Migration Strategies for Object Oriented System to Component Based System

Chahat Monga¹, Ms. Aman Jatain² and Dr. Deepti Gaur²

¹Research Scholar, ITM University, Gurgaon, India
²Assistant Professor, ITM University, Gurgaon, India

ABSTRACT

Migration of object oriented system to component based System is not an easy task, not only technically a lot of changes needs to be done but also numerous other issues needs to be kept in mind. However Component based Software development has been gaining its popularity from the past few years and has higher reusability scope. Programs built using CBSE approach are confirmed to be suitable to new environments. These days it’s a universal practice to reuse components in project to achieve better quality and to save time. So moving to CBSE from object oriented seems wise decision. Number of approaches has been introduced to implement this and each one of them has its own pros and cons. The paper focuses on the brief review on works of different authors in this area from the year 2000 to 2014.

KEYWORDS

Component based, clustering, graph, object oriented, reverse engineering, ROMANTIC, UML.

1. Need of Transformation

Most old object oriented systems commonly known as Legacy systems did not foresee the change in speed of modern projects. Due to this they undergo a lot of limitations; e.g.:

a) Large maintenance costs.
b) Difficult to train employees about old methodologies
c) Difficult to adapt and evolve according to new environment [5].

However looking to the other side of the coin legacy systems could not be rejected altogether. They provide vital information and functionality that has been used for years and remains crucial for smooth working of the project. Thus legacy systems can’t be dumped and at the same time could not be used as it is for a long time. That is why need is felt to reverse engineer the object oriented legacy system to convert it into components that are both reusable, maintainable and efficient to use. Such a system would combine new technology with existing one and also could be easily adapted to the future changes. Components, due to their high cohesiveness have better applicability than object-oriented and also coupling between them is low making them less complex [10]. Thus it is evident that most of the drawbacks of object-oriented system could be easily overcome by components.

Rest of the paper is aligned as follows. Section 2 discusses the inclusion and exclusion criteria and also brief introductions to the selected techniques. Section 3 shows the bubble graph used to represent the studies and finally Section 4 gives conclusions and directions for future work.
2. Inclusion and Exclusion of Search Results

An extensive search on various research papers was done to retrieve the ones those contain the solution to our problem statement. For this activity a set of keywords is generated which consists of search terms that are similar to our research domain. Some of the various search terms used for data retrieval are:-

a) Migration  
b) Object oriented  
c) Component based  
d) Transformation  
e) Legacy  
f) Software development  
g) Reverse engineering  
h) Reengineering

These search terms were combined with Boolean expression in the form of A or B , A or C, A or B or C, D or B or C, A and E and C and F etc. which provided a list of research papers related to our topic of interest. Number of research papers retrieved was enormous so we included only those studies which presented fully implemented techniques for transformation of OOS to CBS. Studies which were not fully implemented or only provided theoretical basis were excluded in this paper. The studies that were published before year 2000 were also excluded. Also the studies which represented similar techniques only the most significant one is included and rest are excluded. The table 1 shows our detailed search strategy:-

TABLE: 2.1. Search Strategy

<table>
<thead>
<tr>
<th>Publishing</th>
<th>2000 to 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browsing libraries and sites</td>
<td>IEEEExplore, ScienceDirect, Springer, ACM Digital Library, Google</td>
</tr>
<tr>
<td>Items searched</td>
<td>Conference and journal papers</td>
</tr>
<tr>
<td>Publication Language</td>
<td>English</td>
</tr>
<tr>
<td>Searched for</td>
<td>Full text</td>
</tr>
</tbody>
</table>

Using the above inclusion and exclusion criteria the following papers (table 2) are found to be fit for our purpose and are discussed in subsequent sections year wise. [TABLE 2]

TABLE: 2.2. Techniques and Tools

<table>
<thead>
<tr>
<th>Year</th>
<th>Author(s)</th>
<th>Methodology</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>2003</td>
<td>Eunjoo Lee et al</td>
<td>CLUSTERING</td>
<td>-</td>
</tr>
<tr>
<td>2005</td>
<td>Suk Kyung Shin &amp; Soo Dong Kim</td>
<td>REFACTORYING</td>
<td>-</td>
</tr>
<tr>
<td>2005</td>
<td>Hironori Washizakia &amp; Yoshiaki Fukazawab</td>
<td>FORMAL METHODS</td>
<td>Rigi, Refactorit</td>
</tr>
<tr>
<td>2008</td>
<td>Sylvain et al.</td>
<td>UML Diag, CLUSTERING</td>
<td>-</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Methodology</th>
<th>Tools/Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008</td>
<td>Hassan Mathkour et al.</td>
<td>ROMANTIC</td>
<td>Code profiling tool, Sparx, ArgoUML</td>
</tr>
<tr>
<td>2011</td>
<td>Dan Li et al.</td>
<td>UML Diag</td>
<td>R cose CASE tool, QVTR-XSLT</td>
</tr>
<tr>
<td>2012</td>
<td>Simon Allier et al.</td>
<td>Reverse engineering</td>
<td>Tracer, CBAExtractor, CBAToOSGi</td>
</tr>
<tr>
<td>2012</td>
<td>Selim Kebir et al.</td>
<td>Reverse engineering, CLUSTERING</td>
<td>-</td>
</tr>
</tbody>
</table>


This paper was presented by Lee et al. in the year 2003[1]. They divided the whole process into 2 sections. In first section they used relationship like inheritance and composition etc to create the basic components. In second step they refine this component system generated using software metrics like cohesion, connectivity strength etc [3].

1) Basic component creation:- under this section four steps were used to create an intermediate product i.e. component system steps involved are 1) functional units are created from the classes having composite relationship 2) unnecessary and redundant classes are removed. 3) Classes having inheritance relationship between them are put into same component as they have more strong cohesion. 4) Once again we will check for Needless classes that were not called by any outside class or method and discard those classes. The resultant of these steps would create an intermediate component system.

2) In addition to the component System that is generated in previous steps, connectivity strength, component complexity and cohesion are also defined. A clustering algorithm is then applied to cluster the similar components into one big component. Usually hierarchal clustering is employed. The final result will provide the components that are generated from initial object oriented code. The whole procedure is shown in figure 1.

![Figure 2.1 Steps involved in Lee et. al methodology](image)

81


2.2. Washizaki et al (2005)

Given by Washizaki and Fukazawa in 2005 it is based on REFACTORING technique called “extract component”[3]. The process starts by defining a CRG (class relationship graph), the main thing to be considered while creating CRG is that only the classes created by user become part of CRG, java core classes do not participate in the process.

Next a clustering algorithm is generated to find clusters in the object oriented code, clusters are basically the independent nodes that do not have any link outside the cluster. For determination of clusters a term called “reachability” is defined. The paper also presents an extraction algorithm which supports clustering and the retrieval of desired components from those clusters. After the components have been created the part outside the components is refactored so that these freshly extracted components could be used independently and any situation of ambiguity does not arise. This refactoring is named “extract component” by the authors and contains about 17 steps [4] after which the output are independent components which could be reused frequently. To automate the whole process the authors also created an automation component extraction system in java so that instead of performing the transformation in batches it could be done in one go. Figure 2 clearly depicts the main steps involved in the process.

Figure 2.2 Steps involved in Washizaki et al methodology

2.3 Suk Kyung Shin et al (2005)

This method was given by Suk Kyung Shin and Soo Dong Kim in the year 2005[3]. This approach basically utilized formal method specification / object z technology for transformation. The paper talks about the meta model of object oriented design after which the meta model modal and finally a set of rules for migration of formal spec of OOD to formal spec. of CBD are given. Using these rules the final transformation is made. The final components could be used in EJB, CORBA or .NET.

2.4 Mathkour et al (2008)

This technique is given by Mathkour et al. [4] in 2008 on the basis of UML diagrams and clustered graph generation. They used JAVA language as their base to make the entire system platform independent. UML unified modeling language was utilized for specification work and XMI (XML based interchange language) was used to exchange file formats as it supported by almost all UML tools. The main phases of transformation involved in the conversion of object oriented to component based are depicted in Figure 3:-
2.5. Sylvain et al. (2008)

This approach was proposed by Sylvain and Mourad in 2008 using ROMANTIC APPROACH [5]. This approach was quite different from previous works of authors as it included architectural semantics and also reduces the need of human expertise which further helped to reduce overall cost of the whole process. Sylvain and Mourad used two principles as their guide through the whole process. The principles used by them are diagrammatically represented in Figure 4. Principle 1 used in Romantic approach that is architecture extraction starts by mapping object concepts to architectural concepts like mapping classes, packages etc to components, interfaces, connectors etc. and thereby generating a correspondence model, using this model they partitioned system classes. And the elements under this partition are called “shapes” representing components. Thus actual requirement was to assimilate these components to shape. These shapes form the architecture. Principle 2 validates the previously generated architecture and also instantiate it. Four guides were considered for this purpose

- Ability to adapt to deployment hardware.
- Correct semantics.
- Quality properties.
- Adhere to documentation constraints.

And finally the semantics of architecture are checked for correctness and this leaves us with required model.

2.6. Mishra et al. (2009)

This technique is presented by mishra et al in 2009 and named it CORE [6]. Component oriented reverse engineering creates reusable software components by using reverse engineering. A tool named rational rose is used to reverse engineer the system for identifying the relevant use cases for the system. These use cases form the base for generation of component structure. Once the use cases are identified a CRUD matrix (create or read or update or delete) is created which
represents structural relationships. As a final step to extract components we apply some clustering techniques according to dependency relationship. Final reusable components are stored in component bank after testing.

2.7. Dan Li et al. (2009)

This technique was given by Dan Li et al. in 2011 in their paper “Interactive Transformations from Object-Oriented Models to Component-Based Models” [7]. The technique makes use of UML diagrams like class diagram and sequence diagrams for the transformation process. First the extraction of object sequence diagram is done, then transformation is applied to object sequence diagram to convert them into component sequence diagram using a validity tool called QVTR-XSLT. At every step a set of object lifelines are chosen that the user intends to change into component and then some checks are made using algorithms to ensure that selected lifeline would qualify as component lifeline [figure 5]. With every component lifeline the corresponding component is sent to component diagram and a protocol is generated for it. After all the object lifelines have been processed we get our desired components extracted from object-oriented code.

![Figure 2.5. Steps involved in Dan Li et, al methodology](image)

2.8. Simon Allier et al. (2012)

Simon Allier et al. proposed this methodology in 2012 which converts object oriented applications to component oriented applications [8]. The whole process revolves around two steps: 1) identifying components 2) identifying required interfaces. In the first step that is component identification we first determine the “core components”. Next the missing classes are added by using static call graph and finally manual refinement of generated classes is done. After this process we over we are left with the main components which creates component based architecture. Further to complete the transformation component based architecture is mapped to component oriented architecture using object oriented concepts [Figure 6].

![Figure 2.6. Steps involved in Simon Allier et, al methodology](image)
2.9. Selim kebir et al. (2012)

It is a component identification methodology proposed by selim kebir et al in 2012[9]. This methodology uses the concept of object oriented language and clustering to convert object oriented application into component oriented application. At first a mapping model to map object concepts to component concepts is proposed. This model also evaluates the semantic correctness of the components. Next the interfaces are identified and a clustering algorithm is utilized for grouping the relevant components together and as a last step everything is documented. The authors applied the technique on two systems 1) JDOM 2) Apache HTTP. This methodology seemed quite similar to the one proposed by Simon Allier et al discussed above.

3. Bubble Graph

The techniques extracted from above discussion were analyzed to answer various research questions, for this purpose bubble plot is utilized to characterize and represent the studies. In Figure.7 the X axis denoted the publication year of the studies i.e.2003, 2005, 2008 etc and y axis denotes the research strategy adopted in them like UML, ROMANTIC, REVERSE Eng. etc. The intersection of the axis shows bubbles with contains the reference number of the studies. The bubble graph compares and diagrammatically represents which technology is most suited and widely used for transformation purpose. The size of the bubble is according to the number of studies covered by that bubble. We have total nine techniques to be compared so relative percentage occurrence of techniques is CLUSTERING=22.2%, FORMAL METHOD=11.1%, REVERSE ENG / REFACTORING=44.4%, UML= 11.1%, ROMANTIC=11.1%. These figures show that the technique that is most widely utilized is reverse engineering or refactoring.

![Bubble graph](image.png)

Figure: 3.1. Bubble graph
4. CONCLUSION and FUTURE SCOPE

Component based software development has emerged as a powerful technique and has been widely adopted these days in most of the organizations for software development. In this paper we reviewed various technologies that are in practice to convert object oriented legacy applications to component oriented applications. For instance nine techniques that have been practically implemented are discussed. Many approaches have been introduced so far and the study is still evolving. A brief introduction and working mechanism is also provided with each methodology and in last a comparative analysis is done using bubble graph as an overall extract of the research. Through this paper we found five important techniques for transforming object oriented system to component based system.

1) Reverse engineering/refactoring
2) Clustering
3) Formal methods
4) UML approach
5) Romantic

Also it is observed that most of the techniques employed for this purpose uses reverse engineering/refactoring approach for the transformation. There is a lot of future scope and chance of evolution in this field, so far a number of techniques have been offered with their own pros and limitations so in future a new technology could emerge as a concrete benchmark for this transformation and system could also be automated.

REFERENCES


