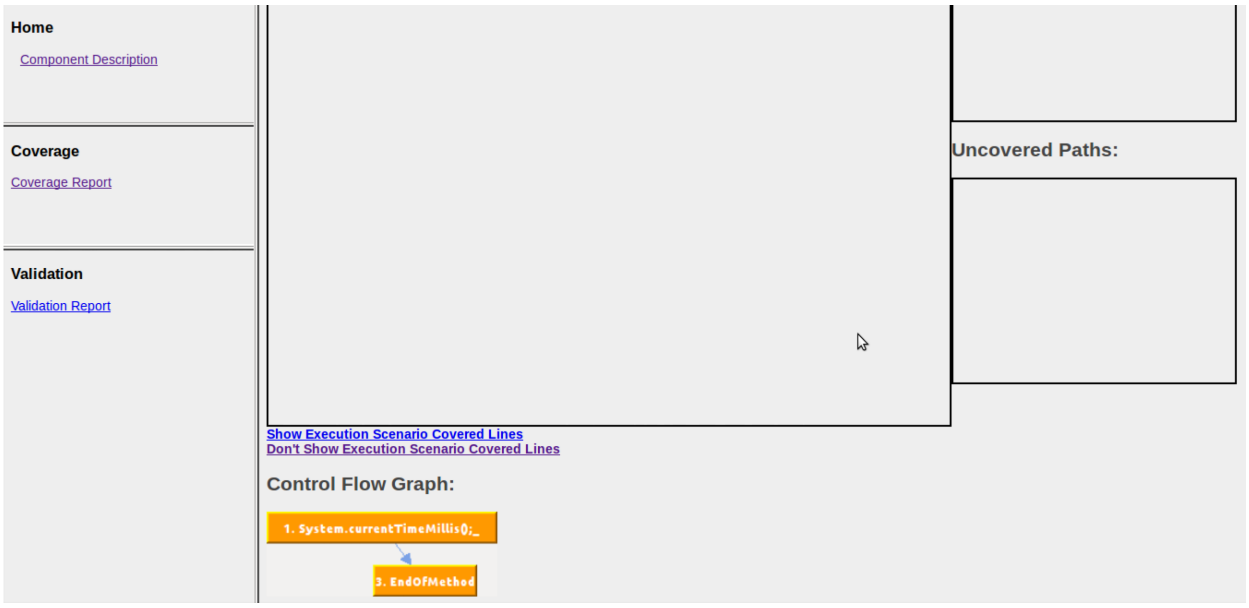
```
public Account() {
    }
```

Covered Paths:

JJPath_0

Uncovered Paths:

Figure 31 - Dynamic Analysis HTML Report: Coverage Report (method view pt. 1)



Uncovered Paths:

Show Execution Scenario Covered Lines

Don't Show Execution Scenario Covered Lines

Control Flow Graph:

Figure 32 - Dynamic Analysis HTML Report: Coverage Report (method view pt.2)

1.3.5.5 Performing History Analysis

The reuse engineer can select to import the history of the project development changes. Currently we only support Subversion repositories. The reuse engineer can use the svn log command with the --xml option to extract the development history and save the results in an XML file. Then we can use the history analysis menu option to import this file in the database.

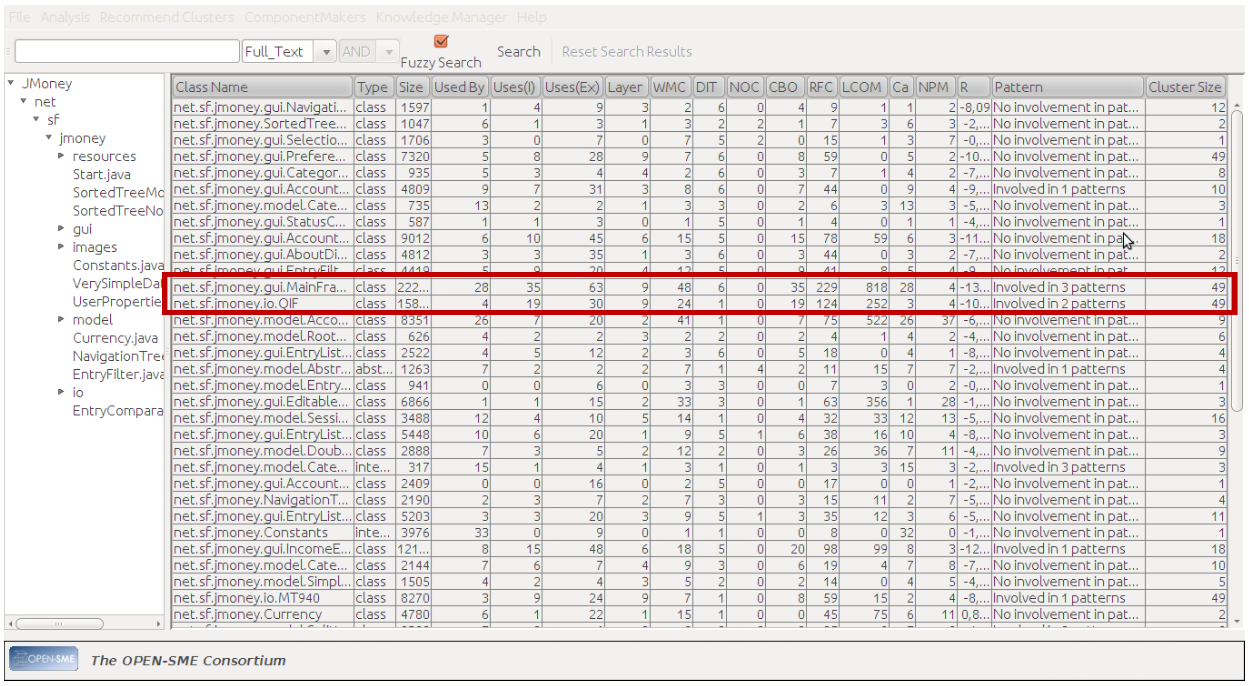
Notice that although we support importing these development history facts in COPE's database, currently we are not using this analysis for the recommendation of components, since initial research results were inconclusive. However we have maintained the feature of importing these facts for further research.

1.3.5.6 Performing Pattern Analysis

Pattern analysis is scanning the source code of the Reuse Project for patterns. Not only it identifies the type of the pattern but it is also able to determine the classes that participate on this specific pattern.

To perform Pattern Analysis for a Reuse Project in COPE:

1. Select **"Pattern Analysis"** from the **Analysis** menu.
2. In the dialog that appears click to the **Start** button.
3. When the generation is complete (the progress bar has reached 100%) you can close the process dialog using the "X" button.



Class Name	Type	Size	Used By	Uses(I)	Uses(Ex)	Layer	WMC	DIT	NOC	CBO	RFC	LCOM	Ca	NPM	R	Pattern	Cluster Size
net.sf.jmoney.gui.Navigati...	class	1597	1	4	9	3	2	6	0	4	9	1	1	2	-8,09	No involvement in pat...	12
net.sf.jmoney.SortedTree...	class	1047	6	1	3	1	3	2	2	1	7	3	6	3	-2,...	No involvement in pat...	2
net.sf.jmoney.gui.Selectio...	class	1706	3	0	7	0	7	5	2	0	15	1	3	7	-0,...	No involvement in pat...	1
net.sf.jmoney.gui.Prefere...	class	7320	5	8	28	9	7	6	0	8	59	0	5	2	-10...	No involvement in pat...	49
net.sf.jmoney.gui.Categor...	class	935	5	3	4	4	2	6	0	3	7	1	4	2	-7,...	No involvement in pat...	8
net.sf.jmoney.gui.Account...	class	4809	9	7	31	3	8	6	0	7	44	0	9	4	-9,...	Involved in 1 patterns	10
net.sf.jmoney.model.Cate...	class	735	13	2	2	1	3	3	0	2	6	3	13	3	-5,...	No involvement in pat...	3
net.sf.jmoney.gui.StatusC...	class	587	1	1	3	0	1	5	0	1	4	0	1	1	-4,...	No involvement in pat...	1
net.sf.jmoney.gui.Account...	class	9012	6	10	45	6	15	5	0	15	78	59	6	3	-11...	No involvement in pat...	18
net.sf.jmoney.gui.AboutDi...	class	4812	3	3	35	1	3	6	0	3	44	0	3	2	-7,...	No involvement in pat...	2
net.sf.jmoney.gui.EntryFilt...	class	4419	5	9	20	4	12	5	0	9	41	8	5	4	-9...	No involvement in pat...	12
net.sf.jmoney.gui.MainFra...	class	222...	28	35	63	9	48	6	0	35	229	818	28	4	-13...	Involved in 3 patterns	49
net.sf.jmoney.io.QIF	class	158...	4	19	30	9	24	1	0	19	124	252	3	4	-10...	Involved in 2 patterns	49
net.sf.jmoney.model.Acco...	class	8351	26	7	20	2	41	1	0	7	75	522	26	37	-6,...	No involvement in pat...	9
net.sf.jmoney.model.Root...	class	626	4	2	2	3	2	2	0	2	4	1	4	2	-4,...	No involvement in pat...	6
net.sf.jmoney.gui.EntryList...	class	2522	4	5	12	2	3	6	0	5	18	0	4	1	-8,...	No involvement in pat...	4
net.sf.jmoney.model.Abstr...	abst...	1263	7	2	2	2	7	1	4	2	11	15	7	7	-2,...	Involved in 1 patterns	4
net.sf.jmoney.model.Entry...	class	941	0	0	6	0	3	3	0	0	7	3	0	2	-0,...	No involvement in pat...	1
net.sf.jmoney.gui.Editable...	class	6866	1	1	15	2	33	3	0	1	63	356	1	28	-1,...	No involvement in pat...	3
net.sf.jmoney.model.Sessi...	class	3488	12	4	10	5	14	1	0	4	32	33	12	13	-5,...	No involvement in pat...	16
net.sf.jmoney.gui.EntryList...	class	5448	10	6	20	1	9	5	1	6	38	16	10	4	-8,...	No involvement in pat...	3
net.sf.jmoney.model.Doub...	class	2888	7	3	5	2	12	2	0	3	26	36	7	11	-4,...	No involvement in pat...	9
net.sf.jmoney.model.Cate...	inte...	317	15	1	4	1	3	1	0	1	3	3	15	3	-2,...	Involved in 3 patterns	3
net.sf.jmoney.gui.Account...	class	2409	0	0	16	0	2	5	0	0	17	0	0	1	-2,...	No involvement in pat...	1
net.sf.jmoney.NavigationT...	class	2190	2	3	7	2	7	3	0	3	15	11	2	7	-5,...	No involvement in pat...	4
net.sf.jmoney.gui.EntryList...	class	5203	3	3	20	3	9	5	1	3	35	12	3	6	-5,...	No involvement in pat...	11
net.sf.jmoney.Constants	inte...	3976	33	0	9	0	1	1	0	0	8	0	32	0	-1,...	No involvement in pat...	1
net.sf.jmoney.gui.IncomeE...	class	121...	8	15	48	6	18	5	0	20	98	99	8	3	-12...	Involved in 1 patterns	18
net.sf.jmoney.model.Cate...	class	2144	7	6	7	4	9	3	0	6	19	4	7	8	-7,...	No involvement in pat...	10
net.sf.jmoney.model.Simpl...	class	1505	4	2	4	3	5	2	0	2	14	0	4	5	-4,...	No involvement in pat...	5
net.sf.jmoney.io.MT940	class	8270	3	9	24	9	7	1	0	8	59	15	2	4	-8,...	Involved in 1 patterns	49
net.sf.jmoney.Currency	class	4780	6	1	22	1	15	1	0	0	45	75	6	11	0,8...	No involvement in pat...	2

Figure 33- COPE's main window after Pattern Analysis

*<u>CASE STUDY</u>: As you can see in "*Figure 33- COPE's main window after Pattern Analysis*" after performing Pattern Analysis for the Reuse Project JMoney in the main window of COPE, some classes are identified as participants in patterns (the ones in the red border are indicative).*

1.3.6 Cluster Recommendation

Using the Cluster Recommendation options, the Reuse Engineer can easily come up with some recommendations of class clusters that could form possible components. For the time being COPE provides two methods for recommending such class clusters:

- **Dependencies Recommender:** uses a genetic algorithm in order to form class clusters using the source code of the Reuse Project.
- **Pattern Recommender:** forms a cluster for each pattern detected in the source code of the Reuse Project.

For component extraction another very useful approach is to select a class and extract a component based on this class. The resulting component will have the interface of the public methods of the class and will include all the required classes for the reuse of this class. The reuse engineer can select this class based on the metrics that are presented in the main window, and especially the Cluster Size, Layer and R (reusability index) metrics. Classes which are lower in the layered digraph of the project (have small layer value), have few dependencies (have small Cluster Size) and have larger R value (are more reusable) are good candidates for reusable components. The reuse engineer can extract components by right-clicking any class from the main window that seems promising based on the aforementioned metrics and extract a component for this class. This same process is also available from the menu item "Component Makers → Dependency Maker" which will be discussed later in Section "1.3.7". Strictly speaking however this is not a recommender although it is an effective way to extract reusable components, and this is why is not provided as a separate option from the "Recommend Clusters" menu.

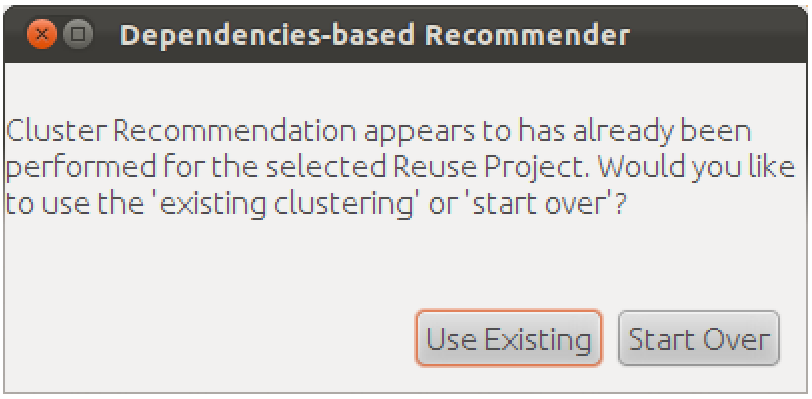
<u>NOTE THAT</u>:

- Since Dependencies Recommender uses a genetic algorithm, each time it runs for the same Reuse Project can lead to different clusters.
- In order for the Pattern Recommender to run, pattern analysis should have been performed first.

1.3.6.1 RUNNING THE DEPENDENCIES RECOMMENDER

To run the Dependencies Recommender for a Reuse Project in COPE:

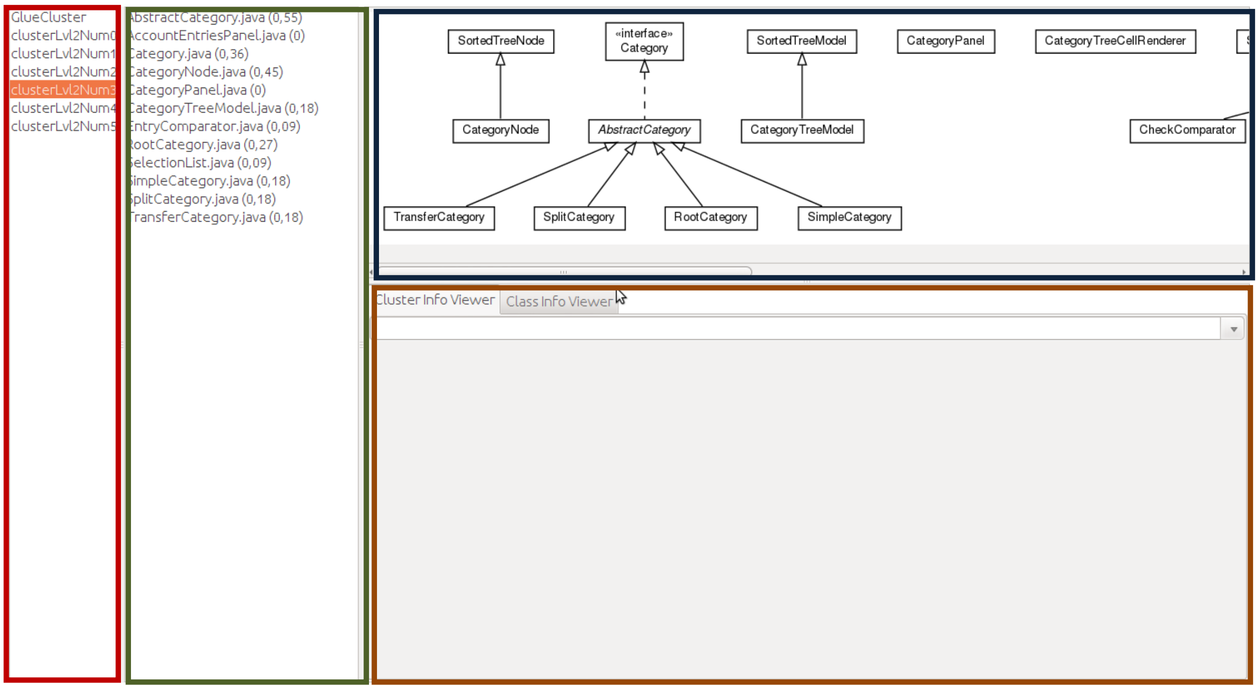
1. Select **"Dependencies Recommender"** from the **Recommend Clusters** menu.
2. If Dependencies Recommender has already run once, the following dialog offers the Reuse Engineer the option of choosing between keeping the current results or run Dependencies Recommender from scratch.


Dependencies-based Recommender

Cluster Recommendation appears to has already been performed for the selected Reuse Project. Would you like to use the 'existing clustering' or 'start over'?

Figure 34 – Introductory dialog to Dependencies Recommender

3. If no errors occur COPE provides the Reuse Engineer with the following dialog


GlueCluster
clusterLvl2Num0
clusterLvl2Num1
clusterLvl2Num2
clusterLvl2Num3
clusterLvl2Num4
clusterLvl2Num5

bstractCategory.java (0,55)
AccountEntriesPanel.java (0)
Category.java (0,36)
CategoryNode.java (0,45)
CategoryPanel.java (0)
CategoryTreeModel.java (0,18)
EntryComparator.java (0,09)
RootCategory.java (0,27)
SelectionList.java (0,09)
SimpleCategory.java (0,18)
SplitCategory.java (0,18)
TransferCategory.java (0,18)

Figure 35 – Dependencies Recommender main dialog

- In the **red area**, the Reuse Engineer can find all the recommended clusters proposed by the Dependencies Recommender.
- By selecting a cluster, the class files of this cluster appear in the **green area**. The Reuse Engineer can move classes from one cluster to another simply by right-clicking to one of the classes located to the green area and selecting the cluster to which he wishes the selected class to be moved. Finally, classes can also be moved in an artificial cluster, the "GlueCluster", which serves as a pool for classes that do not fit to any of the proposed clusters. For each class a **specificity metric** of the class for the cluster is provided in parentheses. This value is the ratio of other classes in the cluster depending on the class divided by the total number of classes in the cluster. Specificity can be used to assist a reuse engineer in deciding if a class belongs to the cluster that it was classified to, or that it should be moved to a different cluster. Right-clicking in the cluster classes' the reuse engineer can choose to sort the classes based on their specificity to quickly view the most and less specific classes in a cluster.
- Apart from the class list, selection of the cluster also results in its UML diagram's generation. This diagram is located in the **blue area** of the dependencies recommender dialog. UML graphs are generated using UMLGraph[8].
- Finally, the **brown area** provides extra information about the keywords derived from Latent Semantic Analysis and Tag Cloud analysis (see Source File Indexing). This information is available in Cluster and Class level. This information can provide the reuse engineer with hints on the purpose of the cluster and the purpose of each class in the cluster.

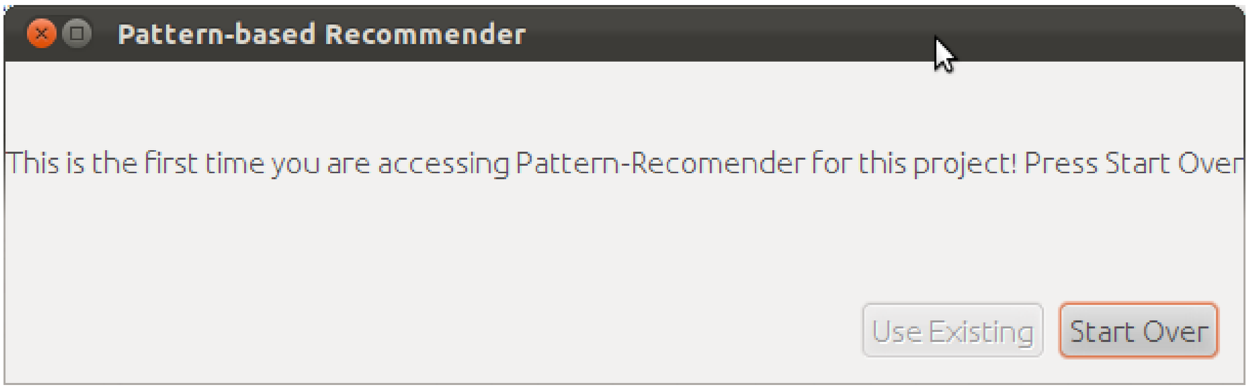
1.3.6.2 Running the Pattern Recommender

To run the Pattern Recommender for a Reuse Project in COPE:

1. Select **"Pattern Recommender"** from the **Recommend Clusters** menu.

[8] http://www.umlgraph.org/

2. If Pattern Recommender has already run once, the following dialog offers the Reuse Engineer the option of choosing between keeping the current results or run Pattern Recommender from scratch.



This is the first time you are accessing Pattern-Recomender for this project! Press Start Over

Figure 36 – Introductory dialog to Pattern Recommender

3. If no errors occur COPE provides the Reuse Engineer with the following dialog

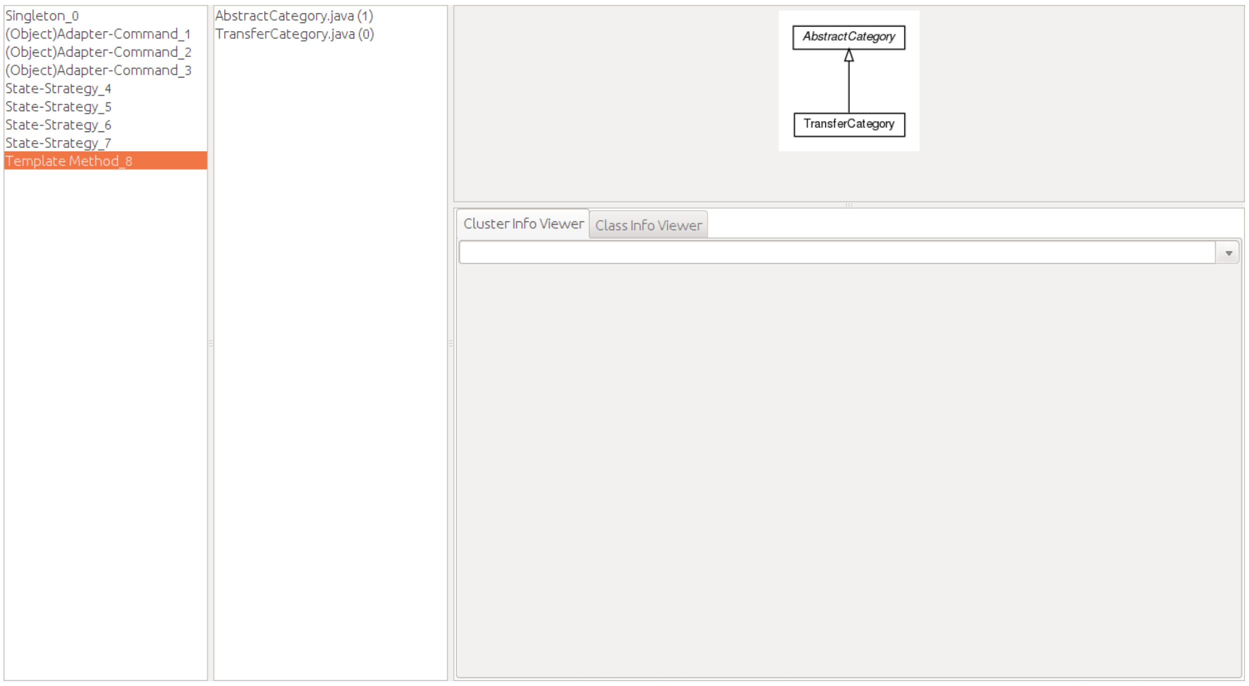


Figure 37 – Pattern Recommender Main Dialog

This dialog is identical to the Dependencies Maker’s. The only difference is that in Patter Recommender dialog, instead of clusters the Reuse Engineer sees all patterns of the Reuse Project, detected during the Pattern Analysis phase (see Pattern Analysis).

1.3.7 Component Making

Based on the analyses and recommendations carried out earlier the Reuse Engineer can now produce independent software components and then place those components in the repository using the ‘Knowledge Manager’ feature. In this section we discuss the different component makers available in COPE. These component makers are available from the “Component Makers” menu. Currently there are four different component makers:

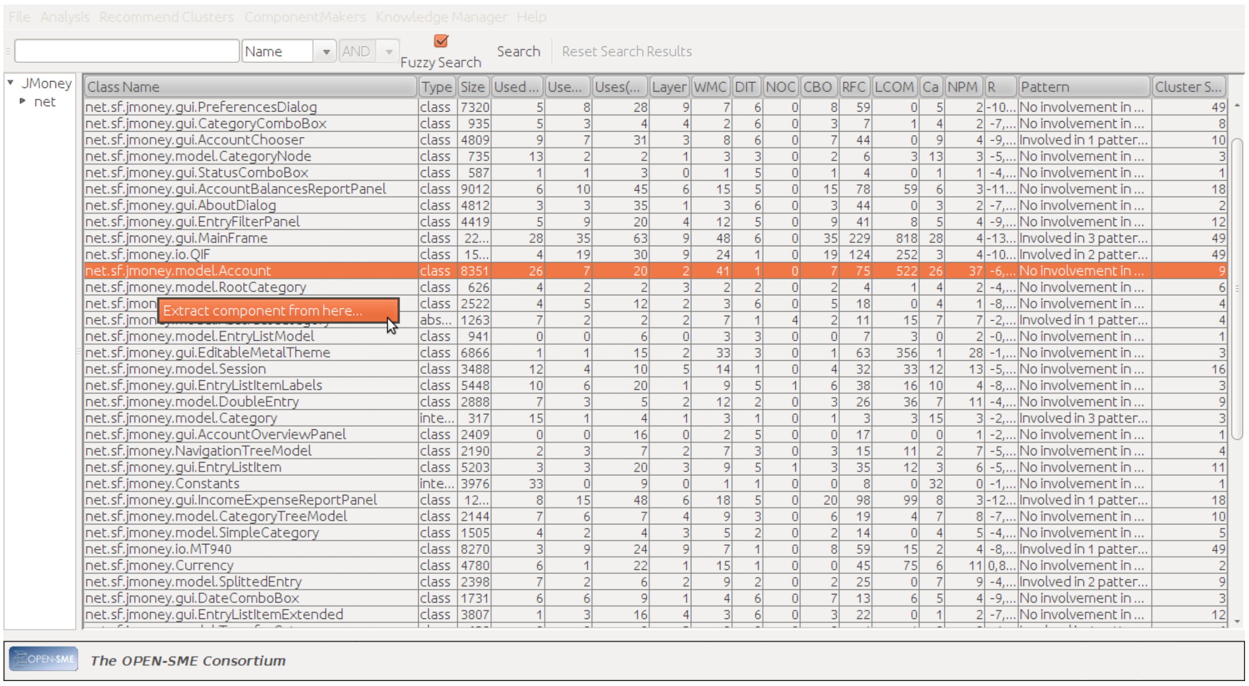
1. **Interface Maker:** This component maker uses as input the clusters produced by the "Dependencies Recommender" (see Section 1.3.6.1). The Reuse Engineer can select one of the clusters produced there and extract a component for this class cluster.

2. **Dependency Maker:** This component maker presents all the classes of the project along with their reusability assessment. The reuse engineer can select a class and extract a component providing the functionality of the selected class. The extracted component will contain the class and its dependencies. Notice that the same functionality is also available from COPE's main window by right-clicking a class and selecting "Extract component from here" (see Figure 38 – Extracting component from classes selected in the main window of COPE).

3. **Adapter Pattern Maker:** This component maker presents the clusters produced by the 'Pattern Recommender' (see Section 1.3.6.2) and displays clusters involved in Adapter pattern instances. The reuse engineer can select a cluster and extract a component for this cluster. The component will have the interface of the adapter.

4. **Proxy Pattern Maker:** This component maker presents again the clusters produced by the "Pattern Recommender" but this time it displays only clusters involved in Proxy pattern instances. The reuse engineer can select a cluster and extract a component for this cluster. The component will have the interface of the proxy.

The dialog displayed in all cases is the same and the results of this process are the same (an independent component). Therefore we will explain here only the second component maker (the Dependencies Maker). The process for the other makers is exactly the same with the caveat the extracted components have the semantics mentioned in the previous list.



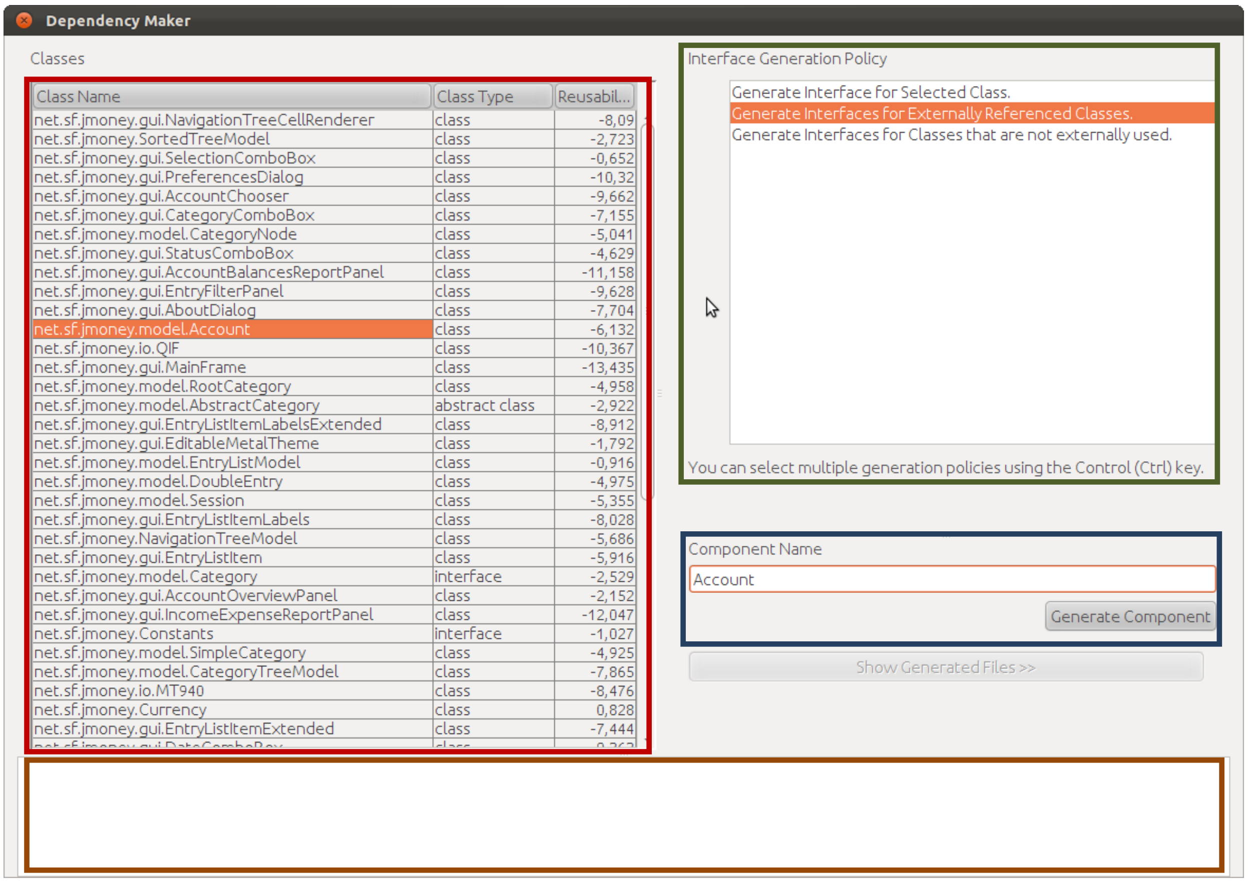
Class Name	Type	Size	Used ...	Use...	Uses(...	Layer	WMC	DIT	NOC	CBO	RFC	LCOM	Ca	NPM	R	Pattern	Cluster S...
net.sf.jmoney.gui.PreferencesDialog	class	7320	5	8	28	9	7	6	0	8	59	0	5	2	-10...	No involvement in ...	49
net.sf.jmoney.gui.CategoryComboBox	class	935	5	3	4	4	2	6	0	3	7	1	4	2	-7,...	No involvement in ...	8
net.sf.jmoney.gui.AccountChooser	class	4809	9	7	31	3	8	6	0	7	44	0	9	4	-9,...	Involved in 1 patter...	10
net.sf.jmoney.model.CategoryNode	class	735	13	2	2	1	3	3	0	2	6	3	13	3	-5,...	No involvement in ...	3
net.sf.jmoney.gui.StatusComboBox	class	587	1	1	3	0	1	5	0	1	4	0	1	1	-4,...	No involvement in ...	1
net.sf.jmoney.gui.AccountBalancesReportPanel	class	9012	6	10	45	6	15	5	0	15	78	59	6	3	-11...	No involvement in ...	18
net.sf.jmoney.gui.AboutDialog	class	4812	3	3	35	1	3	6	0	3	44	0	3	2	-7,...	No involvement in ...	2
net.sf.jmoney.gui.EntryFilterPanel	class	4419	5	9	20	4	12	5	0	9	41	8	5	4	-9,...	No involvement in ...	12
net.sf.jmoney.gui.MainFrame	class	22...	28	35	63	9	48	6	0	35	229	818	28	4	-13...	Involved in 3 patter...	49
net.sf.jmoney.io.QIF	class	15...	4	19	30	9	24	1	0	19	124	252	3	4	-10...	Involved in 2 patter...	49
net.sf.jmoney.model.Account	class	8351	26	7	20	2	41	1	0	7	75	522	26	37	-6,...	No involvement in ...	9
net.sf.jmoney.model.RootCategory	class	626	4	2	2	3	2	2	0	2	4	1	4	2	-4,...	No involvement in ...	6
net.sf.jmon	class	2522	4	5	12	2	3	6	0	5	18	0	4	1	-8,...	No involvement in ...	4
net.sf.jmon	abs...	1263	7	2	2	2	7	1	4	2	11	15	7	7	-2,...	Involved in 1 patter...	4
net.sf.jmoney.model.EntryListModel	class	941	0	0	6	0	3	3	0	0	7	3	0	2	-0,...	No involvement in ...	1
net.sf.jmoney.gui.EditableMetalTheme	class	6866	1	1	15	2	33	3	0	1	63	356	1	28	-1,...	No involvement in ...	3
net.sf.jmoney.model.Session	class	3488	12	4	10	5	14	1	0	4	32	33	12	13	-5,...	No involvement in ...	16
net.sf.jmoney.gui.EntryListItemLabels	class	5448	10	6	20	1	9	5	1	6	38	16	10	4	-8,...	No involvement in ...	3
net.sf.jmoney.model.DoubleEntry	class	2888	7	3	5	2	12	2	0	3	26	36	7	11	-4,...	No involvement in ...	9
net.sf.jmoney.model.Category	inte...	317	15	1	4	1	3	1	0	1	3	3	15	3	-2,...	Involved in 3 patter...	3
net.sf.jmoney.gui.AccountOverviewPanel	class	2409	0	0	16	0	2	5	0	0	17	0	0	1	-2,...	No involvement in ...	1
net.sf.jmoney.NavigationTreeModel	class	2190	2	3	7	2	7	3	0	3	15	11	2	7	-5,...	No involvement in ...	4
net.sf.jmoney.gui.EntryListItem	class	5203	3	3	20	3	9	5	1	3	35	12	3	6	-5,...	No involvement in ...	11
net.sf.jmoney.Constants	inte...	3976	33	0	9	0	1	1	0	0	8	0	32	0	-1,...	No involvement in ...	1
net.sf.jmoney.gui.IncomeExpenseReportPanel	class	12...	8	15	48	6	18	5	0	20	98	99	8	3	-12...	Involved in 1 patter...	18
net.sf.jmoney.model.CategoryTreeModel	class	2144	7	6	7	4	9	3	0	6	19	4	7	8	-7,...	No involvement in ...	10
net.sf.jmoney.model.SimpleCategory	class	1505	4	2	4	3	5	2	0	2	14	0	4	5	-4,...	No involvement in ...	5
net.sf.jmoney.io.MT940	class	8270	3	9	24	9	7	1	0	8	59	15	2	4	-8,...	Involved in 1 patter...	49
net.sf.jmoney.Currency	class	4780	6	1	22	1	15	1	0	0	45	75	6	11	0,8...	No involvement in ...	2
net.sf.jmoney.model.SplittedEntry	class	2398	7	2	6	2	9	2	0	2	25	0	7	9	-4,...	Involved in 2 patter...	9
net.sf.jmoney.gui.DateComboBox	class	1731	6	6	9	1	4	6	0	7	13	6	5	4	-9,...	No involvement in ...	3
net.sf.jmoney.gui.EntryListItemExtended	class	3807	1	3	16	4	3	6	0	3	22	0	1	2	-7,...	No involvement in ...	12

Figure 38 – Extracting component from classes selected in the main window of COPE

1.3.7.1 RUNNING THE DEPENDENCIES MAKER

To run the Dependencies Maker for a Reuse Project in COPE:

1. Select **"Dependencies Maker"** from the **Component Makers** menu.



Class Name	Class Type	Reusabil...
net.sf.jmoney.gui.NavigationTreeCellRenderer	class	-8,09
net.sf.jmoney.SortedTreeModel	class	-2,723
net.sf.jmoney.gui.SelectionComboBox	class	-0,652
net.sf.jmoney.gui.PreferencesDialog	class	-10,32
net.sf.jmoney.gui.AccountChooser	class	-9,662
net.sf.jmoney.gui.CategoryComboBox	class	-7,155
net.sf.jmoney.model.CategoryNode	class	-5,041
net.sf.jmoney.gui.StatusComboBox	class	-4,629
net.sf.jmoney.gui.AccountBalancesReportPanel	class	-11,158
net.sf.jmoney.gui.EntryFilterPanel	class	-9,628
net.sf.jmoney.gui.AboutDialog	class	-7,704
net.sf.jmoney.model.Account	class	-6,132
net.sf.jmoney.io.QIF	class	-10,367
net.sf.jmoney.gui.MainFrame	class	-13,435
net.sf.jmoney.model.RootCategory	class	-4,958
net.sf.jmoney.model.AbstractCategory	abstract class	-2,922
net.sf.jmoney.gui.EntryListItemLabelsExtended	class	-8,912
net.sf.jmoney.gui.EditableMetalTheme	class	-1,792
net.sf.jmoney.model.EntryListModel	class	-0,916
net.sf.jmoney.model.DoubleEntry	class	-4,975
net.sf.jmoney.model.Session	class	-5,355
net.sf.jmoney.gui.EntryListItemLabels	class	-8,028
net.sf.jmoney.NavigationTreeModel	class	-5,686
net.sf.jmoney.gui.EntryListItem	class	-5,916
net.sf.jmoney.model.Category	interface	-2,529
net.sf.jmoney.gui.AccountOverviewPanel	class	-2,152
net.sf.jmoney.gui.IncomeExpenseReportPanel	class	-12,047
net.sf.jmoney.Constants	interface	-1,027
net.sf.jmoney.model.SimpleCategory	class	-4,925
net.sf.jmoney.model.CategoryTreeModel	class	-7,865
net.sf.jmoney.io.MT940	class	-8,476
net.sf.jmoney.Currency	class	0,828
net.sf.jmoney.gui.EntryListItemExtended	class	-7,444

Generate Interface for Selected Class.
Generate Interfaces for Externally Referenced Classes.
Generate Interfaces for Classes that are not externally used.

You can select multiple generation policies using the Control (Ctrl) key.

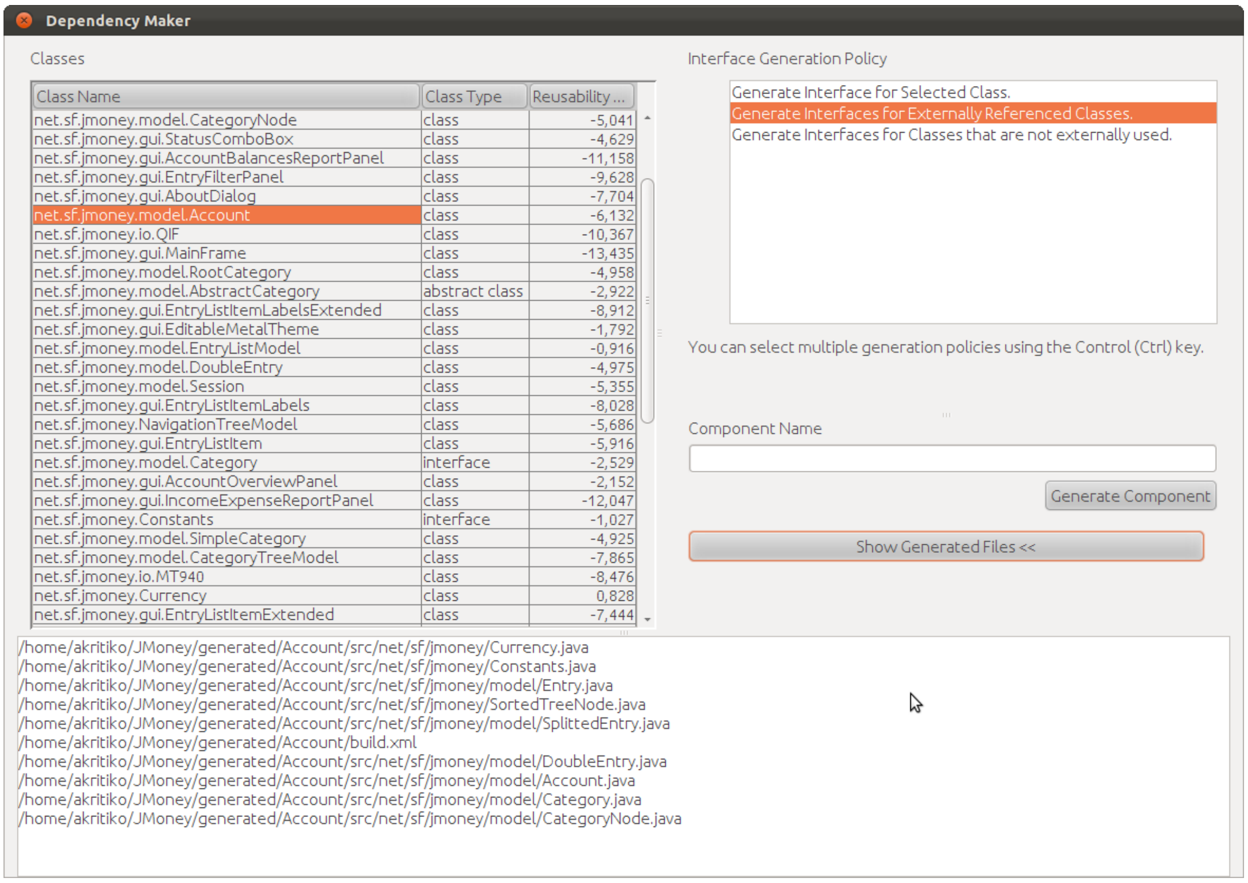
Component Name: Account

Figure 39 – Dependency Maker main dialog

- In the **red area** the Reuse Engineer can see all the classes (their fully qualified names) of the Reuse Project along with their "Class Type" and "Reusability Index" (for learning more about the Class Type and Reusability Index information please see Performing Static Analysis section)
- In the **green area** COPE provides the available "Interface Generation Policies" for the Reuse Engineer to choose. More specifically these are:
    - **Generate Interface for Selected Class:** with this option, COPE automatically generates an interface only for the class selected by the Reuse Engineer.
    - **Generate Interfaces for Externally Referenced Classes:** with this option, COPE automatically generates interfaces for all classes that are included in the generated component and also are referenced from project classes that are not included in the selected component.
    - **Generate Interfaces for Classes that are not used:** with this option, COPE automatically generates interfaces for all the classes that are members of the selected component the Reuse Engineer is about to create and they are not used by any class of the component.

<u>NOTE THAT</u>: The Reuse Engineer can select more than one of the aforementioned "Interface Generation Policies" using the CTRL button.

- In the **blue area** the Reuse Engineer gives a name to the component and he can generate the component by clicking the "Generate Component" button.
- Once the component has been successfully generated, its files are listed in the **brown area**.



Class Name	Class Type	Reusability ...
net.sf.jmoney.model.CategoryNode	class	-5,041
net.sf.jmoney.gui.StatusComboBox	class	-4,629
net.sf.jmoney.gui.AccountBalancesReportPanel	class	-11,158
net.sf.jmoney.gui.EntryFilterPanel	class	-9,628
net.sf.jmoney.gui.AboutDialog	class	-7,704
net.sf.jmoney.model.Account	class	-6,132
net.sf.jmoney.io.QIF	class	-10,367
net.sf.jmoney.gui.MainFrame	class	-13,435
net.sf.jmoney.model.RootCategory	class	-4,958
net.sf.jmoney.model.AbstractCategory	abstract class	-2,922
net.sf.jmoney.gui.EntryListItemLabelsExtended	class	-8,912
net.sf.jmoney.gui.EditableMetalTheme	class	-1,792
net.sf.jmoney.model.EntryListModel	class	-0,916
net.sf.jmoney.model.DoubleEntry	class	-4,975
net.sf.jmoney.model.Session	class	-5,355
net.sf.jmoney.gui.EntryListItemLabels	class	-8,028
net.sf.jmoney.NavigationTreeModel	class	-5,686
net.sf.jmoney.gui.EntryListItem	class	-5,916
net.sf.jmoney.model.Category	interface	-2,529
net.sf.jmoney.gui.AccountOverviewPanel	class	-2,152
net.sf.jmoney.gui.IncomeExpenseReportPanel	class	-12,047
net.sf.jmoney.Constants	interface	-1,027
net.sf.jmoney.model.SimpleCategory	class	-4,925
net.sf.jmoney.model.CategoryTreeModel	class	-7,865
net.sf.jmoney.io.MT940	class	-8,476
net.sf.jmoney.Currency	class	0,828
net.sf.jmoney.gui.EntryListItemExtended	class	-7,444

Generate Interface for Selected Class.
Generate Interfaces for Externally Referenced Classes.
Generate Interfaces for Classes that are not externally used.

You can select multiple generation policies using the Control (Ctrl) key.

```
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/Currency.java
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/Constants.java
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/model/Entry.java
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/SortedTreeNode.java
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/model/SplittedEntry.java
/home/akritiko/JMoney/generated/Account/build.xml
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/model/DoubleEntry.java
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/model/Account.java
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/model/Category.java
/home/akritiko/JMoney/generated/Account/src/net/sf/jmoney/model/CategoryNode.java
```

Figure 40 – Dependency Maker dialog after a component was successfully generated

Extracted components will be opened for further processing using an IDE (e.g. Eclipse or NetBeans). The reuse engineer will use the IDE to comprehend the component, create test cases for it or execution scenarios, discover further dependencies that are required which are not recoverable through static analysis (e.g. data dependencies). The component can then be tested dynamically (as described in Section 1.3.5.4) using the test cases or execution scenarios that were developed by the reuse engineer.

Next we will discuss the component extracted from the component makers and have a closer look to their structure before looking at importing these components in the COMPARE component repository with the help of the Knowledge Manager.

1.4 COPE'S COMPONENTS STRUCTURE

As we have already mentioned, COPE is a set of tools that assist the reuse engineer to produce autonomous, fully functional components. These components apart from the source code consist also of a variety of accompanying meta-data. All this information is being packaged as depicted in the following figure.

Notice that components are generated in the project's folder which resides in user's home directory (usually in /home/<username>). The project folder will have the name of the project provided by the reuse engineer during the project creation. In this folder there will be a folder called 'generated' in which different folders will be created (one for each generated component). The component folder will have the component name provided by the reuse engineer (see Figure 40) and will contain sub-folders as depicted in Figure 41. The project folder will look similar to the one depicted in Figure 42.

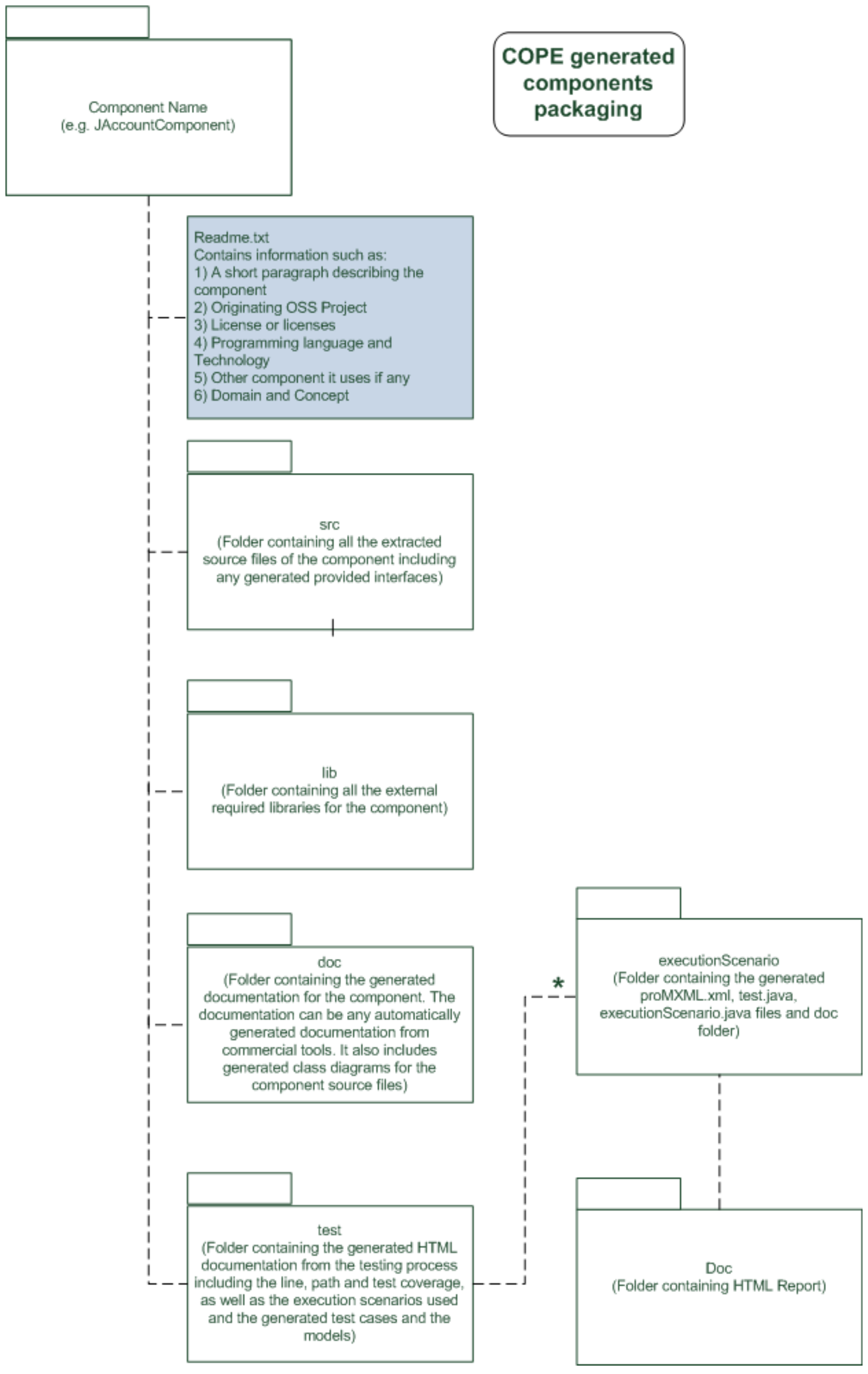

COPE generated components packaging
Component Name (e.g. JAccountComponent)
Readme.txt
Contains information such as:
1) A short paragraph describing the component
2) Originating OSS Project
3) License or licenses
4) Programming language and Technology
5) Other component it uses if any
6) Domain and Concept
src (Folder containing all the extracted source files of the component including any generated provided interfaces)
lib (Folder containing all the external required libraries for the component)
doc (Folder containing the generated documentation for the component. The documentation can be any automatically generated documentation from commercial tools. It also includes generated class diagrams for the component source files)
test (Folder containing the generated HTML documentation from the testing process including the line, path and test coverage, as well as the execution scenarios used and the generated test cases and the models)
*
executionScenario (Folder containing the generated proMXML.xml, test.java, executionScenario.java files and doc folder)
Doc (Folder containing HTML Report)

Figure 41: COPE generated components' packaging

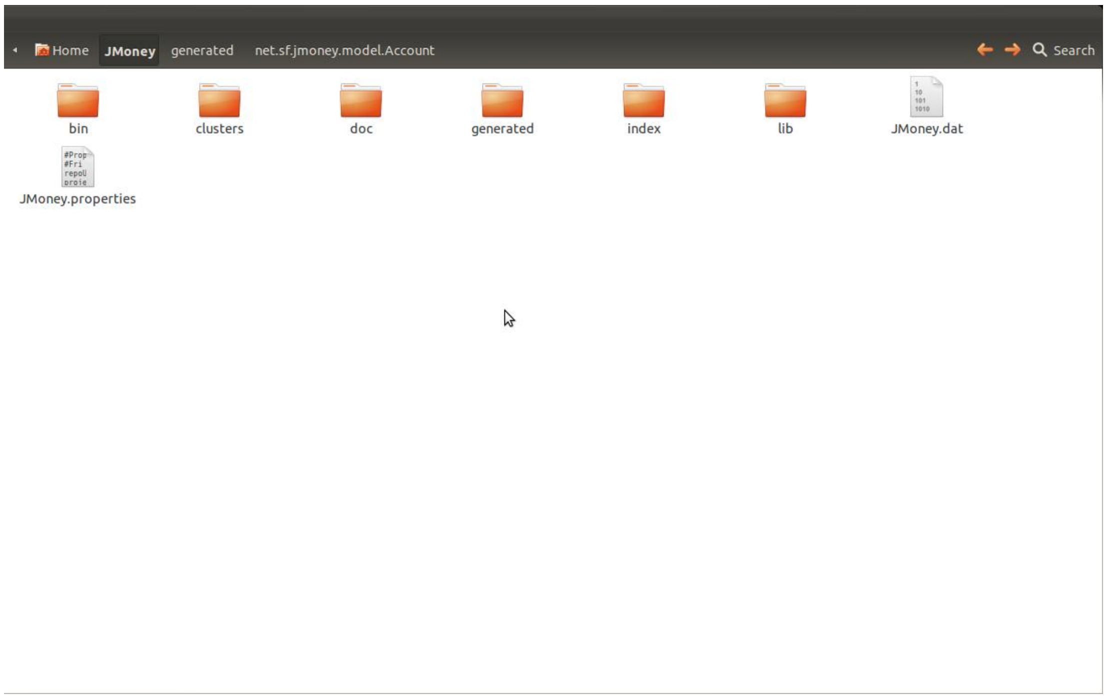


Figure 42: Project Folder

1.4.1 Root Directory (Component's Name)

The root folder has the name of the generated component and includes all the available information for this specific component.

1.4.2 README.TXT

A summary for the generated component.

- A short description of the functionality of the component
- The name of the Free/Libre Open Source Project from which the component was extracted
- The type of license (or licenses) under which the source code of the component was published
- The programming language of its source code
- Information about other technologies used by the component (if any)
- Information about other components used by the component (if any)
- The domain concept of the component

1.4.3 Source Directory (src/)

The source directory contains the source code of the extracted component. It also contains all the provided interfaces generated automatically by COPE.

1.4.4 Libraries Directory (lib/)

The libraries directory contains all the necessary external libraries in order for the component to be runnable.

1.4.5 DOCUMENTATION DIRECTORY (DOC/)

The documentation of the source code and UML class diagrams for the class files of the component. Both the documentation and the UML diagrams can be generated using appropriate free or commercial software.

1.4.6 TEST DIRECTORY (TEST/)

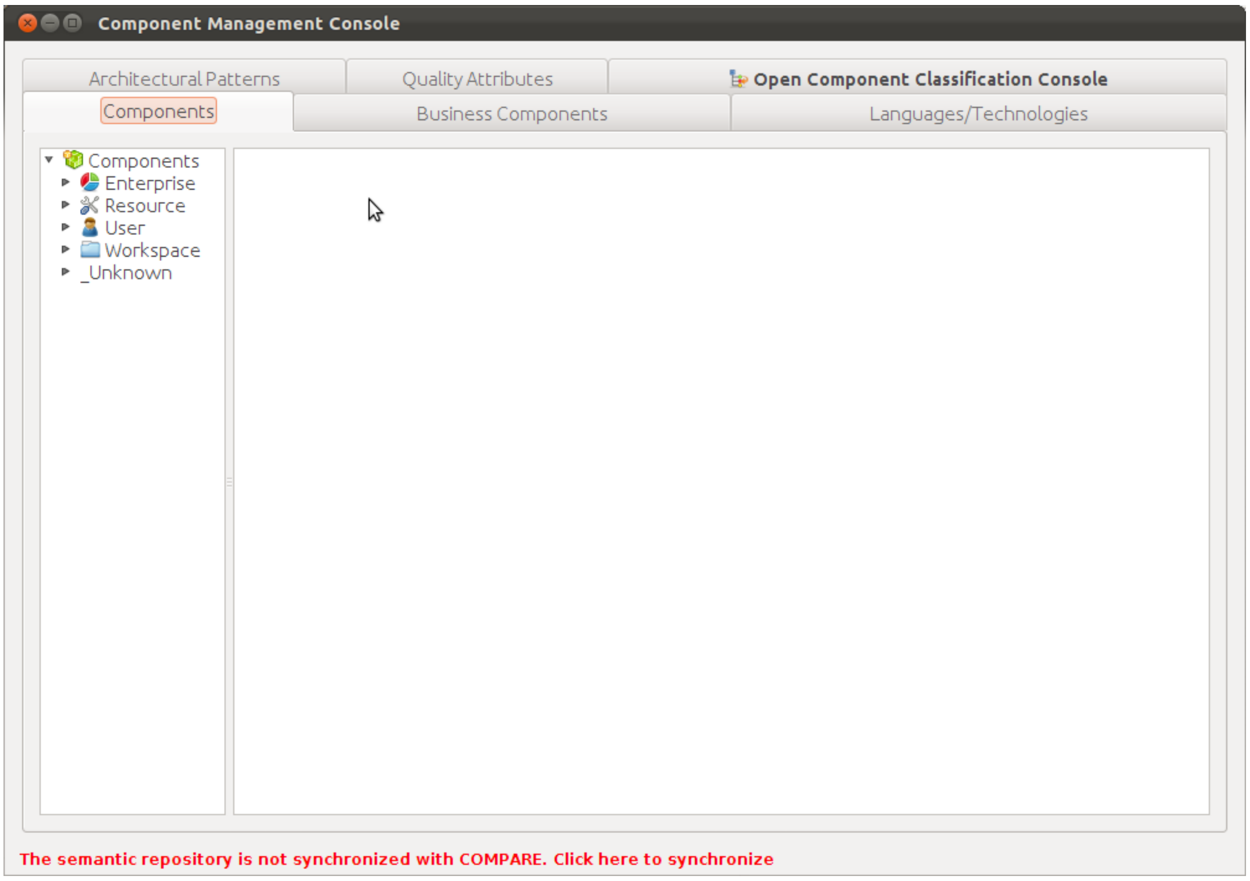
Test material for the extracted component. Contains:

- Execution Scenario (directory)
    - Line, Path and Test coverage information
    - Execution scenarios used by the Reuse Engineer for the specific component
    - Generated test cases and possible ModelJUnit models
- Generated documentation (in form of HTML) including the results of the testing process (directory)

1.5 THE KNOWLEDGE MANAGER

The Knowledge Manager provides the Reuse Engineer with a way of providing meta-information for the components s/he generates. Moreover s/he can classify those components to a specific domain and concept.

Moreover, the Knowledge Manager serves as an intermediate between the COPE platform and the component repository of COMPARE.



The semantic repository is not synchronized with COMPARE. Click here to synchronize

Figure 43 – Knowledge Manager: Main dialog

Components (tab): in this tab the Reuse Engineer can navigate through all the available generated components of COPE using the menu to the left. As you see in "Figure 43 – Knowledge Manager: Main dialog" there are some predefined categories (tiers) for the generated components [3]. More specifically:

- <u>Enterprise</u>: These components represent enterprise-level concepts. For example an Account component falls into this category, because it represents a business concept of the Accounting business domain.
- <u>Resource</u>: These components represent more low-level infrastructure software components. For example a DatabaseUtil component providing functions for storing, retrieving etc. data from a database would fall into this category. Notice that an Enterprise component may contain internally a resource component (e.g. an Account component would include internally a component for accessing, storing, removing etc. accounts from a database). Usually when these lower-layer components are encapsulated by an enterprise component their interface is hidden from external reusers, however in some cases they may be useful by themselves (e.g. a logging component, or an email component).
- <u>User</u>: These components represent user interface level components (for example an AccountCreationDialog component could be useful for creating accounts from the user interface)
- <u>Workspace</u>: These components represent workflow or session handling software components. For example a reuser could reuse a component that handles the creation of an Account by copying data from another Account. The component that handles the transaction of moving the information from one Account component to the other, could be a Workspace component.

- Unknown: This category contains the components that have been generated using COPE but were not associated with one of the aforementioned tiers by the Reuse Engineer.

Languages / Technologies (tab): in this tab the Reuse Engineer can add or remove programming languages (e.g. Java, C, C++, etc.) and technologies (e.g. J2SE, J2ME, etc.) relative to the components he works with.

Open Component Classification Console (tab): this tab helps the reuse engineer to classify his components with the use of MetaModel, domains and concepts. More specifically:

- A MetaModel group is a category of Meta-Models associated with an application domain (e.g. E-Business domain could represent a MetaModel group).
- A MetaModel is a model representing concepts from the business domain and their associations (e.g. Accounting MetaModel could be a MetaModel of the E-Business domain). Such a meta-model can be reused if available or constructed as the reuse engineer discovers concepts of application domains as s/he reverse engineers existing OSS systems.
- A domain is a specific area or market segment for which applications are being developed (e.g. Accounting domain).
- A concept is a specific entity or activity relating to an application domain (e.g. Account, Business Transaction, Shopping Cart etc.)

NOTE THAT: The data provided at the component classification console is optional and is expected to start taking shape as the reuse engineers are using the COPE platform to create components. It is estimated that at least 3 systems are necessary as input to a domain engineering process. To avoid the antipattern of domain analysis paralysis [4], COPE does not require that reuse engineers should build application domains MetaModels before starting extracting and providing components for reuse. Instead components can be reused and the domain MetaModel along with its concepts can be formed as the reuse organizations become more mature and systematic in their reuse processes. In other words, we anticipate that the usage of COPE will provide a gradual and iterative path towards systematic reuse.

1.5.1 CHARACTERIZING A GENERATED COMPONENT

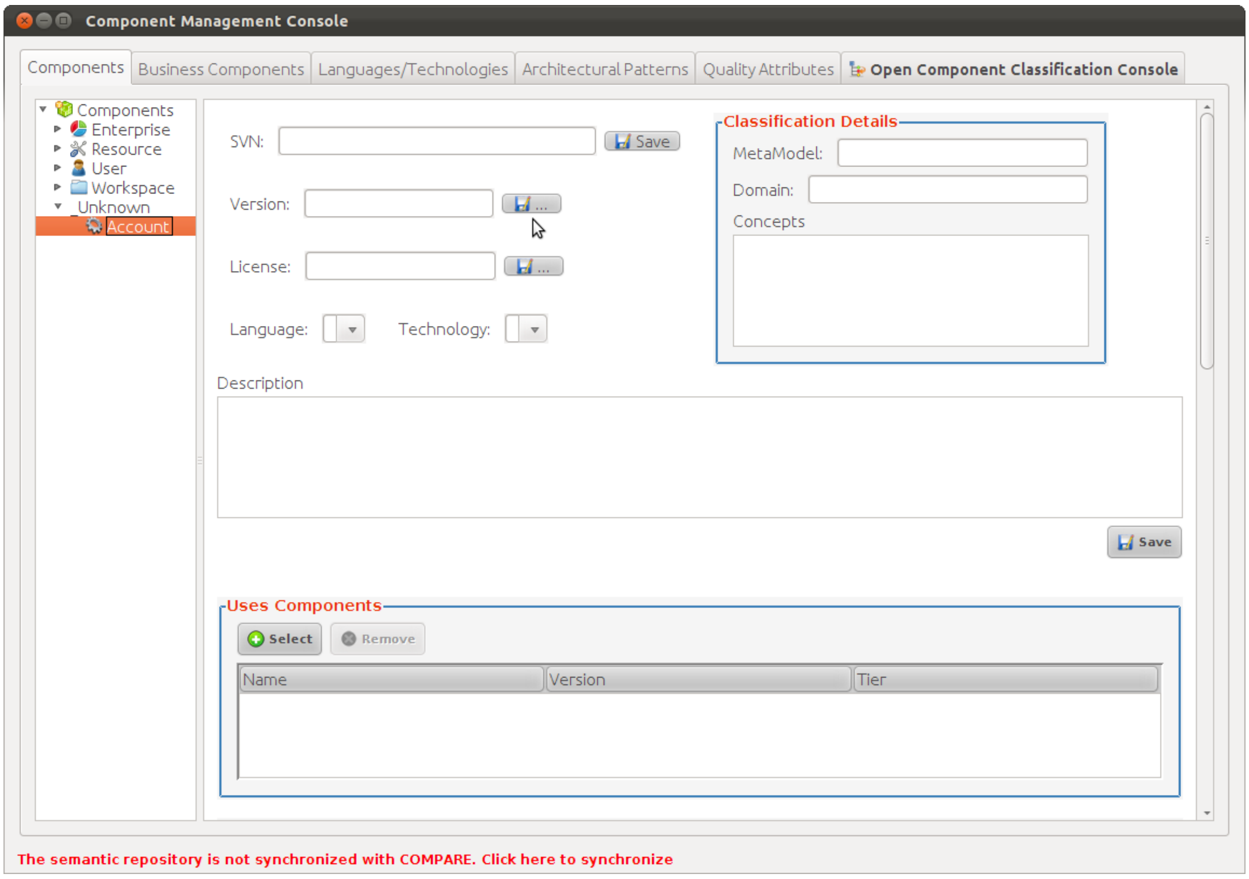
Component characterization consists of a series of information the Reuse Engineer needs to specify for a generated component. As you can see in the following image, this information is:

- SVN repository: The SVN address from where the packaged component can be downloaded.
- Version: The version of the component
- License: The license under which the component can be reused
- Language: The programing language to which the component was implemented
- Technologies: A specific technology with which the component was designed
- Description: A description of the main functionality of the component
- Uses components: Possible dependencies to other components
- Classification details: MetaModel, domain and concept(s) specification for the component

NOTE THAT:

- In order for the information to be saved you need to use the “save” buttons located to the right of each form field.

- All fields are optional. This means that every field can be left empty and be updated later on.
- In the classification details the Reuse Engineer can specify multiple concepts but only one MetaModel and domain.
- The contents of the fields of “Language”, “Technology” and “Classification Details” can be dynamically changed by the Reuse Engineer.

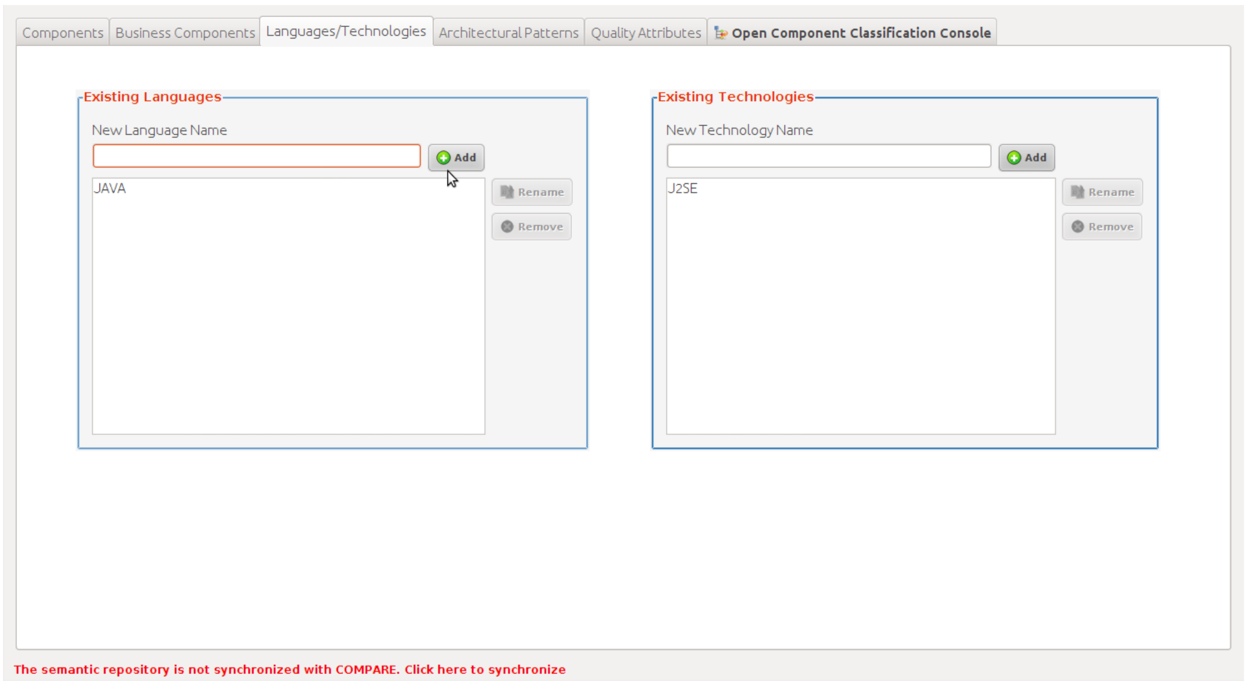


The semantic repository is not synchronized with COMPARE. Click here to synchronize

Figure 44 – Characterizing a component

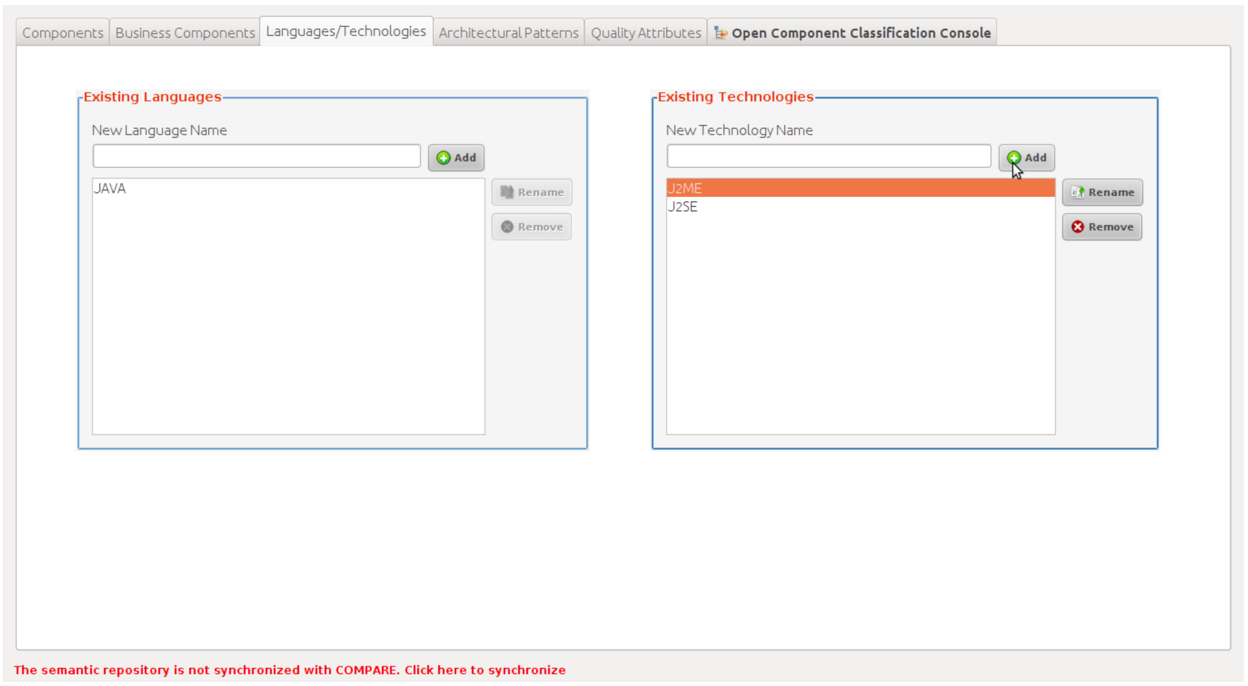
1.5.2 Adding / Removing / Renaming Languages and Technologies

The content of the fields of Language and Technology can be managed by the Reuse Engineer using the “Languages / Technology” tab. As you can see in the following image, in this tab, there are two separated areas that provide the Reuse Engineer with the options of adding, removing and renaming Languages and Technologies at will.



The semantic repository is not synchronized with COMPARE. Click here to synchronize

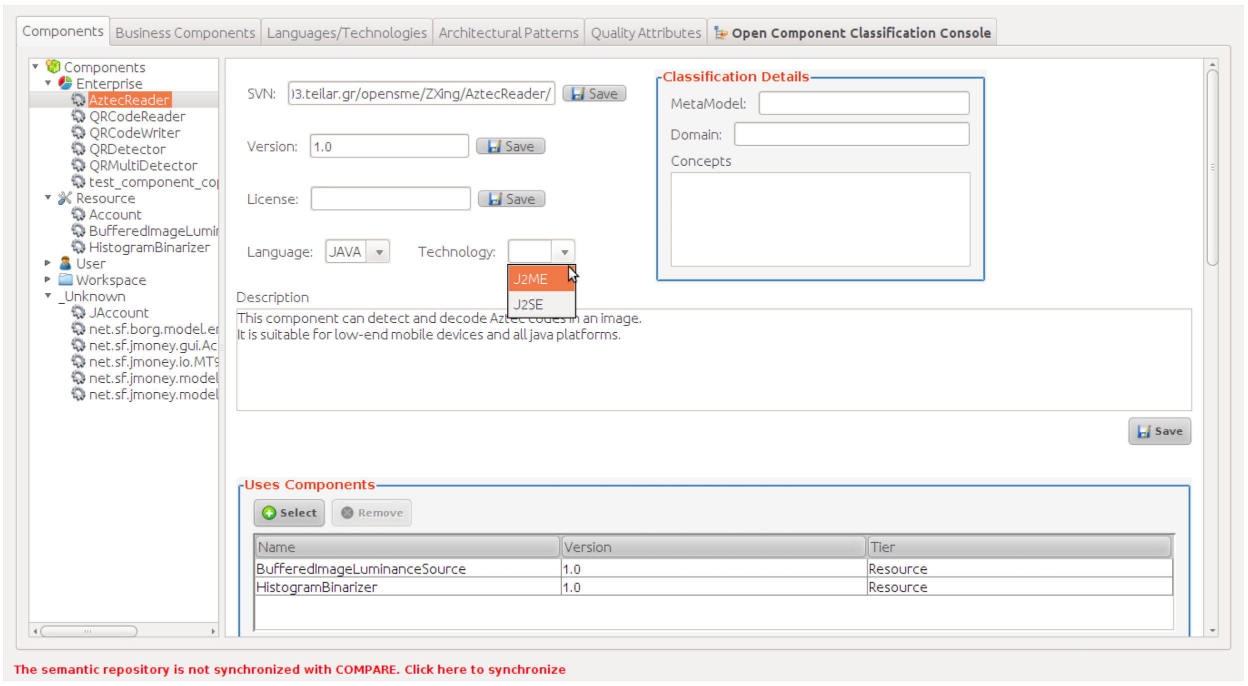
Figure 45 – Managing Languages and Technologies dialog



The semantic repository is not synchronized with COMPARE. Click here to synchronize

Figure 46 – J2ME technology added

<u>NOTE THAT</u>: The changes made using this dialog become automatically available to "Knowledge Manager" as you can see in the following image the "J2ME" technology we added in the previous step is now available to the "Components" tab.



SVN:)3.teilar.gr/opensme/ZXing/AztecReader/

Version: 1.0

License:

Language: JAVA Technology: J2ME J2SE

Classification Details

MetaModel:

Domain:

Concepts

Description

This component can detect and decode Aztec codes in an image.
It is suitable for low-end mobile devices and all java platforms.

Uses Components

Name	Version	Tier
BufferedImageLuminanceSource	1.0	Resource
HistogramBinarizer	1.0	Resource

The semantic repository is not synchronized with COMPARE. Click here to synchronize

Figure 47 – J2ME technology available at the Components Tab

1.5.3 Classifying Components

Classifying a component using the Knowledge Managers is identical to mapping the component with a MetaModel. Before the Reuse Engineer can do that s/he usually must create such a MetaModel using the "Open Component Classification Console" tab of Knowledge Manager.

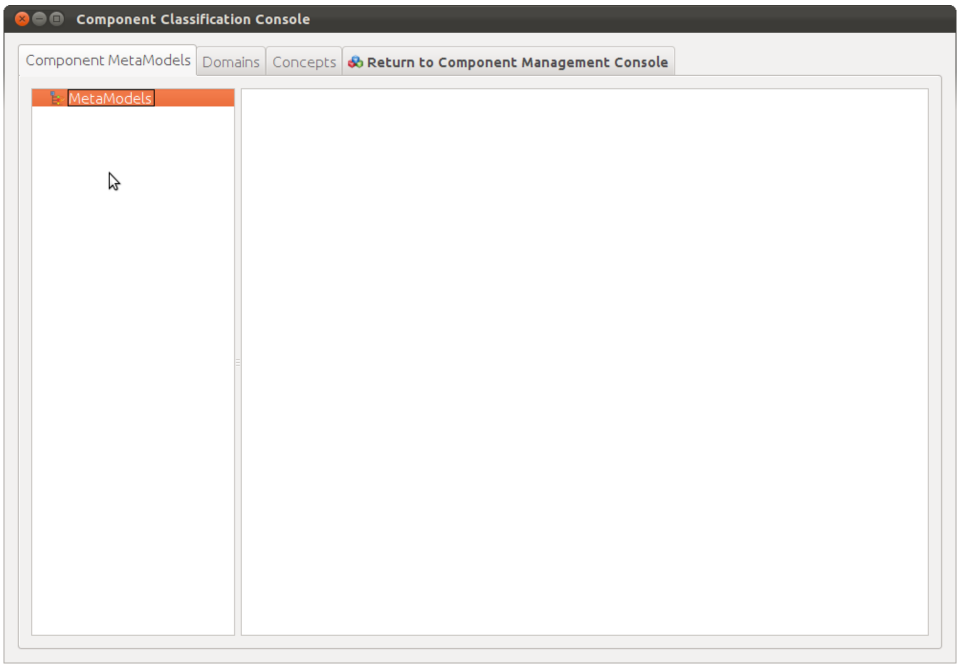

Component Classification Console
Component MetaModels
Domains
Concepts
Return to Component Management Console
MetaModels

Figure 48 – The main dialog of the Component Classification Console

A MetaModel consists of three parameters:

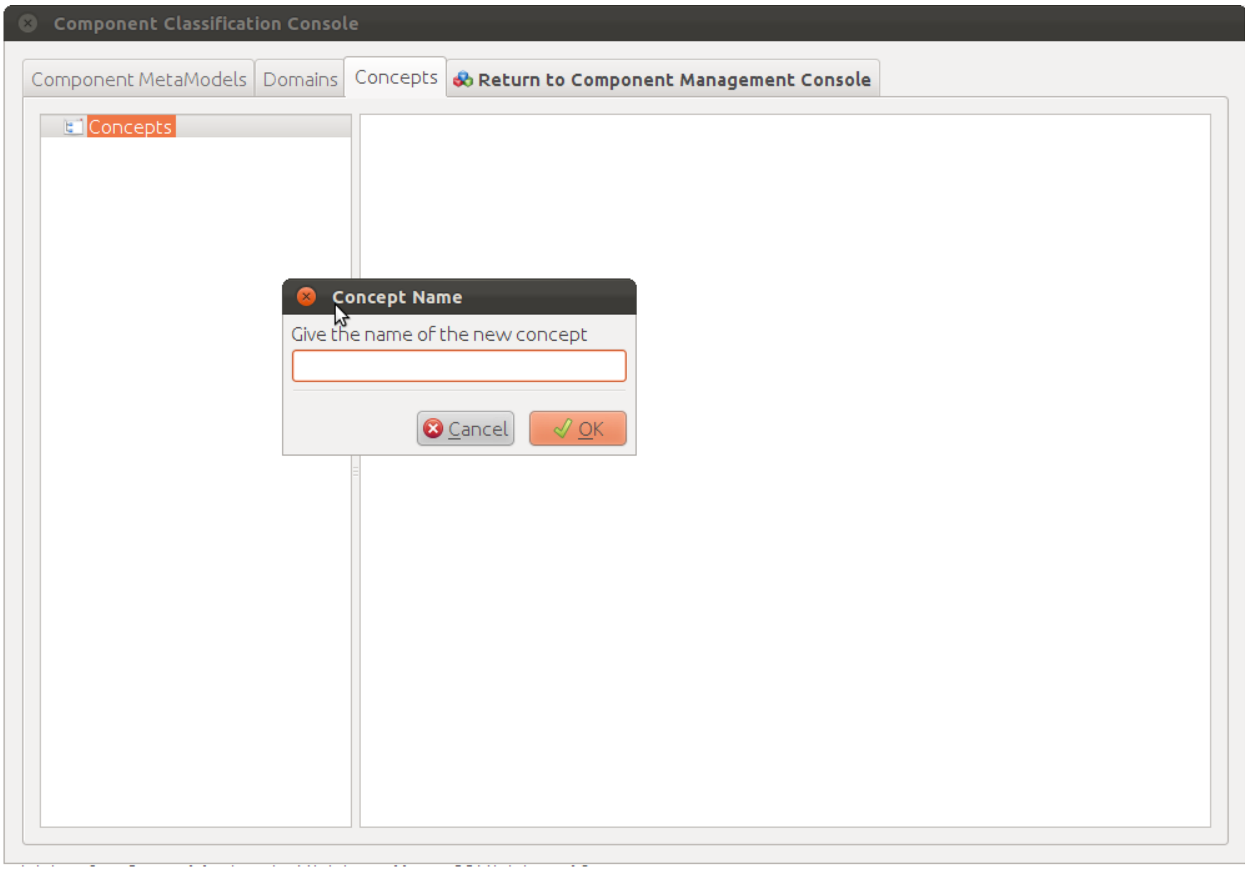
- Domain (unique)

- Conpept(s)
- Tier (optional)

In order for the Reuse Engineer to be able to define such a meta-model, the appropriate domain and concept(s) should have been created beforehand.

1.5.3.1 Adding a new Concept

1. Go to the **"Concepts"** tab of the **Components Classifying Console**
2. Right-click to "Concepts" element located to the left area of the dialog
3. From the pop-up menu choose "Add Concept"
4. In the dialog that appears type the name of the new concept
5. Click "OK" (at this step you can also abort the new concept creation by clicking on "Cancel")


Component Classification Console
Component MetaModels
Domains
Concepts
Return to Component Management Console
Concepts
Concept Name
Give the name of the new concept
Cancel
OK

1.5.3.2 Adding a new Domain

1. Go to the **"Domains"** tab of the **Components Classifying Console**

Follow steps 2 to 5 of the "Adding a new concept" process (see Adding a new Concept)

1.5.3.3 Assigning Concepts to Domains

After a domain has been created the Reuse Engineer can assign concepts to it. In the following image we have just created the "Accounting" domain and we need to assign the concept "Account" to it.

1. We click on the “Select” button located in the right area of the dialog (see Figure 49 – Assigning concept to domain)

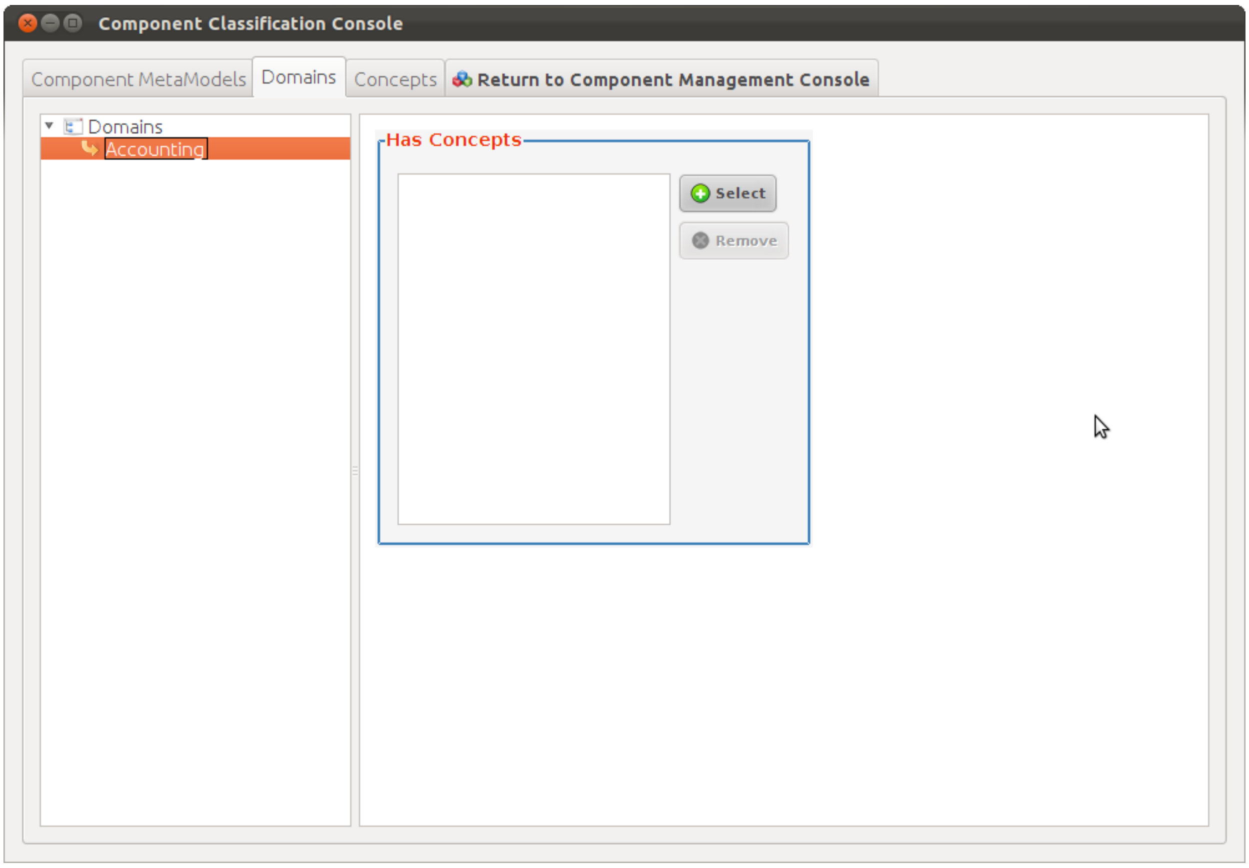


Figure 49 – Assigning concept to domain

2. In the dialog that appears (“Select Concept”) select the concept you want to assign.

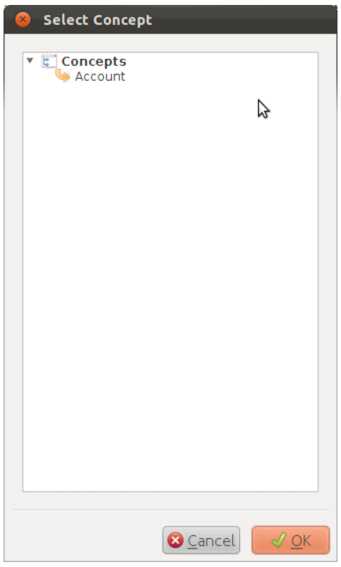


Figure 50 – Select concept dialog

3. Click to the “OK” button

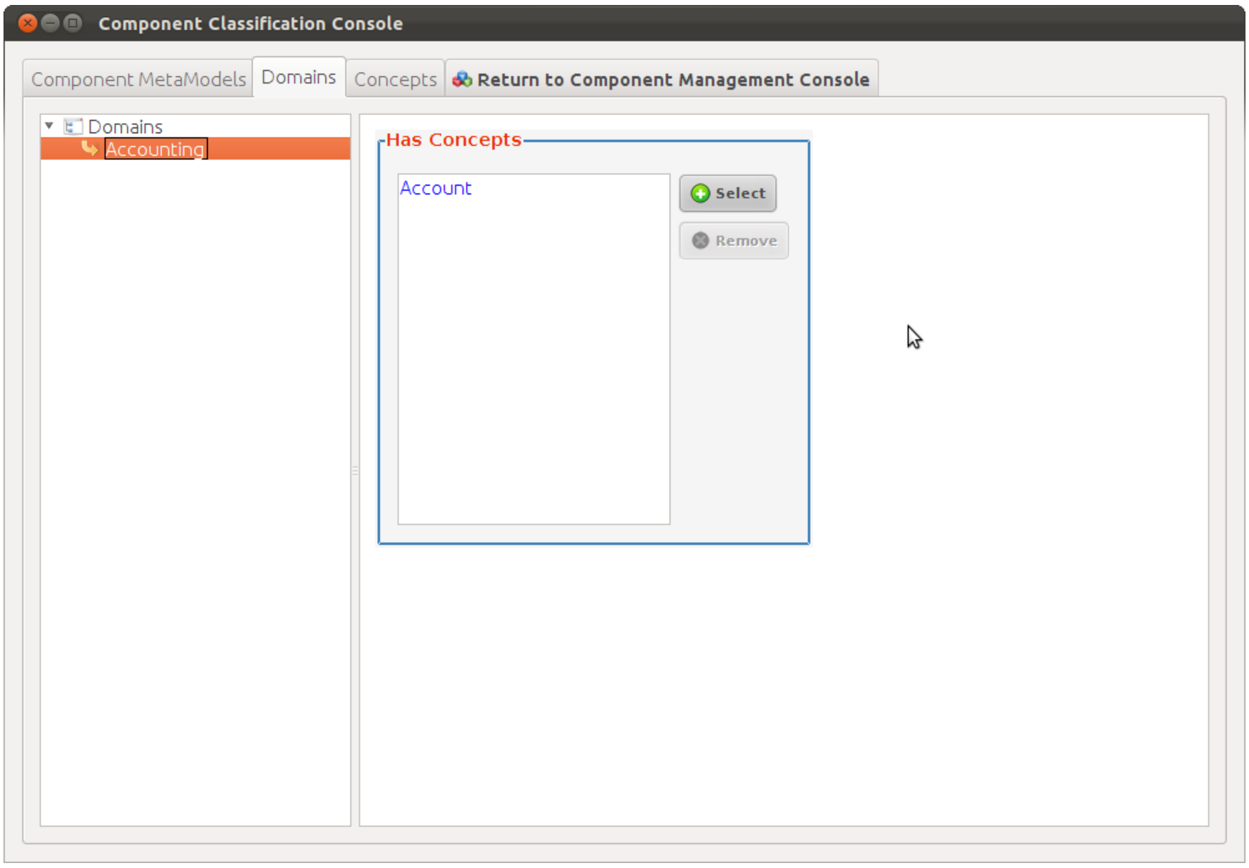


Figure 51 – Concept successfully assigned to a domain

The concept “Account” was successfully assigned to the “Accounting” domain.

NOTE THAT: If you go to the “Concepts” tab and select the “Account” concept you will see that the “Accounting” domain was automatically defined as the domain of the “Account” concept.

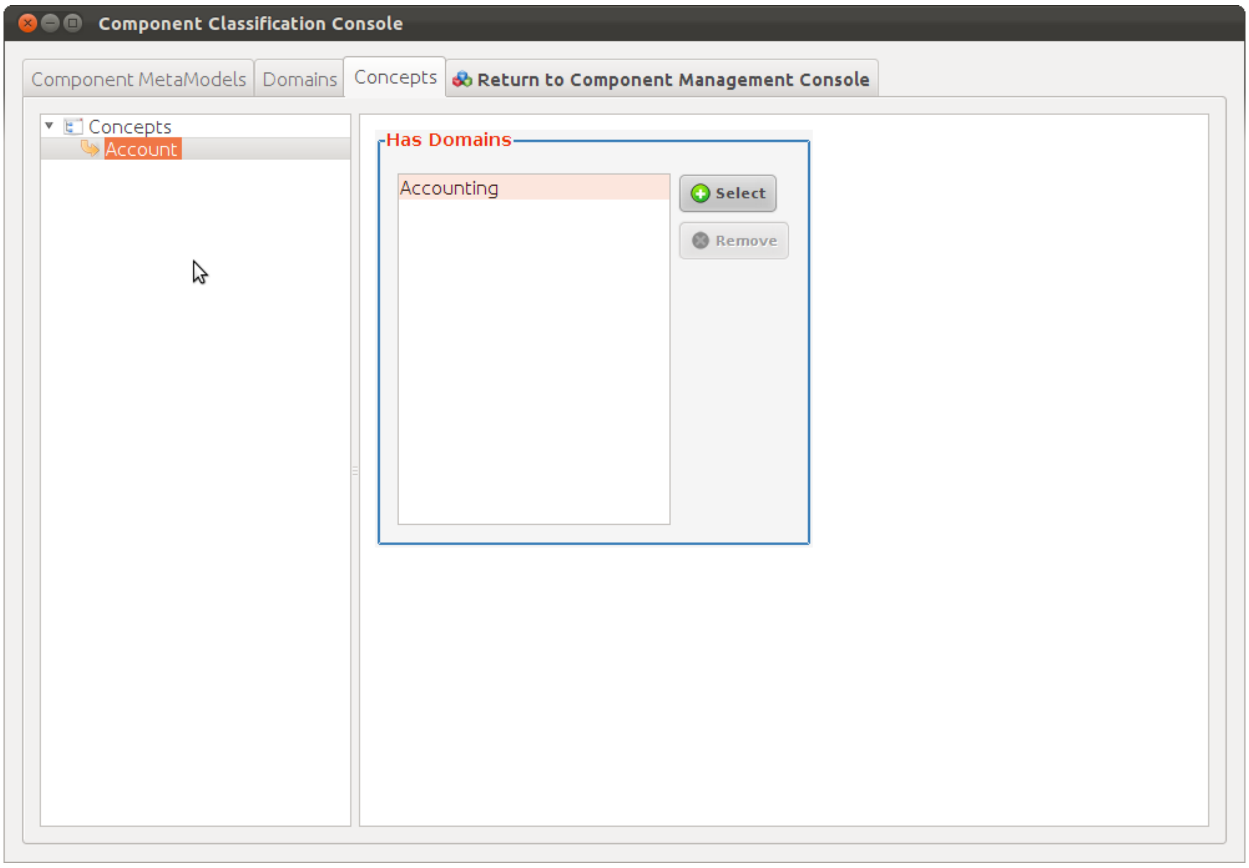


Figure 52 – Domain successfully assigned to concept

1.5.3.4 Creating the Metamodel

Having crated the necessary concepts and domains the Reuse Engineer can now proceed with creating a MetaModel.

<u>NOTE THAT</u>: For COPE, each MetaModel should be a member of a MetaModel group.

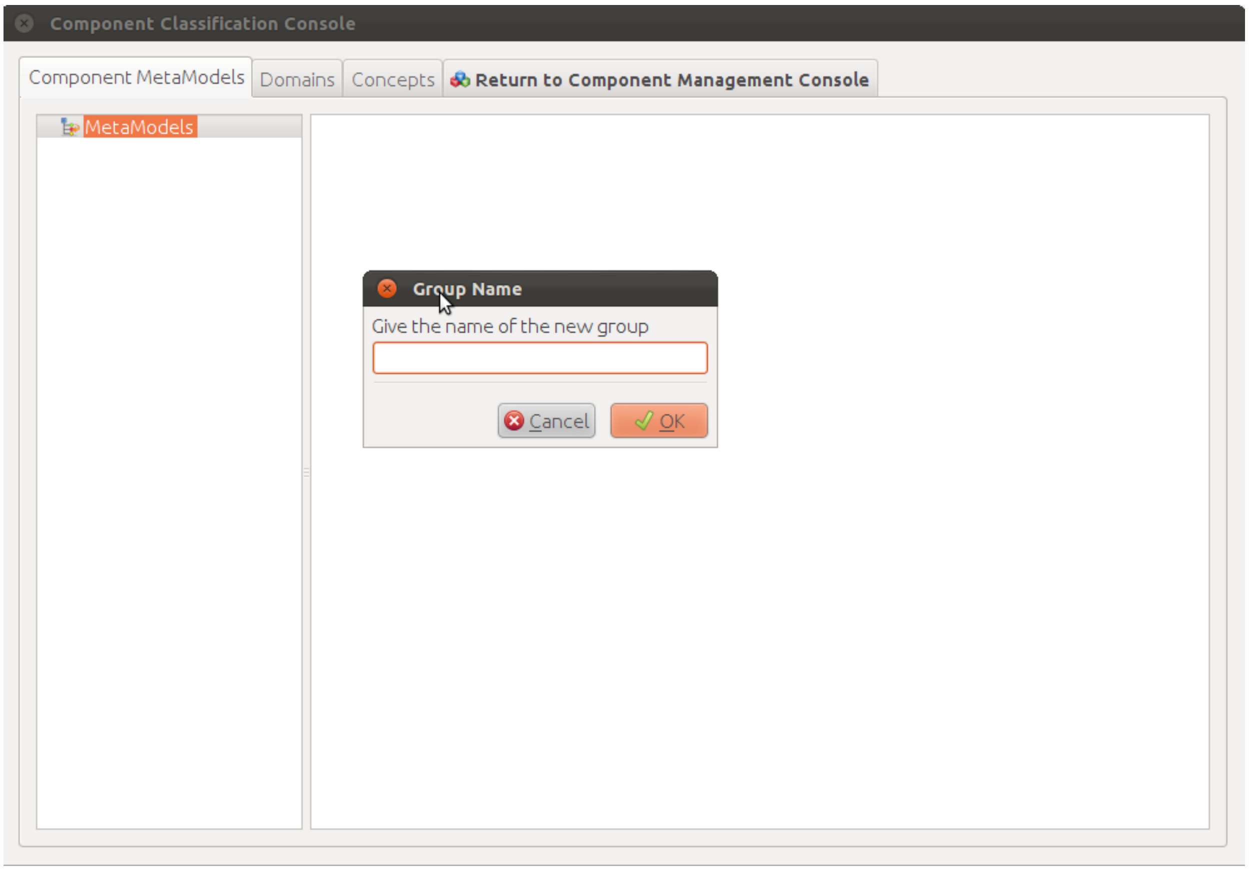

Component Classification Console
Component MetaModels
Domains
Concepts
Return to Component Management Console
MetaModels
Group Name
Give the name of the new group
Cancel
OK

Figure 53- Component MetaModels Dialog

To create a MetaModel:

2. Go to the **“Component MetaModels”** tab of the **Components Classifying Console**
3. Right-click to “MetaModels” element located to the left area of the dialog
4. From the pop-up menu choose “Add Group”
5. In the dialog that appears type the name of the new MetaModel group
6. Click “OK” (at this step you can also abort the new concept creation by clicking on “Cancel”)

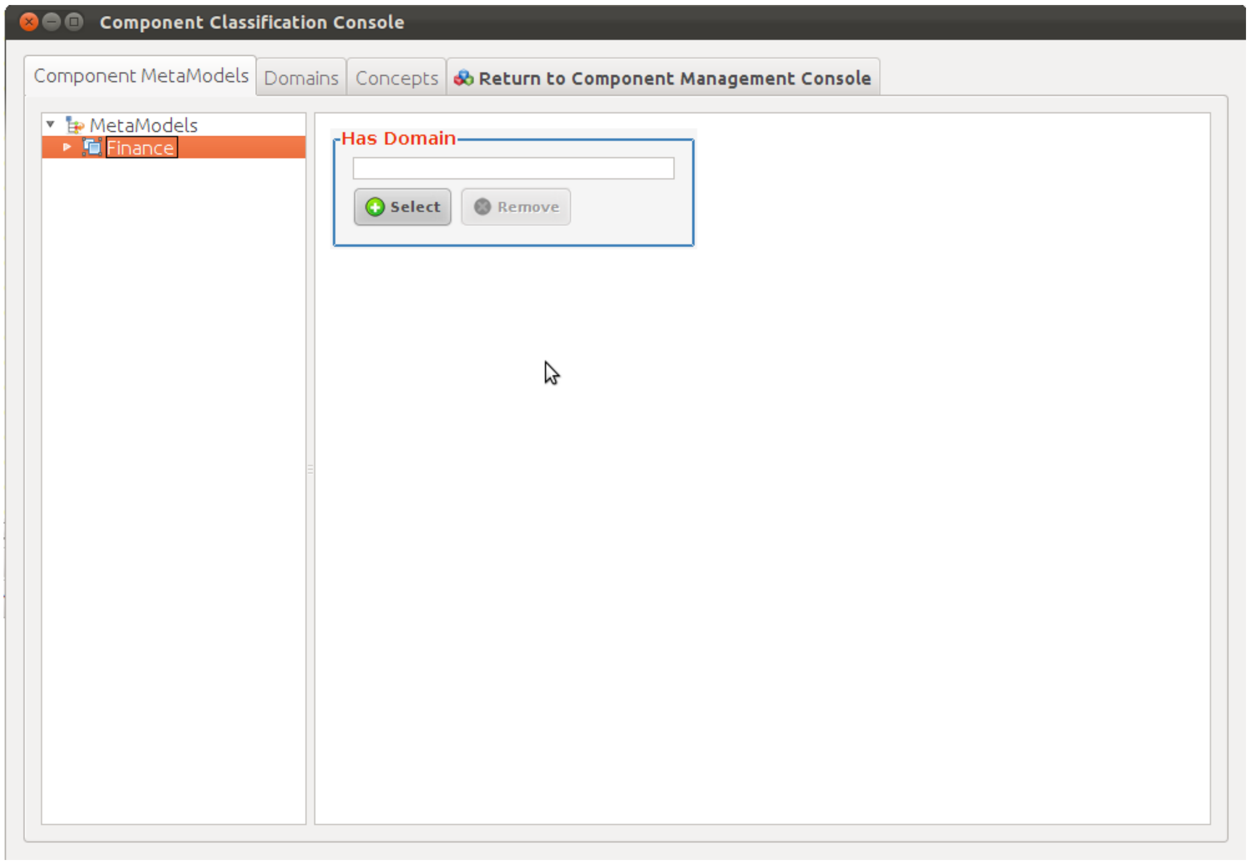


Figure 54 – MetaModel group successfully created

Once the MetaModel Group is created:

1. Right-click to the name of the MetaModel group
2. From the pop-up menu choose “Add MetaModel”
3. In the dialog that appears type the name of the new MetaModel
4. Click “OK” (at this step you can also abort the new concept creation by clicking on “Cancel”)

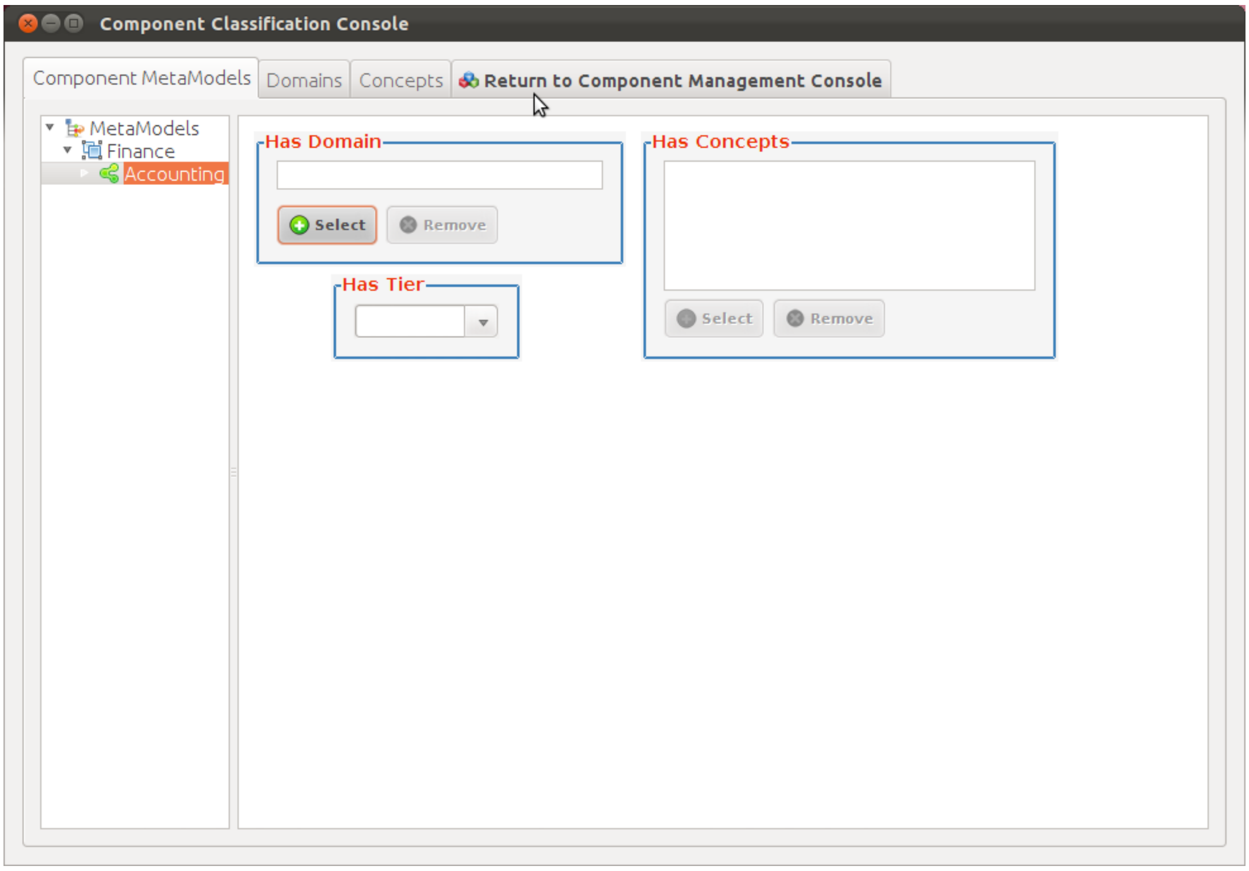


Figure 55 – MetaModel successfully created

After the successful creation of the MetaModel, the reuse engineer can now assign a domain to the MetaModel.

To do so:

1. Click to “Select” button located to the “Has Domain” of the Component Classification Console
2. From the dialog that appears, select the appropriate domain and click “OK”.

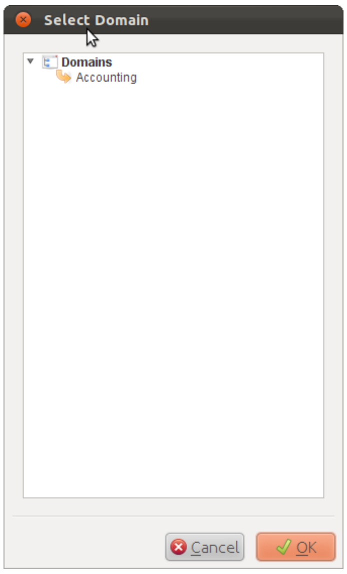


Figure 56 – Domain selection window

As you can see in the following image the domain is now selected. In addition the concepts associated with this domain have also been set to the "Concepts" area of the MetaModel.

<u>NOTE THAT</u>: All the dialogs of the Knowledge Manager that allow the Reuse Engineer to add elements also allow removal and renaming of those elements.

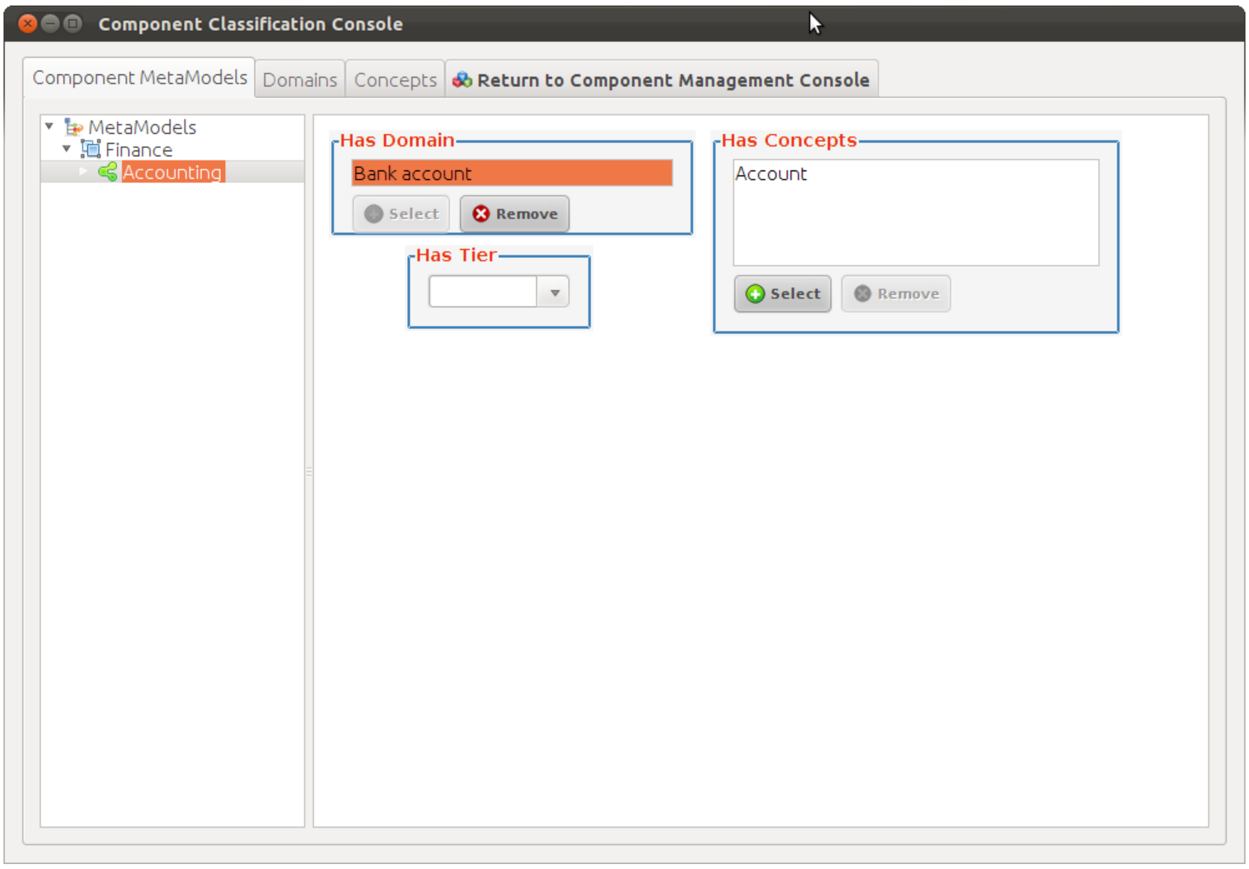
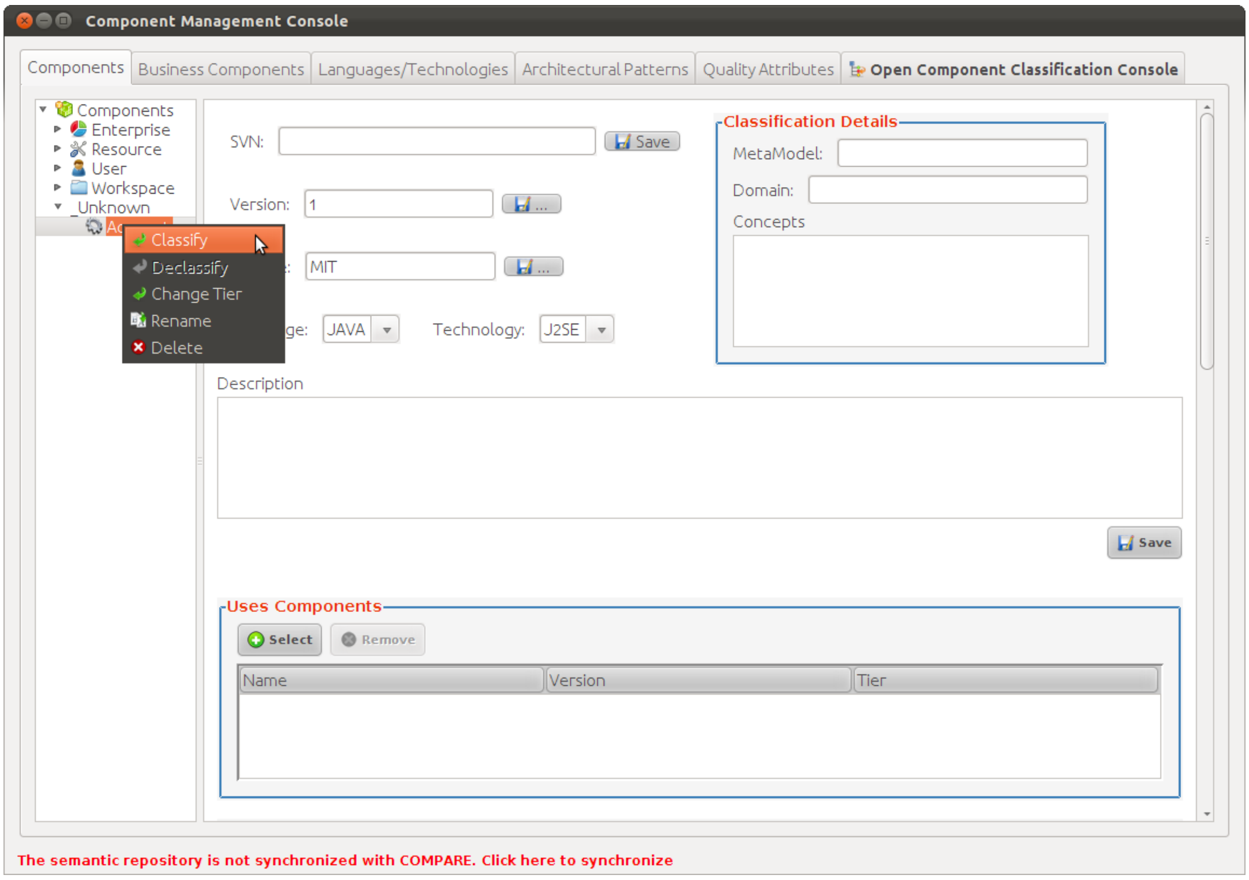


Figure 57 – Metamodel successfully created

With the creation of the MetaModel the Reuse Engineer has all the necessary information to classify the component. To do so s/he must return to the main dialog of the Knowledge Manager by visiting the "Open Component Classification Console" tab.

1.5.3.5 Classifying the Component

1. From the components tab located in the main dialog of the Knowledge Manager select the component you wish to classify (usually located under the "_Unknown" group).
2. Right-click to the component.
3. From the pop-up menu that appears choose "Classify".
4. The "Select Metamodel" dialog appears. Choose the metamodel and click to the "OK" button.



The semantic repository is not synchronized with COMPARE. Click here to synchronize

Figure 58 – Classifying a component

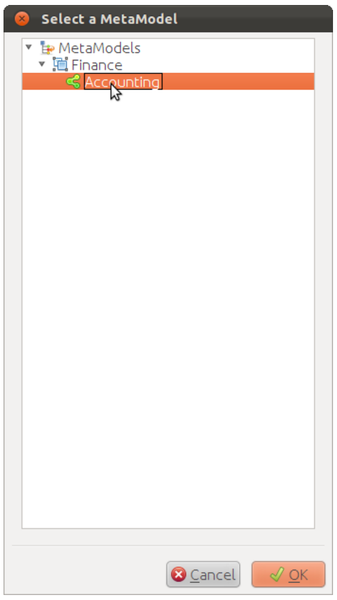
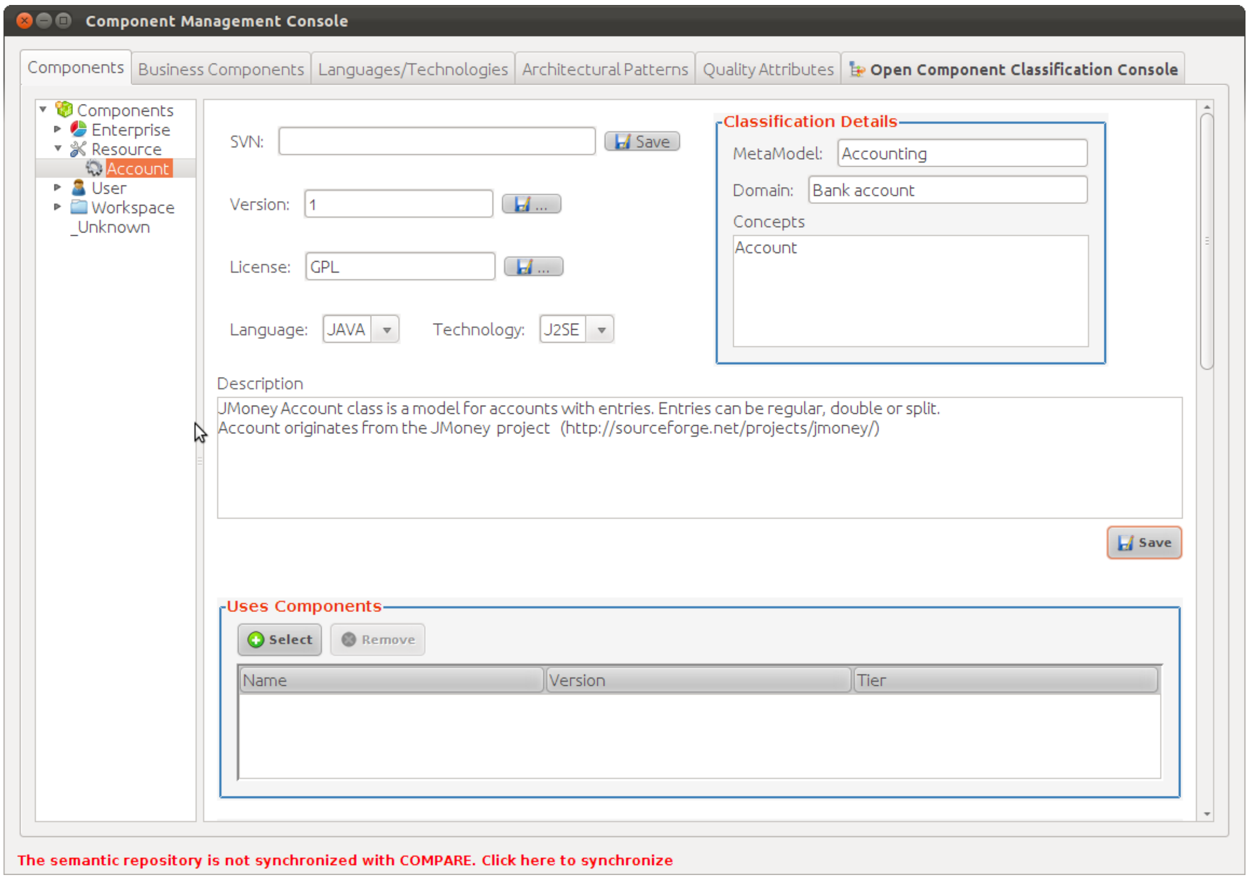


Figure 59 – MetaModel selection

Once the MetaModel is selected the classification details area for the component is automatically updated.



SVN:

Version: 1

License: GPL

Language: JAVA Technology: J2SE

Classification Details

MetaModel: Accounting

Domain: Bank account

Concepts

Account

Description

JMoney Account class is a model for accounts with entries. Entries can be regular, double or split.
Account originates from the JMoney project (http://sourceforge.net/projects/jmoney/)

Uses Components

Name	Version	Tier

The semantic repository is not synchronized with COMPARE. Click here to synchronize

Figure 60 – Component classification successfully finished

NOTE THAT: The classification of a component can easily change by right-clicking to the component and choosing “De-classify” and then repeating the process of classifying from scratch.

1.5.3.6 SELECTING TIERS

The Reuse Engineer can easily set the "Tier" for a component from the main dialog of the Knowledge Manager.

1. From the components tab located in the main dialog of the Knowledge Manager select the component (usually located under the "_Unknown" group).

2. Right-click to the component.

3. From the pop-up menu that appears choose "Change Tier".

4. The "Select Tier" dialog appears. Choose the appropriate tier and click to the "OK" button.

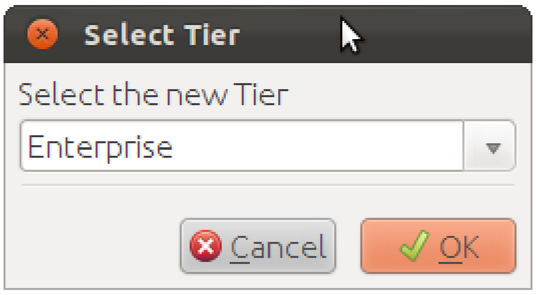


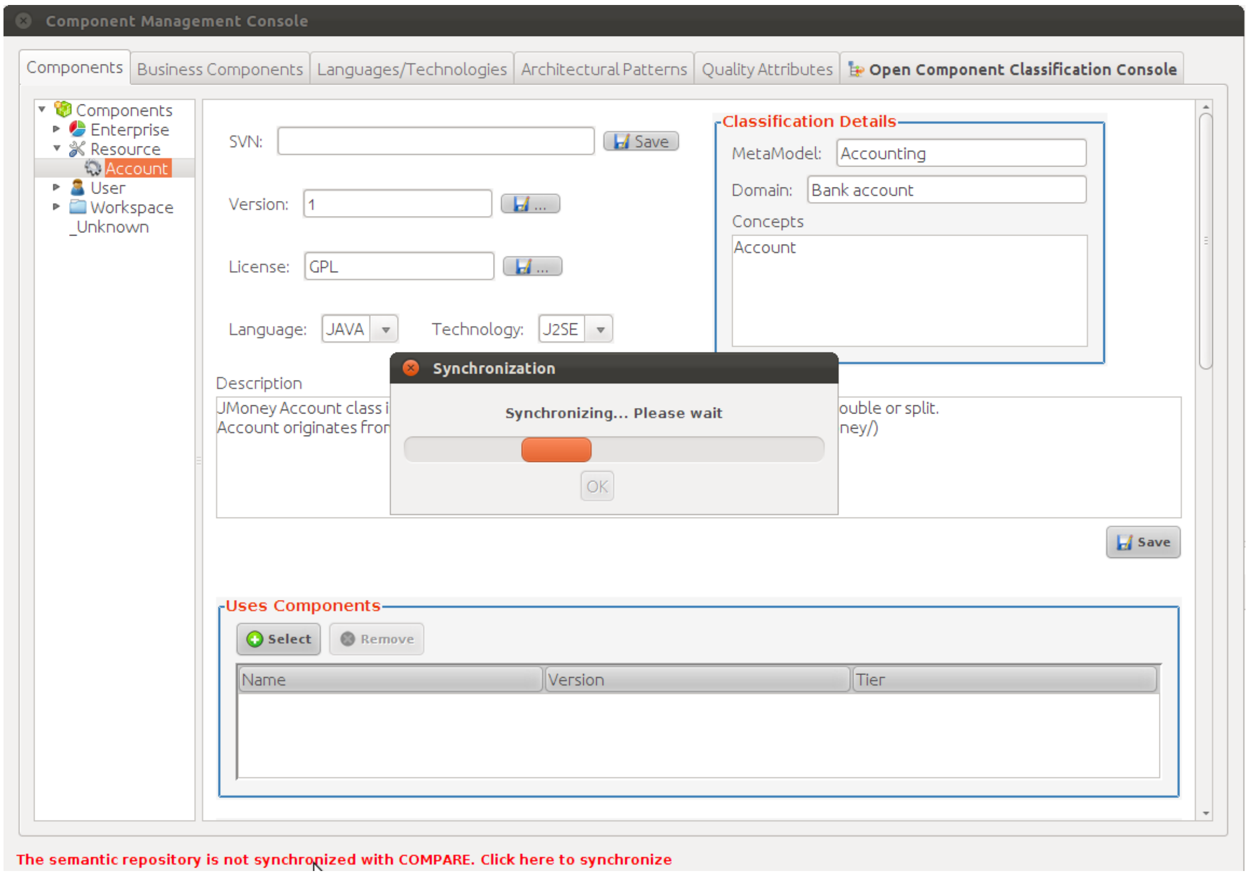
Figure 61 – Select tier dialog

The tier was successfully changed.

1.5.4 SYNCHRONIZING WITH COMPARE REPOSITORY

Once the reuse engineer has finished the characterization and classification for one or more of the generated components he can synchronize those changes with the COMPARE repository.

1. Click on the phrase "The semantic repository is not synchronized with COMPARE. Click here to synchronize" located in the bottom of the Knowledge Manager. The synchronization dialog appears.



SVN:

Version: 1

License: GPL

Language: JAVA Technology: J2SE

MetaModel: Accounting

Domain: Bank account

Concepts

Account

Description

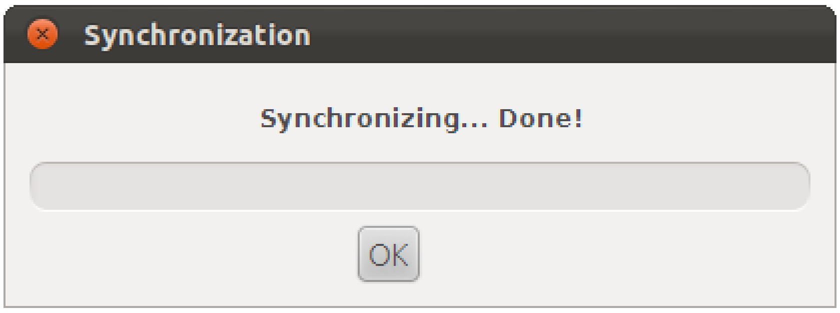
JMoney Account class i... ouble or split.
Account originates fro... ney/)

Synchronizing… Please wait

The semantic repository is not synchronized with COMPARE. Click here to synchronize

Figure 62 – Synchronization process between COPE and COMPARE

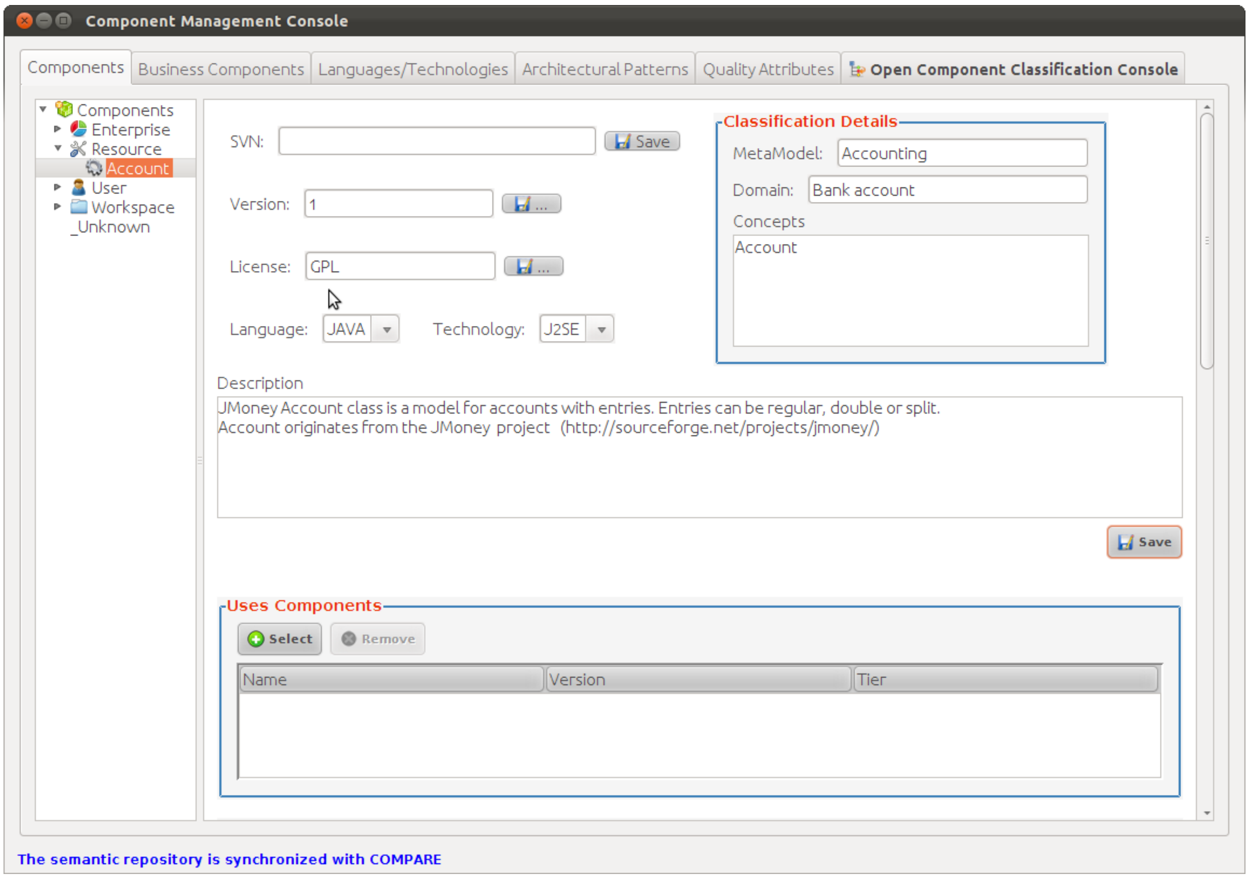
If no errors occur, the following dialog appears informing the Reuse Engineer that the synchronization was completed successfully.



Synchronizing… Done!

Figure 63 – Successful synchronization message

Also, the indication in Knowledge Manager’s main dialog changes to “The semantic repository is synchronized with COMPARE”.



SVN:

Version: 1

License: GPL

Language: JAVA Technology: J2SE

Classification Details

MetaModel: Accounting

Domain: Bank account

Concepts

Account

Description

JMoney Account class is a model for accounts with entries. Entries can be regular, double or split.
Account originates from the JMoney project (http://sourceforge.net/projects/jmoney/)

Name	Version	Tier

The semantic repository is synchronized with COMPARE

Figure 64 – Knowledge Manager's main dialog after the successful completion of the synchronization process

1.6 Known Issues & Workarounds

In this section we provide some known issues and possible workarounds for the COPE platform. The information is organized in subjects to make navigation easier and more efficient.

1.6.1 Documentation Generation

Known Issue: When the .jar file of a F/LOSS project and the given source code are not identical (in terms of files) malfunctions are likely to occur in the documentation generation process.

Workaround: Reuse Engineers are advised to manually produce the .jar file of the target F/LOSS project by compiling its source code.

1.6.2 Dynamic Analysis

1.6.2.1 Coverage

Known Issue: Methods that are not included in try blocks and throw exceptions, are not connected to the end of the method in the control flow graph. Due to that, LCSAJ might be missing some paths

Known Issue: Break labels are not recognized, so they are treated as simple statements in cfg. Due to that, LCSAJ might be missing some paths.

Known Issue: Over-approximated connection is used in methods inside try-block (meaning that every method call is connected to a catch statement). This might lead in generation of infeasible paths in LCSAJ

Known Issue: Ternary operators are not recognized, they are treated as simple statements. Due to that LCSAJ might be missing some paths.

1.6.2.2 VALIDATION

Known Issue: Components using classes from “java.utils” package raise exceptions in validation process.

Known Issue: Components generating huge state traces files (~3.000.000 lines) can’t pass the validation process successfully yet (e.g. components reading / writing bytes of images).

1.6.3 COMPONENT MAKERS

Known Issue: When the .jar file of a F/LOSS project and the given source code are not identical (in terms of files) malfunctions are likely to occur in the component making process (regardless of the component maker used).

Workaround: Reuse Engineers are advised to manually produce the .jar file of the target F/LOSS project by compiling its source code.

2. REFERENCES

1. E. Gamma, R. Helm, R. Johnson, και J. M. Vlissides: “*Design Patterns: Elements of Reusable Object-Oriented Software*”, Addison-Wesley Professional, 1994.
2. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin: “Aspect-oriented programming”, 11th European Conference on Object-Oriented Programming (ECOOP’97), LNCS vol. 1241/1997, pp. 220-242, Springer, 1997.
3. P. Herzum and O. Sims: “Business Component Factory : A Comprehensive Overview of Component-Based Development for the Enterprise”, Wiley, 1999.
4. J. Long: “*Software reuse antipatterns*”, SIGSOFT Softw. Eng. Notes, vol. 26, no. 4, pp. 68-76, 2001.