CBPR: CASE BASED PATTERN RETRIEVAL – A FRAMEWORK FOR PATTERN RETRIEVAL

Kumar K. and Saravanaguru R. A. K.
School of Computing, Science and Engineering, VIT University, Vellore, India
E-Mail: kunschy@gmail.com

ABSTRACT

Nowadays, existing patterns are available in the form of catalogs. These catalogs, however, are always searched by a pattern name. Searching for patterns using their name is useful for only experienced analysts who already knows what patterns they need. Inexperienced developers, on the other hand, may have to spend time to read throughout each pattern description before they can find the right patterns [18]. So it is necessary to have systematic framework to help inexperienced developer to search for appropriate pattern based on their design problems. The proposed system addresses the problems which are mentioned above. It uses the case based reasoning techniques to retrieving the patterns and uses formal concept analysis for organizing the cases (design problems). Along with this case weight is added to intent (keywords identified from the problem) so that the ranking efficiency of the system can be improved. CBPR based tool has been adopted the proposed framework and compared with three types of applications.

Keywords: requirements engineering, design patterns, problem frames, similarity scoring.

INTRODUCTION

Design is one of the important steps in software development. A design pattern is a general reusable solution to a commonly occurring problem in software design. Experienced software developers and researchers develop and record solutions to design problems, which occurred again and again in many contexts, as reusable common design solutions and call them design patterns. Patterns originated as an architectural concept by Christopher Alexander (1977/79)[9]. Design patterns gained popularity in computer science after the book Design Patterns: Elements of Reusable Object-Oriented Software was published in 1994 by the so-called “Gang of Four” [2]. That same year, the first Pattern Languages of Programming Conference was held and the following year, the Portland Pattern Repository was set up for documentation of design patterns. Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues [10] that can cause major problems, and it also improves code readability for coders and architects who are familiar with the patterns. The CBPR (Case based Pattern Retrieval) is a framework which helps the developer/analyst to find appropriate solution (pattern) to their design problems.

RELATED WORK

When experts work on a particular problem, it is unusual for them to tackle it by inventing a new solution that is completely distinct from existing ones. They often recall a similar problem they have already solved and reuse the essence of its solution to solve the new problem. Patterns are general reusable solutions to recurring design problems in an application-independent way as suggested by [10], [2] and [11]. To accommodate the major principle of critical / real time system such as interoperability these design pattern favors a lot [12]. Though the patterns exploit much of reusability, still it remains to yield poor system quality due to its ambiguity as pointed by [13]. Also these patterns helps in the early evaluation of design which highly essential for any system [14]. The concept of pattern aims at supporting designers and system architects in their choice of suitable solutions for commonly recurring design problems [15]. The framework contains the description of the pattern structure and representation that can be used to model the agents in specific application [16].

To address patterns came as catalogs which is electronic readable and searchable form like MVCASE [3], this made little easier to search for the patterns. This increased the productivity of the development process to a considerable level. The main problem of using these forms is developers necessary to understand clearly about detail of design pattern because they have to use the pattern name for searching. In this way, inexperienced developers need to spend time to study entire existing design patterns before capable to select the pattern for solving the design problems [17]. The experienced developers may also need tool because they have to learn new design patterns that are created continually. This created a situation need for new searching tool. The model of Gustavsson and Ersson [4] presented an idea for indexing the patterns based on formalizing their intent specified by the GoF design pattern template [1]. Petri and Csertan [5] used the same template [1] to build a meta-model of design patterns described in PCML (Pattern and Component Markup Language) that is suitable for simplified searching of design patterns. They provided appropriate format of design patterns has to be elaborated that can also contain the summary. The summary has to be created for every pattern based on their informal sections. The search method has to be worked out. The main
advantage of the verbs and subjects are paired. Several verbs can belong to a design pattern (or any class or object of it) and several subjects can be ordered to every verb and vice versa. The ability to query properties of design pattern or any class and component inside a pattern is done in a standard way. But this method suffered with the problem of the process of collecting the properties of a pattern and the collaborating classes has to be done manually. This means extra work has also to be done in the case existing, already documented design patterns.

Later the Intakosum and Muangon [6] applied the ideas mentioned above along with the modern information retrieval techniques and developed retrieving model. The model is used to allow developers to enter keywords to search for the proper design patterns for the design problems that they are going to solve. Index of each document is developed and index weight is calculated by analyzing and restructuring design patterns descriptions, and gives the priority to each structure component. This has an advantage that developers can then search for the appropriate design patterns by entering keywords from a problem domain context, the system responses with the patterns that are related to the keywords, ranking is done by their similarities to the keywords. This method has a limitation that result is dependent on the matching between document and query indexes, and the index weight [18]. This created major two problems that are the creation of indexes is done manually so it leads to ‘keyword search’ problem and ranking problem is occurred in some queries. The model proposed by the same authors later [7], this model used information retrieval along with the case based reasoning techniques. This addressed the problem of ranking. But this model still suffered with problem of absence of learning. So these authors again gave a model [8] which used case based reasoning for retrieving patterns and they added learning by organizing the cases using formal concept analysis (FCA). The technique analyzes a case base to obtain two useful knowledge resources [20]. One is an index lattice structure that is flexible enough to add new indexes into the case base. Another, we obtain dependency implication knowledge that is used to suggest appropriate indexes for new problem. Even though the problems are addresses but still refinement is needed in retrieving patterns. In this research same case based reasoning and FCA is still used along with the index weighting is added. The patterns are ranked based on both FCA and index weight. This will increase the efficiency of retrieving tool.

From the above discussion it is clear that there is need for new framework which must addresses the problems of referring books and electronic catalogs, which do not needs the prior knowledge about the patterns, and must not suffer with problems of ‘keyword search’ and ranking with less manual work. And it must be more efficient so that even an inexperienced developer can also develop high quality system effortlessly. For that purpose the system first parses the design problem of the developer and extracts the intents. Then it compares with the intents of the existing cases using similarity scoring technique. Based on the total score the similar cases are ranked. The case with highest score is retrieved and displayed as a solution.

**SYSTEM ARCHITECTURE**

The main purpose of system architecture is to offer guidance to making the task of creating and maintaining a complex system easier. Procedural decomposition is used to decompose the system into entities and those entities were decomposed further in subordinates based on functional decomposition. The identified modules are Problem Case Intent Extraction (M1), Similarity Scoring (M2), Ranking (M3), Similar Case Extraction (M4) and Formalized Case Base (M5).

![System architecture](Figure-1. System architecture)

<table>
<thead>
<tr>
<th>ID</th>
<th>Purpose</th>
<th>Dependencies</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>For extracting new intents from the problem case</td>
<td>M1 calls M2</td>
<td>New intents</td>
</tr>
<tr>
<td>M2</td>
<td>Retrieving best case with the help of similarity scoring</td>
<td>M2 depends on M1, M5</td>
<td>New intents, Solution cases</td>
</tr>
<tr>
<td>M3</td>
<td>For providing ranking</td>
<td>M3 depends on M1</td>
<td>Solution cases, Ranking Feedback</td>
</tr>
<tr>
<td>M4</td>
<td>Reorganize the similar case with analyst</td>
<td>M4 depends on M3, M5</td>
<td></td>
</tr>
<tr>
<td>M5</td>
<td>To provide available case list in the formalized case base</td>
<td>M5 depends on M2, M4</td>
<td>Data in case base</td>
</tr>
</tbody>
</table>

**ProBLEM CASE INTENT EXTRACTION**

For the given problem case, the system tokenizes the problem and it will save it in a separate array. Then each token will be compared with the array which contains the unwanted words. It deletes all the words which match with the unwanted array. Then the other tokens are passed
for retrieving the synonyms and compared with the intents in the case base. Then it retrieves those intents which are matched and its corresponding concepts and cases belongs to those concepts are also retrieved. These details are stored in a separate array.

**Similarity Scoring**

The similarity scoring is done based on the count of number of intents matched with the concepts. Then the concepts are ranked based on the count value that the concept with highest count value is ranked first. Then the top rank concept is identified. Then the case with maximum weightage is identified. For the problem case number of the solution case is retrieved and pattern that belongs to that case is identified. Once the pattern is identified, its details are retrieved from the case base.

**Ranking**

The solution case which is identified by the similarity scoring is used to identify its corresponding concept and using the super concept relation the concepts which are above the hierarchy is identified.

**Similar Case Extraction**

The super concept’s patterns and intents are retrieved and suggested to the user based on its weightage. Then the feedback will be retrieved from the analyst and if the feedback is positive then the weightage for that case will be increased. If the feedback is negative then the system will redirect and refine the search based on the suggestions provided.

**Formalized Case Base**

The case bases have to be updated according to the arrival of the new patterns. The system will provide a form to add new pattern into the case base and the system also provide option to view pattern details.

Formal concept analysis is a principled way of automatically deriving ontology from a collection of objects and their properties. The term was introduced by Rudolf Wille in 1984, and builds on applied lattice and order theory that was developed by Birkhoff and others in the 1930s.

A (formal) context consists of a set of objects O, a set of attributes A, and an indication of which objects have which attributes. Formally it can be regarded as a bipartite graph $I \subseteq O \times A$.

A (formal) concept for a context is defined to be a pair $(O_i, A_i)$ such that

1. $O_i \subseteq O$
2. $A_i \subseteq A$
3. every object in $O_i$ has every attribute in $A_i$
4. for every object in $O$ that is not in $O_i$, there is an attribute in $A_i$ that the object does not have
5. for every attribute in $A$ that is not in $A_i$, there is an object in $O_i$ that does not have that attribute

$O_i$ is called the extent of the concept, $A_i$ the intent.

**CBPR (CASE BASED PATTERN RETRIEVAL) FRAMEWORK**

It is a Web-based application executing on a Web server and connected to case base which is formalized. As its architecture shown in Figure-2, CBPR accepts and processes requests from two patrons: Analyst/developers and authors/administrator. The system is expected to have a Web user interface. Its release has merits of being efficient and precise in searching solution for new design problems and easy to add new patterns and modify the existing patterns.

**CBPR Process**

CBPR framework realized with the help of CBPR tool which take case (design problem) as an input and provides the solution by the way of selecting the pattern. The tool tokenizes the problem and save it in a separate array. Then each token will be compared with the array which contains the unwanted words. It deletes all the words which match with the unwanted array. Then the other tokens are passed to the JAWS (Java Wordnet Searching) API for retrieving the synonyms from the wordnet dictionary.

These tokens are called the intent which is matched with the intent in case base. If any intent of the problem matches with the intent cases, those patterns id will be retrieved. The pattern with maximum intent matching length will be considered for ranking. Finally, solution pattern will be given for the problem case.
Comparative Assessment

There are three different types of applications like web, embedded and small scale applications considered for comparative assessment against the CBPR Tool. Its performance was improved with CBPR. The results show that the percentage of work achieved with the same time with CBPR tool and without CBPR tool.

![Figure-3. Comparative assessment graph.](image)

CONCLUSION AND FUTURE WORK

The CBPR framework based tool successfully retrieves the pattern for various design problems of the analyst. The system effectively learns the new intents based on the analyst feedback. After analyzing the results it becomes clear that the CBPR (Case based Pattern Retrieval) saved the time of the user by providing best solution to their design problems and made even an inexperienced developer/analyst to develop a system with better quality.

Data mining clustering algorithm can be included so that similar patterns can be grouped together, that will increase the effectiveness of the case retrieval [19] and it will also help in proving suggestions to the analyst. Learning can also be included while adding new patterns which help to identify the unwanted words and extract appropriate intents from the problem. Despite the fact that the case weight has been added along with intent weight can be included so that retrieval processes can have much more clarity.

REFERENCES


