CHAPTER 5
ONTOLOGY EXTRACTION FOR PRODUCING ARCHITECTURE WITH PROPOSED ODRET TOOL

5.1 INTRODUCTION
The term ontology has become popular in several fields of informatics like artificial intelligence, agent systems, and database or web technology [122]. In the context of computer sciences the term ontology stands for a formal explicit specification of a shared conceptualization [123]. Software engineering (SE) is a field where conceptualisation plays a major role, e.g. in the early phases of software development, in the definition, use and re-use of software components and as a basis for their integration. Thus, ontologies are likely to invade the SE field as well soon. Currently, researchers not only emphasize on ontology development, but also ontology analysis. As more ontologies are generated and reused, more tools will be available to analyse ontologies [123]. Hongji et. al [124] suggested that ontology has a great potential for legacy systems understanding and re-engineering. To reengineer legacy system is to bring out the underlying models and conceptualization used by software systems and to represent them explicitly. The explicit representation of conceptualization is commonly called ontology in the context of knowledge sharing. Ontology is considered to be the prerequisite of any system for reuse and interoperation [125]. Natalya et. al [126] facilitated guidelines to create first ontology. Authors have used experience of using Protégé-2000 (Protege 2000), Ontolingua (Ontolingua 1997), and Chimaera (Chimaera 2000) as ontology-editing environments and described an ontology-development methodology for declarative framework based systems. They listed the steps in the ontology-development process and addressed the complex issues of defining class hierarchies and properties of classes and instances. However, after following all the rules and suggestions they concluded that: there is no single correct ontology for any domain.

Ontology design is a creative process and no two ontologies designed by different people would be the same. The potential applications of the ontology, designer’s understanding and view of the domain will undoubtedly affect ontology design choices. The quality of one’s ontology can be assessed only by using it in applications for which it is designed. Wache et al. [127] discussed a useful review on ontology-based integration of information. They presented the
results of an analysis of existing information integration systems from an ontological point of view. The analysis focused on systems and approaches in which ontologies are a main element. Wessman et al. [128] discussed that ontologies can achieve a high degree of accuracy in data extraction while maintaining resiliency in the face of document changes. Author presented an ontology-based data-extraction framework written in Java a means for explicitly modelling extraction plans, an extraction-ontology schema, OSMX, written in XML Schema, and a re-implementation of the legacy version of ontos within the new framework. Gopinath et al. [129] proposed a framework to extract the metadata from the source code using QDox code generators and stores it in the OWL using the Jena framework automatically. The framework extracts the metadata from the source code using QDox code generators and stores it in the OWL using the Jena framework automatically. Only Java projects are used in this framework. The Java source file or folder that consists of Java files is passed as input along with project information like description of the project, version of the project.

Hesse et al. [130] outlined various ontology-based software engineering (OBSE) approaches and discussed languages and tools that are preferred for OBSE. According to Wolfgang an ontology language in the SE context should primarily be made for humans, but is hopefully being translatable for automata. Of course, UML is a serious candidate for an OBSE language. Author concluded that OBSE might become an attractive software engineering paradigm which serves for closer cooperation, better compatible models, more re-usable components and less costs in the software development field. Generating ontologies have eminent benefits for re-engineering of a software system [44]. While transforming a software system, it is important to understand the different concepts and vocabulary associated with the system and ontology is the explicit representation of these concepts. Ontology is also considered to be the prerequisite for any system for its reuse and interoperation [131]. In literature ontology generation is considered to be very useful for recovering software architecture from object oriented design and facilitating the exploration of unknown and hidden artifacts [132]. An ontology consists of all relevant concepts of the studied system and provides good coverage of the whole system by identifying the relationship between domain concepts [133].

In proposed framework simplified view of ontology is captured and it defines set of terms, including classes, objects, methods, relations and instances. In chapter 4 components are generated from the object oriented system using RSA tool with application of hierarchical clustering and binary weighting scheme after feature extraction. Here, ontology driven reverse
engineering tool (ODRET) is developed which is a run time environment for automatically generating the ontology pool of components from the source code of the object oriented system and represent them in the form of design diagrams. Various tools are available in literature for automatic transformation of software system to design diagrams, but none of them considered ontology generation as an intermediate stage for understanding the system dependencies and its vocabulary. Also, most of the developed tools do not support reverse engineering capabilities for .Net applications developed using C# language and also ontology generation is not considered as an intermediate stage to understand functionalities of a software systems. So, after examining various tools available in literature viz. RSA, Star UML, Solidsx, U Model, Enterprise Architecture, and Rational Rhapsody, ODRET is developed for ontology based integration of design information and extraction of architectural view of the re-engineered system efficiently. The developed tool provides a user friendly environment to accept software systems developed in c#.net.

To extract ontology, a hierarchical search is performed on the entry point of software systems. The structural information extraction is performed module wise, so that the unrequired code filtration is also performed. Only the linked modules are identified along with integrated module representation. These modules are represented by relative elements as well as the interconnection with other modules and the low level components are extracted by applying the code level analysis. The algorithmic model is defined to extract these components and categorize them under module, methods, objects etc. Once these components are generated, the next task is to define them in structured form by keeping the relative components in one pool. Later on the relation is established between these components based on access flow. This relational result is finally transformed to different design time models: class diagrams, object diagrams and ER diagrams.

5.2 SOFTWARE COMPONENT RETRIEVAL METHODOLOGY
Component retrieval techniques provide a list of components relevant to search query. A lot of research is performed on how to search the best component in return. To provide result quickly, even with large data sets, some sort of indexing is required to map query terms to components, as well as appropriate algorithms for building the index ahead of time.

The main criteria for retrieval of components broadly depend upon three factors:

1. Representation scheme, i.e. how the components are indexed in memory.
2. The classification mechanism used for components
3. Searching mechanism (keyword, facet, attributes, etc.).

Outline for component retrieval methodology used in this study is explained in figure 5.1.

As the size of repository grows, it becomes a time consuming process to extract the artifacts for software components. The presented research is focused on optimization of this task. The system will perform an efficient search from repository using heuristic search. It will also perform a classification to categorize the components. The proposed architecture for the efficient search is given as:

In the figure 5.2 first users will enter the file name which acts as a repository where required component is searched. After selection of repository, components that present in that repository are retrieved and their classification is done on the basis of type of components. As in proposed
work the concept of object oriented programming is considered, so the classification is done, whether it represents a class, a method or an attribute.

5.2.1 Pseudocode for Retrieval Process

Next step is to retrieve the best component from the repository and their effective representation. Heuristic search is used for representation, because it is one of the most important data structure which makes searching more refined and efficient. Then the user level keyword search is performed to check, whether the required component exists in the repository or not. This process is repeated for all files over the software system. Three step process is defined to perform the search.

Pseudocode 1- Classification of Components:

1. CExtraction
2. {
3. Input the program filename called FName
4. While s = Get Line(FName) <> null
5. If contains (s, “class”) 
6. {
7. Module = Module +1
8. Mname = Extract Module(s)
9. M = To Module (Mname)
10. Methods = GetMethods (M)
11. Total Methods = Total Methods + length (Methods)
12. Attributes = Get attributes (M)
13. Total Attributes = Total Attributes + length (Attributes)

14. Return (Module, Total Methods, Total Attributes)
15. }

This code is used for extraction of components from the required file which acts as a repository.

Pseudocode 2- Construction of Heuristic Search:

Heuristic Search (Component Name [], Component Type [], N)

/* Here N is total number of components. Component Name contains list of all components and component Type contains Type of components*/
This code constructs the heuristic search of all retrieved components from pseudocode 1.

The component extraction process under different classes is shown in these discussed pseudocodes. After extraction of classes, the method and data elements are also mapped under language constraints. Once all kinds of high and low level components are extracted, the relational mapping is done between these modules. Finally, design models for these systems are generated.

**Pseudocode 3- Searching Process from Heuristic Search:**
It performs searching of components received from the user heuristic search.
1. Search (Structure, Component)
2. {
3.   While (Structure <> null )
4.     Node 1 = get node (structure)
5.     Set Status = 0
6.     If Type (component) = Type (Node 1)
7.       {
8.         M = length (Node 1)
9.         N = length (component)
10.        While (N>0 and M>0)
11.          {
12.             If (Extract Heuristic (Node 1, <> Extract Heuristic (component, N)
13.               { Status = 1 }
14.             }
15.           } If status = 0
16.             { Print ‘Component Not Found’ }
17.           Else
18.             { Print ‘Component Found’ }
19.         }
20.       }

5.3 ONTOLOGY GENERATION PROCESS TO EXTRACT DESIGN MODELS

The categorization of components is done in terms of module level components, functional components and the data driven components. Now, each of the code file is processed one by one with specification of algorithmic method. Here, the code filtration and the keyword mapping are done for each line of code. At first stage of the model, components are extracted. Later on the component classification and the relational mapping is done. Based on relational measurement of these extracted components, the design diagrams are generated from the system. The ontology generation for the system is done to provide the structural representation of individual code files as well as for the complete software project. The functional model of component extraction is shown here in figure 5.3.
Figure 5.3 depicts the functional behaviour implemented on a complete software system as well as on individual code file to extract the component elements. The code level and language based check is used to retrieve these elements. Along with the component extraction, the classification of the components is also done. The classification of the extracted code elements is performed to identify the component class. These components can be module level components, functional components or the data elements. After obtaining the components and relative categories, the next work is to identify the relational observation between these components. Once, all the components from each code file and the relational behaviour is generated, the final task is to represent them in the form of a different design diagrams. The class diagram, object diagram and the entity relationship (ER) diagrams are generated to represent the system. These design diagrams are able to represent the structure of a software system.
5.3.1 Data Collection

In the proposed framework the developed tool is evaluated with the help of case studies comprising of five different types of software systems presented in chapter 3 table 3.2 in order to generate ontologies and produce design artifacts in the form of UML diagrams because UML notations are easy to understand and adopted by software industries as a standard to represent the design information of OOSD. UML diagrams provide reliable reverse engineering functionality [62]. Feasibility of the tool is analysed on multiple small codes and large code by extracting all the functional and non-functional artifacts from source code, which are missing in earlier developed tools. The evaluation is also performed to unveil unstructured code, dead code and unrequired code at the pre-processing level.

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Author &amp; Year</th>
<th>Reverse Engineering Tool</th>
<th>Input Language</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2009, Bosten and Klint [134]</td>
<td>DEFACTO</td>
<td>C, C++ and Java</td>
<td>Extracts basic terms: variable declaration, method calls and control flow</td>
</tr>
<tr>
<td>2</td>
<td>2010, Holger M. Kienle and Hausi A. Muller [60]</td>
<td>Rigi</td>
<td>C, C++</td>
<td>Function and Structure Data type are presented through call graph</td>
</tr>
<tr>
<td>3</td>
<td>2010, Auber and Weiskopf [135]</td>
<td>Solidsx</td>
<td>C, C++, and Java</td>
<td>Visualizes tree graphs and hierarchical edge in single IDE</td>
</tr>
<tr>
<td>4</td>
<td>2012, Bellucci, Ghiani et. al., [136]</td>
<td>Reverse Engineering Tool</td>
<td>Web Application</td>
<td>Transform web application into model based pattern</td>
</tr>
<tr>
<td>5</td>
<td>2013, Saha [137]</td>
<td>Code Structure Visualization Tool</td>
<td>Java</td>
<td>Represents hierarchical structure of system.</td>
</tr>
<tr>
<td>7</td>
<td>2014, Thongtanunam et al. [139]</td>
<td>ReDA</td>
<td>Web Application</td>
<td>Visualize code in form of Graph</td>
</tr>
<tr>
<td>8</td>
<td>2014, Reniers et al.[140]</td>
<td>Solid*</td>
<td>C, C++, Java</td>
<td>Visualize tree maps, annotated text</td>
</tr>
<tr>
<td>9</td>
<td>2015, T. Khan [141]</td>
<td>eCITY and IVA</td>
<td>C++, Java</td>
<td>Presents measurements results with software visualization</td>
</tr>
</tbody>
</table>
5.4 Results and Discussion
Various reverse engineering tools are available in the literature, but their functionality is limited. A comparison of some of the recent tools is provided in table 5.1. Tools are selected on the basis of input taken in terms of source code and produced outputs. Existing reverse engineering tools only able to find out structural relationships. They lack concrete description and algorithms to clearly represent an association, use, composition and aggregation between the software entities. Also, there are very few tools capable of recovering more than class diagram. When the reverse engineering process is performed, it is very complex to understand the code for the developer who see the very limited design information. So, design diagrams must need to display the details of data type, access mode of methods and types of relations. Most of the discussed tools also do not support entity relationship (ER) diagram, which are very important in code enhancement, reverse engineering and code ruse [142]. Also a class diagram generated by tools does not provide details such as attributes, objects, method functionality, access mode and data type. These details are very important to understand when migration of the old system is done to an upgraded system. Therefore, there is a need arises for a tool that can comprehensively perform reverse engineering activities.

The ODRET focuses on extracting multiple architectural view of existing software systems in the form of various design diagrams representing different aspects of a software system and attempts to remove above mentioned issues related to reverse engineering of software systems. The ODRET provides a simple .Net environment which can be integrated with other IDEs also such as NetBeans and enable study and experimentation of real C# applications. Our methodology enables reverse engineering and understanding of program by representing various ontologies related to the system and its design information. The main objective of this chapter is to present the graphical user interface (GUI) behaviour of the ODRET that automatically extracts the design models. To illustrate the various intermediate processing stages of the reverse engineering, datasets of Table 3.2 (discussed in chapter 3) are considered again to compare the results of RSA with ODRET and which depicts that along with the class information, it performs better in extracting other necessary artifacts. These extracted artifacts finally contribute in improving the accuracy for independent components generation process and in some of the software systems (Bomberman, Student Admission System and JChatBox), we are able to mine more number of components because of the detailed domain ontologies extracted using ODRET. Extracted elements of different software systems using ODRET are shown in table 5.2, and stepwise process is presented for one of the case Student Admission
System (SAS). The ODRET is having the effective support of a graphical interface for the integrated object oriented language and it provides the effective functioning on different data types.

Table: 5.2 Architecture Recovery Results for Subject Systems using ODRET and Comparing with RSA

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Subject System</th>
<th>No. of Files</th>
<th>No. of Code Files</th>
<th>Classes</th>
<th>Object(s)</th>
<th>Methods</th>
<th>Class Attributes</th>
<th>No. of Interactions</th>
<th>Extracted Components ODRET/RSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bomberman</td>
<td>50</td>
<td>31</td>
<td>23</td>
<td>8</td>
<td>15</td>
<td>32</td>
<td>29</td>
<td>15/14</td>
</tr>
<tr>
<td>2</td>
<td>Payroll management</td>
<td>46</td>
<td>28</td>
<td>10</td>
<td>4</td>
<td>10</td>
<td>19</td>
<td>23</td>
<td>5/5</td>
</tr>
<tr>
<td>3</td>
<td>Banking System</td>
<td>25</td>
<td>19</td>
<td>10</td>
<td>5</td>
<td>6</td>
<td>16</td>
<td>21</td>
<td>6/6</td>
</tr>
<tr>
<td>4</td>
<td>Student Admission System</td>
<td>41</td>
<td>27</td>
<td>21</td>
<td>3</td>
<td>8</td>
<td>18</td>
<td>32</td>
<td>14/12</td>
</tr>
<tr>
<td>5</td>
<td>JChatBox</td>
<td>61</td>
<td>40</td>
<td>29</td>
<td>7</td>
<td>22</td>
<td>23</td>
<td>39</td>
<td>9/7</td>
</tr>
</tbody>
</table>

Figure 5.4 Extracted and filtered project files from a software project
At the higher level, the contents are extracted in the form of project files. In first list box, all the files of SAS including the code file, the design file and configuration files are extracted. In second list box, the filtration is applied and only the code files of the SAS are extracted. The descriptive information from the files is also derived to generate the component description.

Figure 5.5: Extracted class information

Figure 5.6: Extracted object information
After extracting the code files from software SAS, the next stage is to identify the existence of various classes used as definition or the members. The figure 5.5 is showing the class button to extract the class information. It shows the extraction of the internal class elements. The existence of various objects used as a class member also extracted. Figure 5.6 shows the object button to extract the integrated objects from the software system. It presents extracted objects with relative description. The process of object generation with relative class specification is also visualized.

![Image of class button](image)

**Figure 5.7: Interaction (Methods) based extraction along with its functionality**

The actual process level integration between two objects or modules is implemented with function level access. The function calling and the function utilization of one module or class in another is done using method call. The method call can be with or without an object specification. In figure 5.7, the object level method access is defined. It depicts a method button to extract all the method calling used in the particular code. To understand the system level structural description, it is required to identify the features of the class modules as well as its relationship with other classes and modules. Figure 5.8 depicts such extracted relationship and their cardinalities. The relationship is based on the object integration between the project modules. The relationship is identified from other class modules exist in the same project or taken from other projects as the libraries.
Once the extraction of structural components of software systems is completed, the next job is to organize and relate this information to generate the ontology components. The design time components or the structural generation is formulated to design models. The ontology specification of these design components is required in terms of inclusive members and processes. In figure 5.9 the structural components and relationship between these modular components is presented as the ontology information. The first level ontology composition and utilization is described by generating object diagrams. The object diagram discusses about the specification of relative components, elements and associated members. Different blocks in the framework represent the component specific extraction. The class components and the relative use case responsibilities along with the generated classes and the associated members of the project are displayed in figure 5.10, figure 5.11 and 5.12.
Figure 5.9: Generated object diagram

Figure 5.10 Generated UML models: Class diagram
After the ontology components are extracted for the software project, now the different design time diagrams are generated such as class diagram, Use case diagram and ER diagram with specification of inclusive elements. The class diagrams generated using RSA (discussed in Chapter 3) did not yield a picture that gives easy interpretation, because if the code was not well structured, then design diagrams does not exhibit entities relationship and dependencies much more obvious. Whereas, diagram generated using ODRET provides clear and unambiguous methods, attributes, relationships and dependencies among entities.

5.5 Summary

There are number of reverse engineering tools available in both industry and academia designed according to their needs. Apart from the commercial architectural tools available in the market, the research community has also developed numerous tools. Commercial tools are mostly developed to be used as it is, while research tools are open source that can be customized according to the user requirements. Despite the automated help of reverse engineering tools, some software industries prefer to re-engineer legacy code from scratch without having visualized model or some of them reverse engineer the code into UML and then forward
engineer it manually. Such UML diagrams are typically of little use, unless the extracted information is organized into discrete components or package. The contribution of this chapter is to provide a comprehensive methodology which facilitates all the needed steps to understand a system through reverse engineering. Our evaluation of the ODRET on various software projects revealed, its abilities to filter required and unrequired code that helps to identify the useful artifacts. Design models generated using ODRET are concrete in nature. Ontology produced as intermediate stage also reduces the efforts for understanding the inner structure and elements of the entire software system.