
©2012 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE
Extracting Components from Open Source
The Component Adaptation Environment (COPE) Approach

George Kakarontzas
Dept. of Informatics, Aristotle University of Thessaloniki
54124, Thessaloniki, Greece
Dept. of Computer Science & Telecom., TEI of Larissa
41110, Larissa, Greece
gkakaron@teilar.gr

Ioannis Stamelos, Stefanos Skalistis, Athanasios Naskos
Dept. of Informatics, Aristotle University of Thessaloniki
54124, Thessaloniki, Greece
(stamelos, sskalist, anaskos)@csd.auth.gr

Abstract—Open Source Software (OSS) represents an extremely valuable resource that is reused systematically almost in every software project. The reuse of OSS components however is restricted to ready-made components and developers who want to reuse code that exists in OSS projects but is not offered as a black-box component often resort to copying existing code and adapting it in their projects. OPEN-SME is a European research project which aims at providing ready-to-use OSS components that originate from existing OSS projects but are not necessarily provided as such. In this work we describe the Component Adaptation Environment (COPE) tool that was developed in the context of the OPEN-SME project and enables software experts, called reuse engineers, to extract components from OSS projects, test them and provide test documentation, validate them with Model-Based Testing techniques, package them and upload them in a component repository for reuse. The whole approach aims at creating an ever increasing repository of trustworthy reusable software components from different application domains.

Keywords—software components; component extraction; Open Source software;

I. INTRODUCTION

Open Source Software (OSS) reuse has the potential to improve software quality, shorten time-to-market and bring competitive advantages to Software Development SMEs. Currently OSS reuse is restricted to (a) complete OSS projects (e.g. Apache web server, MySQL Database), (b) the opportunistic reuse of isolated classes, or (c) well-known infrastructure components (e.g. Apache Commons). The OPEN-SME project proposal is to extend the landscape of OSS reuse to domain-specific components extracted by arbitrary OSS projects. Valuable OSS components exist in every OSS project. However it is difficult to recognize them, extract them, test them, document them etc. During software development, usually there is no time for the aforementioned activities. Developers often prefer to develop new code from scratch although this code has been written before many times by many others. Even when developers recognize the opportunity to reuse OSS code there are several uncertainties related to the provided functionality and quality. For example questions such as ‘What the component does?’ and ‘How well it does it?’ are questions that require time and effort to answer satisfactorily and usually this time is not available to developers during the project lifecycle.

The OPEN-SME approach introduces a reuse service that is operated by SME Association Groups (AGs) on behalf of their SME software development members. This service is performed by software experts of the SME AGs who produce components from OSS projects, test them, generate documentation, resolve licensing etc. asynchronously to application development by SMEs and independently from the SMEs. The components are related to domains that are relevant to the SMEs. Therefore when the SMEs will want to reuse them, the components will already be available in the component repository.

OPEN-SME results are two processes and three tools. The processes are (1) A Domain Engineering process carried-out by SME AGs who identify interesting OSS projects for their member SMEs and extract components from them, and (2) An Application Engineering process applied by software development SMEs. This application engineering process provides specific activities and tasks centered on reusing OSS software components. The tools are: (1) An Open-Source Software meta-search engine (OCEAN) which uses existing OSS search engines and provides a unified search capability available through a search engine website. (2) The Component Adaptation Environment (COPE) tool that is used by reuse engineers of SME AGs to recognize, extract, test, document etc. components from OSS projects, and (3) The extracted components are then placed in the Component Repository and Search Engine (COMPARE) tool that is used by SMEs to discover the extracted components in the context of the application engineering process.

In this work we will describe in some detail the COPE tool. The interested reader can find more details for the project’s results in the project website

1 http://opensme.eu
II. COPE

SME AGs experts who are the operators of COPE, are called reuse engineers. After they have identified a potentially interesting OSS project for the application domain of their software development SMEs they create a reuse project for this OSS project using COPE. A “Reuse Project” combines the source code related information (of the original OSS project) with information resulted from the analysis process carried out by the reuse engineer. A Reuse Project’s lifecycle consists of four phases. First there is an Analysis phase in which the source code of the target OSS project is being analyzed and the results of this analysis are being stored in the reuse project database. Then in the Component Recommendation phase the COPE tool automatically suggests class clusters that could serve as reusable components. The suggestions can be based on different criteria as discussed in Sec. II-A. Following in the Component Making phase a 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 reusable component. Finally in the Knowledge Management phase the user provides information for the generated components. Using the “Semantic Application” feature, the user can describe the functionality of each component. Moreover the reuse engineer can classify the resulting component to a specific domain and concept and finally upload the component to the COMPARE component repository.

The creation of a reuse project entails a preparatory phase in which the reuse engineer collects some project artifacts that are required by the COPE analyzers and recommenders. These artifacts include: (a) The binary file of the compiled program which in the case of Java is a Java Archive (JAR file), (b) The libraries used by the project which are a collection of external JAR files that the project reuses, (c) The Version Control System URL of the project if available, and (d) The source code directory of the project which contains the source files. COPE reuses itself a number of Open Source components to perform its analysis. Some of these components require the binary JAR file.

After a reuse project has been created the first step is to perform static analysis. Static analysis is used to collect dependencies and metrics from the source code. COPE stores these facts in a relational database that relates information extracted by different types of analysis and related tools.
extract. Furthermore after the component extraction has been performed the reuse engineer uses COPE to perform the testing and validation of the component and to create the test documentation for the testing and validation process. He or she also classifies the component under a domain and category and uploads it to the component repository where it becomes available to the reusers.

A. Component Recommenders

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 the following 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 clusters based on design patterns detected in the source code of the Reuse Project. Patterns are detected using the approach and the tool described in [4]. Currently Adapter and Proxy design pattern instances are used as indications for recommendation of clusters. These two patterns were selected as more relevant for the purpose of component identification. Other design patterns (e.g. Façade) may also be appropriate. The effectiveness of the different design patterns for component extraction is currently an active research area in our team.
- **Reusability Recommender**: 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 (i.e. the number of recursive dependencies of the class), the class Layer (i.e. how high or low is the class in the digraph of the project) and $R$ (our own reusability index based on the Chidamber and Kemerer metrics suite for OO design complexity) 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.

All recommenders present a similar dialog to the reuse engineer who can examine the recommendations. In Fig. 3 we can see this dialog.

B. Component Makers

Based on the analysis and recommendations carried out earlier the Reuse Engineer can now produce independent software components and then place these components in the repository using the ‘Knowledge Manager’ feature of COPE. Four different kinds of component makers are currently provided. The Interface Maker uses as input the clusters produced by the ‘Dependencies Recommender’. The Dependency Maker presents all the classes of the project along with
their reusability assessment and the reuse engineer can select a class and extract a component providing the functionality of the selected class. The Adapter Pattern Maker presents the clusters produced by the ‘Pattern Recommender’ and displays clusters involved in Adapter pattern instances. The Proxy Pattern Maker presents again the clusters produced by the ‘Pattern Recommender’ but this time it displays only clusters involved in Proxy pattern instances.

The user interface is again similar for the provided component makers as depicted in Fig. 4.

The reuse engineer can select a component as well as an interface generation policy (e.g. generation of an interface for the selected class, or generation of an interface for each externally referenced class) and provide a name for the component. The generated component contains all the required classes which are extracted from the project along with one or more generated interfaces for the component. Besides the original source code files and the generated interface or interfaces, the project libraries are also copied and an Ant build script is generated for the compilation of the component in an Integrated Development Environment (IDE).

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 and discover further dependencies that are required which are not recoverable through static analysis alone (e.g. data dependencies). The component can then be tested dynamically using the test cases or execution scenarios that were developed by the reuse engineer as we explain in the following Subsection.

C. Component Testing & Validation

After the component source files have been extracted the reuse engineer will process the component further in an IDE. This is an essential program comprehension step in which unit tests or execution scenarios examining a specific functionality are created. Also it is important to resolve additional dependencies, such as data dependencies, that are required for the component to work.

After the reuse engineer has created some test cases for the component using the IDE and has resolved any additional dependencies which are necessary for the component to work independently, returning to COPE the feature of Dynamic Analysis will enable the reuse engineer to do the following:

1) Compute different types of test coverage based on the tests that were created. The types of coverage include Statement Coverage of the Component, Statement Coverage per Method of the Component, Linear Code Sequence and Jump (LCSAJ) coverage of the Component, and LCSAJ Coverage per Method of the Component.

2) Produce a Control Flow Graph per method of the Component which depicts the paths followed during the method execution of the test cases. CFGs are generated statically parsing the source code of the component. Aspect-Oriented instrumentation is then used to instrument the byte code and generate the trace of the execution. The instrumentation is necessary for tracing the execution path through the CFG and for calculating the LCSAJ and Statement coverage.

3) Perform validation which is a Model-Based Testing (MBT) [5] approach in which a large number of unit tests are generated automatically, utilizing method invariants provided by the Daikon invariant detector [6] and the component is then tested against the generated tests.

4) Produce the test HTML report which is a number of HTML pages, similar to JavaDoc, that package all the aforementioned information to an easily accessible format. The test HTML report will be included in the component package when it is uploaded in the component repository.

D. Component Packaging & Classification

The component package that is generated from the usage of COPE is depicted in Fig. 5. It includes the following: (a) A top directory with the component name, (b) A readme.txt file which contains information such as: A short description of the component, the originating OSS Project, license or licenses, the programming language and technology, other components it uses if any, and the domain and main concept of the domain the component provides, (c) Component source files, (d) Required Libraries, (e) Component Documentation generated by UML commercial or open source tools, and (f) The test HTML report which includes separate subdirectories for each test case along with the test results (coverage etc.).

The component package is then compressed to a file that is then classified using the Knowledge Manager feature of COPE and is uploaded in the Component Repository.

The Knowledge Manager (Fig. 6) allows the reuse engineer to provide metadata for the component including:
The tier of the component. This is a characterization of the component’s intended layer in the system. The component can be an Enterprise level component which encapsulates domain-specific functionality, a Resource level component which provides a generic service (e.g. database storage), a Workspace component which can, for example, coordinate different Enterprise level components in a workflow, or a User Interface component.

- The URL of the component package from which the reusers can download the component.
- The version of the component
- The programming language (e.g. Java) of the component and the technology (e.g. Java Enterprise Edition)
- The other components that the component uses, and
- The Domain metamodel under which the component was classified and the domain and concept that the component implements from this metamodel.

In addition the reuse engineer can use an ‘Open Component Classification Console’ to define domain metamodels for domains and concepts of these domains that are used when providing the aforementioned component metadata. Finally the reuse engineer can upload the component after this classification to the component repository (called COMPARE) which makes it available to the reusers.

III. EARLY RESULTS OF COPE USAGE

In this section we provide details from using the Reusability Recommender and the corresponding Dependency Maker to extract components. Other recommenders were also used but the extracted components are too few at the moment to allow reporting conclusive results. Our experiments concerned five CRM/ERP and Project Scheduling projects, CentraView, Plazma, Ebineutrino, jBPM, GanttProject, which produced 60 components in total. In other domains we have also extracted components from the Zebra Crossing (ZXing) project which is a barcode image processing application in Java and the RosJava project, a pure Java implementation of the Robot Operating System (ROS). ROS provides a set of libraries and tools to create robot applications.

Regarding the Reusability Recommender procedure based on a selected class two approaches were followed. Both of them, require the selection of a central class to extract a component from. The first approach requires the selection of a class for which we want to extract a component and subsequently code inspection of every extracted class of the resulting cluster to determine the usefulness of the component under extraction. This approach proved to produce many meaningless components requiring substantial time for the cluster inspection. As a result a second approach was followed requiring only the examination of the selected class before extracting a component from it. An easy to use interface is provided by the COPE tool to instantly view the source code of a selected class. This approach proved more reliable, producing fewer but meaningful components.

Due to lack of space we provide details for two of the aforementioned projects, one from the CRM/ERP domain and one from the robotics domain.

A. The CentraView Project

CentraView is a browser based Enterprise Java (J2EE) Customer Relationship Management application which can be executed locally or as a hosted service. Components extracted from this application can be divided into categories such as customer relationship management, project management and email handling components.
Although the local execution nature of this application helped in the component detection and extraction process, producing wide variety of components, a problem considering the cluster size of many of these components was encountered. More specifically, there were about 350 classes between the fifth to eighth layer in the project digraph, producing components with cluster size between 50 to 70 classes. Components with large cluster sizes are difficult to understand and handle and usually contain irrelevant functionality regarding the functionality provided by the central class, which we wish to extract. A closer look at the source code of these components, helped us identify about 45 classes in common to most of them. Removing a specific method for database authorization, from the central class, alleviated the dependencies allowing us to remove many classes, reducing the cluster size of these components between 10 to 20 classes.

In Table I we can see the details of the components extracted from the CentraView project. Along with the description some time measurements are also provided. An interesting observation here is that the adaptation effort for the Invoice component enabled the reuse engineer to achieve significantly reduced adaptation times for the other components due to a similar pattern, with the exception of the Proposal component which had some additional issues.

### B. RosJava

The RosJava project is a pure Java implementation of the Robot Operating System (ROS). ROS provides a set of libraries and tools to create robot applications. Table II provides a list of the components extracted from RosJava.

Applying the component extraction process, the resulting components can be divided into enterprise and resource level components. Resource level components are related to cross-cutting features of an application such as security, message-passing, etc. The resource-level components generated from this project were three components that implemented the Publish - Subscribe design pattern. This design pattern is a messaging design pattern that allows the senders, called publishers, to send messages while being unaware of who the receivers, called subscribers, are. The application of this pattern provides a loose coupling between publishers and subscribers as publishers are not aware of who has subscribed to a specific message topic and subscribers are not aware of who the publisher (or publishers) of a specific message topic is (or are). Furthermore, if this design pattern is applied over a network it provides better scalability if specific techniques are used. The components that belong to the enterprise category are related to two different concepts of the robotics domain, namely range-finder and spatial representation. A laser scanner is a range-finder that uses a laser to find the distance between the observer, i.e. the robot, and possible obstacles, walls, other robots, etc. The components extracted provide a representation of a single

<table>
<thead>
<tr>
<th>Component</th>
<th>Detection (min.)</th>
<th>Analysis Evaluation (min.)</th>
<th>Adaptation (min.)</th>
<th>Testing &amp; Validation (min.)</th>
<th>Description</th>
<th>Min (No. of classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attendee</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Holds all the necessary information of an attendee individual</td>
<td>6</td>
</tr>
<tr>
<td>Email Account Detail</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>45</td>
<td>Provides detailed email account information</td>
<td>2</td>
</tr>
<tr>
<td>Expense</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>14</td>
<td>Holds information about any kind of expenses</td>
<td>16</td>
</tr>
<tr>
<td>Expense Form</td>
<td>5</td>
<td>7</td>
<td>14</td>
<td>40</td>
<td>Provides a general purpose expense form</td>
<td>10</td>
</tr>
<tr>
<td>FAQ</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>7</td>
<td>Holds details about frequently asked questions with their answers</td>
<td>10</td>
</tr>
<tr>
<td>Individual</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>45</td>
<td>Extensible component holding all the necessary information of an individual</td>
<td>6</td>
</tr>
<tr>
<td>Invoice</td>
<td>10</td>
<td>6</td>
<td>59</td>
<td>45</td>
<td>Provides a complete Invoice Form</td>
<td>16</td>
</tr>
<tr>
<td>License Information</td>
<td>3</td>
<td>10</td>
<td>9</td>
<td>45</td>
<td>Checks for the validity of a given license file</td>
<td>5</td>
</tr>
<tr>
<td>Note</td>
<td>2</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>Holds information about notes and their priority</td>
<td>10</td>
</tr>
<tr>
<td>Order</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>Holds information about orders</td>
<td>29</td>
</tr>
<tr>
<td>Payment</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>21</td>
<td>Holds details about payment transactions</td>
<td>17</td>
</tr>
<tr>
<td>Project</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>20</td>
<td>Provides information such as budgeted/actual hours, used hours, cross-modified/time-tracked data, owner/manager/identifier and much more</td>
<td>3</td>
</tr>
<tr>
<td>Promotion</td>
<td>5</td>
<td>5</td>
<td>14</td>
<td>40</td>
<td>Holds information about a marketing promotion</td>
<td>10</td>
</tr>
<tr>
<td>Proposal</td>
<td>3</td>
<td>5</td>
<td>39</td>
<td>45</td>
<td>Used for storing user input data during asking or editing of proposal</td>
<td>16</td>
</tr>
<tr>
<td>Question</td>
<td>10</td>
<td>3</td>
<td>9</td>
<td>7</td>
<td>Holds the details of a question submitted to a system</td>
<td>11</td>
</tr>
<tr>
<td>Report</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>14</td>
<td>Holds details about reports</td>
<td>4</td>
</tr>
<tr>
<td>Report Last</td>
<td>15</td>
<td>0</td>
<td>8</td>
<td>12</td>
<td>Holds a list with details about reports</td>
<td>8</td>
</tr>
<tr>
<td>Search</td>
<td>4</td>
<td>5</td>
<td>9</td>
<td>20</td>
<td>Holds the values of saved searches</td>
<td>7</td>
</tr>
<tr>
<td>Ticket</td>
<td>4</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>Holds details about a reported ticket</td>
<td>17</td>
</tr>
<tr>
<td>Email Account</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>Provides simple email account information</td>
<td>1</td>
</tr>
<tr>
<td>Entity</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>7</td>
<td>Provides a general purpose entity ontology</td>
<td>6</td>
</tr>
<tr>
<td>Email Message</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>10</td>
<td>Provides all the necessary properties of an email message</td>
<td>48</td>
</tr>
<tr>
<td>Contact</td>
<td>7</td>
<td>0</td>
<td>15</td>
<td>15</td>
<td>Provides simple yet complete contact information</td>
<td>4</td>
</tr>
<tr>
<td>User</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>10</td>
<td>Provides information about a CRM system user such as Customer, Employee, Administrator and many more</td>
<td>6</td>
</tr>
</tbody>
</table>
scan of the laser scanner as well as features related to the laser scanner device such as device configuration and operation. The spatial representation concept refers to the representation of points and vectors into 3-dimensional space; the components relevant to this concept provide such capabilities including various operations like multiplication, addition, normalization as well as a domain-specific representation called Quaternion. Regarding the range-finder concept, a component for a specific wide-spread laser-scanner device was extracted that handles the connection of the device, provides device state monitoring and error handling. A component was also extracted related to the spatial representation, that uses the quaternion component, in order to perform fast transformations of vectors. This feature is required in the robotics domain as a robot must be able to transform absolute to relevant positioning and vice versa in order for the robot to map range-finder readings to a global representation system, know its exact location, where other objects are located, etc.

IV. RELATED WORK

The authors in [7] describe a refactoring [8] for java programs called “Extract Component” which transforms a group of Java classes based on their dependencies to a Java Bean and uses the classes that use the Java Bean from the original program as usage examples. The clustering algorithm of [7] is only one of the possible algorithms that can be used as COPE component recommenders. Furthermore COPE provides features for component testing/validation and domain classification.

Allier and colleagues in [9], use dynamic analysis to identify components. Their approach assumes that existing use case documentation or system-level use cases are available. Also in order to construct the Dynamic Call Graph between the classes of the program the availability of meaningful test data is assumed. In our experience these may not always be available in OSS projects. Furthermore, the partitioning proposed requires that a class belongs to one and only one component. However (a) classes lower in the digraph of the project are usually general classes with high fan-in that are called by many higher-layer classes and (b) these classes are not called directly by any use case scenario because they represent infrastructure features. Due to (b) they cannot be identified as independent components by dynamic analysis with use cases and due to (a) it is overly restrictive to enforce their inclusion in any particular component. The above suggest that this method will be probably more successful if applied at the business logic layer of the application.

The work in [10] describes a semi-automated approach to concept identification in source code. The authors use metrics as indicators of important concepts in a domain. The metrics collected are those produced by the Eclipse metrics framework. After collecting the metrics they use a supervised learning algorithm (i.e. machine learning) called Support Vector Machines (SVM) to classify the results. They have applied their method in two projects and showed that the proposed approach provides a good indication of which classes in the source code represent concepts of the application domain. The conclusion is that the domain classes have metrics’ vectors of both size and complexity that are considerably different than the respective metrics’ vectors for classes which do not represent domain concepts. In COPE we also use static analysis and the CK metrics [2] to compute a reusability assessment which is indeed statistically different for domain, infrastructure, user interface etc. classes. Reusability assessment is one of the methods used in COPE to identify potentially interesting classes as starting points for component extraction. However we have found that although a useful indication metrics alone are not sufficient for software architecture recovery.

In another related line of research concerning dynamic analysis and program comprehension, Cornelissen and colleagues in [11] present a literature survey of research in the field of program comprehension through dynamic analysis. They provide a faceted classification of the research area. Facets include the activity (e.g. feature location), the target (e.g. OO software), the method (e.g. slicing), and the evaluation approach (e.g. case study). An example of such an

<table>
<thead>
<tr>
<th>Component</th>
<th>Detection (min.)</th>
<th>Analysis Evaluation (min.)</th>
<th>Adaptation (min.)</th>
<th>Testing &amp; Validation (min.)</th>
<th>Description</th>
<th>Size (No of classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Topic</td>
<td>5</td>
<td>5</td>
<td>15</td>
<td>80</td>
<td>Defines a Topic to implement the Publisher-Subscribe pattern</td>
<td>6</td>
</tr>
<tr>
<td>Default Publisher</td>
<td>5</td>
<td>15</td>
<td>20</td>
<td>120</td>
<td>Publishes messages over a Topic.</td>
<td>30</td>
</tr>
<tr>
<td>Default Subscriber</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>60</td>
<td>Subscribes to a Topic and receives messages from Publishers.</td>
<td>69</td>
</tr>
<tr>
<td>Laser Scan</td>
<td>1</td>
<td>6</td>
<td>15</td>
<td>20</td>
<td>Holds all the necessary information about a laser range-finder.</td>
<td>2</td>
</tr>
<tr>
<td>Laser Scanner Device</td>
<td>5</td>
<td>6</td>
<td>30</td>
<td>45</td>
<td>Provides configuration, operation and error-handling about laser range-finders</td>
<td>5</td>
</tr>
<tr>
<td>Laser Scan Publisher</td>
<td>5</td>
<td>15</td>
<td>35</td>
<td>40</td>
<td>Publishes laser scan to anyone subscribed.</td>
<td>9</td>
</tr>
<tr>
<td>Hokuyo Laser Scanner Device</td>
<td>3</td>
<td>12</td>
<td>25</td>
<td>25</td>
<td>Provides configuration, operation and error-handling for the Hokuyo URG laser-finder</td>
<td>12</td>
</tr>
<tr>
<td>Vector3D</td>
<td>1</td>
<td>4</td>
<td>10</td>
<td>15</td>
<td>Represents a 3-dimensional vector and provides basic operations between them</td>
<td>1</td>
</tr>
<tr>
<td>Quaternion</td>
<td>3</td>
<td>6</td>
<td>20</td>
<td>10</td>
<td>Represents a quaternion and provides fast operations for transformation</td>
<td>2</td>
</tr>
<tr>
<td>Geometric Transformation</td>
<td>3</td>
<td>12</td>
<td>35</td>
<td>35</td>
<td>Defines and performs a transformation in terms of rotation and translation</td>
<td>3</td>
</tr>
</tbody>
</table>

Table II: RO$\tilde{\text{O}}$JAVA COMPONENTS
The effect of the experience of the reuse engineer in the COPE in supporting different languages and technologies. and technology neutral model that we can exploit to extend the process. COPE currently supports only Java applications.

The existence of the database however provides a language and technology neutral model that we can exploit to extend COPE in supporting different languages and technologies. The effect of the experience of the reuse engineer in the component extraction process is also an interesting future research area.

ACKNOWLEDGMENT

This work is partially funded by the European Commission in the context of the OPEN-SME ‘Open-Source Software Reuse Service for SMEs’ project, under the grant agreement no. FP7-SME-2008-2/243768.

REFERENCES


