Automatic Use Case Flow Pattern Generation Using Verb Clustering

Deokyoon Ko\textsuperscript{1}, Sooyong Park\textsuperscript{1}, Suntae Kim\textsuperscript{2*} and Mansoo Hwang\textsuperscript{3}

\textsuperscript{1}Dept. of Computer Science \& Engineering, Sogang University, Seoul, South Korea
\textsuperscript{2}Dept. of Software Engineering, Cheonbuk National University, Jeonju Si, Jeollabuk Do, South Korea (Corresponding Author)
\textsuperscript{3}Electronic Engineering, Shinhan University, Seoul, South Korea

\textsuperscript{1}\{maniara,sypark\}@sogang.ac.kr, \textsuperscript{2}\{stkim\}@jbnu.ac.kr, \textsuperscript{3}\{mshwang\}@shinhan.ac.kr

Abstract

Software requirements completeness is one of the key elements for successful software development. Incomplete requirements can cause one to misunderstand requirements and eventually build a wrong system as a developer interprets them in an ad-hoc manner. In order to handle this issue, the use case pattern is suggested, which is a set of commonly discovered scenarios in the use case specification. However, identifying the use case pattern is not systematic and comprehensive without providing any algorithmic or statistical rationale. This paper proposes an automatic approach to identifying the use case flow pattern from various use case specifications. In this paper, we gathered 83 use case specifications from eight industrial systems, and presented two use case flow patterns identified from the use case specification.

Keywords: Requirement engineering, Use case scenario, Requirement patterns

1. Introduction

Handling software requirements is the most critical step at the earlier phase of the software development. Among the various characteristics for better software requirements, completeness is one of the key elements, because incomplete requirements cause one to misunderstand requirements and eventually build a wrong system [13, 6]. However, completeness of software requirements is usually achieved by experts' intuition and their experience in reality [7]. In order to handle this issue, use case pattern is suggested, which is a set of commonly discovered scenarios in the use case specification [1]. However, identifying the use case pattern is not systematic and comprehensive without providing any algorithmic or statistical rationale. Thus, it can guide only limited parts of scenario authoring.

There has been some researches to address above issue. A. Mahfouz et al. [9] introduced the requirement patterns and their relationships for Service-Oriented Computing and S. Konrad et al. [8] suggested the problem models and patterns for embedded system using UML notations. R. Biddle et al. [1] shows the dialogue patterns for essential use case. There are two approaches for generating patterns from the domain knowledge. S. Robertson [12] suggested the extracting approach for common event/use case pattern from multiple event/use case models. S. Ketabchi et al. [7] provided the approach for generating domain business rules which called Norm, common problems and their solutions from domain models.

This paper proposes a machine-learning based approach to automatically identifying the use case flow pattern from existing use case specifications. It is composed of three
steps. First, the agents and main verbs are extracted from each scenario in use case specifications. Then, the verbs are divided into high and low frequency groups in terms of their occurrences. The verbs in the high frequency group are clustered in terms of the semantic distance between the verbs, and the use case flow graph are generated from the semantic link between the agent-verb sets. Finally, we identify the use case pattern from the use case flow graph. In this paper, we gathered 83 use case specifications from eight industrial systems, and presented two use case flow patterns identified from the use case specification to demonstrate our approach.

The remainder of the paper is organized as follows: Section 2 presents the background on software requirements patterns and previous researches for mining patterns from requirements specifications. Section 3 introduces our approach to generating use case flow patterns from existing use case specifications. After Section 4 presents case studies to illustrate our approach, we conclude this paper and also discuss future work in Section 5.

2. Background and Related Works

This section introduces software requirement patterns that is frequently discovered in requirements specifications, and presents previous researches to identify the patterns.

2.1. Patterns in Software Requirements Specification

The design pattern is a set of reusable building block for software design, and a widely used approach to delivering design knowledge to design beginners or software developers [5]. While the design pattern is a generally accepted, software engineering community tries to identify various types patterns discovered in a whole software development lifecycle [8] such as architecture patterns [2], analysis patterns [4] and software requirements patterns [15]. Among those patterns, requirements pattern is defined as “an approach to specifying a particular type of requirements” [7]. They provide various benefits to requirements authors as well as readers. The benefits are summarized as follows [7, 15]:

- **Readability**: Widely accepted requirements are easy to understand and accept rather than domain specific requirements.
- **Completeness**: Authors can realize missing parts of the scenarios, and it helps them to complete the scenarios with various alternatives.
- **Efficiency**: Use of the patterns enables one to save time and efforts, because the patterns provides the template to write scenarios. Thus, authors do not need to write scenarios from the scratch.
- **Reuse**: The requirement patterns can be used in other projects. Also, well defined patterns provide the guidance in developing software design, code and test case patterns.
- **Requirement validation**: As requirement patterns covers diverse alternatives or actions, it is used to validate the requirements.

Several researchers introduced requirement patterns. A. Mahfouz et al., suggested the requirement pattern for developing service-oriented computing (SOC) based systems [9]. They suggested eleven requirements patterns (e.g., *Barrier*, *Co-location*, and *Deadline*) and their relationships such as *Query* and *Retry*. S. Konrad et al. suggested requirement patterns for embedded systems [8]. They extracted ten patterns and their problem models in the UML class diagram. These two studies exemplify domain-specific requirements patterns relying domain knowledge.
On the contrary, general requirement patterns are also proposed. R. Biddle et al. introduce the essential use case dialog patterns discovered in use case specifications [1]. They suggested six patterns based on the sequence of the use case scenarios and its intentions. The following summarized the six essential use case patterns.

- **Request**: When a user requests information to a system, the system provides requested information.
- **Monitor**: A system displays its information to a user.
- **Alarm**: A system warns a user.
- **Command**: When a user requests to modify information, a system performs the request.
- **Prompting**: A system offers a prompt for a user to enter his/her decisions.
- **Confirming Step**: When information is updated, a system and a user confirms the updates.

While these patterns introduces conceptual approaches for characterizing requirements patterns, these come from expert’s intuition and experiences. Thus, the patterns cannot be validated in terms of its coverage and accuracy, so that mining the patterns from requirements specification is necessary. Through the approach, it is possible to extract domain specific or general requirements patterns.

### 2.2. Methods for Extracting Requirements Pattern

There has been some work for extracting requirements patterns. S. Robertson [12] suggests a method to identify requirement patterns from a set of events and use cases. It assumes that use cases are invoked from external events, and there exists the similar events and use cases in a specific application domain. In the approach, it first depict the event and use case models to achieve diverse business goals. Then, the models are consolidated. During this step, similar events and use cases are grouped and abstracted, and eventually requirement patterns composing of process and data patterns are gathered. In a book shop application, as an example, the “buying book” and “buying consulting service” events can be grouped and abstracted into “Customer wants to buy a product”. In this way, this paper provides the domain specific event/use case model from application event/use case models.

S.Ketabchi et al. [7] provides requirement pattern generation process from a domain model. In this approach, requirement and problem patterns are built from a domain model and stored in the repositories. The problem patterns indicate the common business rules in the business domain such as the business regulations, conditions or restrictions. The requirement patterns imply a set of software requirements that handle each business problems. The requirement is expressed in norm, and stored in the requirement pattern repository that can be reused in a similar business domain.

In order to identify the problem and requirement patterns, a domain analyzer first elicits stakeholder model for identifying stakeholder and their roles, process model for analyzing interactions between the system and stakeholders, and norms repository storing common functional and non-functional requirements. Through the models, it builds a link between a process indicating a business goal and a norm handling the process.

These researches only propose a manual process for identifying requirement patterns from application specific requirements specifications without mentioning automatic or semi-automatic ways. Thus, the quality of domain modeling is critical to identify the patterns. Also, though the patterns are identified, it only can be applied into the similar application domains. This is mainly caused by the lack of automatic ways to extract requirement patterns from a set of requirements specifications.
3. Pattern Generation Method

This paper proposes an automatic approach to identifying the use case flow pattern from existing use case specifications. Figure 1 shows an overview of the suggested approach. In order to identify use case flow pattern, a set of software requirement scenarios is a prerequisite. Given the scenarios, an agents such as a user or a system are identified at first, and the main verbs of each scenario are automatically extracted by the natural language process (NLP) parser. Then, the verbs with its agent are separated into two groups in terms of its frequency, and cluster together in each group according to the semantic distance. After then, the directed graph is built based on the sequence of the agent and the corresponding main verbs, containing weights depending on the appearance of the scenario. From the directed graph, the use case flow patterns are extracted as an output of our approach.

Figure 1. An Overview of Pattern Generation Method

3.1. Extract Main Verbs of Scenario

This step intends to extracts a main verb of each scenario. We assumes that the agent such as a user or a system is separately entered by a scenario author. The main verb of each scenario is obtained by the NLP parser. We applied the Stanford NLP parser [13], which is the popular and well documented tool for supporting NLP parsing, to manipulate the elements of a natural language sentence. The NLP parser produces the parsing tree of a sentence composing of the Penn Treebank (e.g., VP denoting a verb phrase). Once the parsing tree is generated (see Figure 2), our algorithm presented in Algorithm 1 traverses the tree from the root node. During traversing, if the node value starts with V, then it keeps traversing to the leaf node. When the traversing is finished, the leaf node turns out to be the main verb.

Figure 2 presents an example for extracting a main verb of a natural language sentence obtained from Payroll System. There might be more than one verb in a sentence (e.g., request and specify in Figure 2). Algorithm 1 handles this issue by traversing each node only starting with V* from the root node, once it meets a node that starts with V. Thus, it first starts with ROOT followed by S, VP and VBZ. When it meets the VP, it does not go into the SBAR because it does not start with V.
Algorithm 1. Pseudo-code of identifying the Main Verb from the NLP Parsing Tree

1: function extractMainVerb(char[] s)
2: Input s : scenario sentence to find main verb
3: Output main verb of sentence
4: Tree t = parsing(s)
5: while true do
6: for all Tree subT = subtrees of t do
7: if value of subT begins “V” then
8: t = subT
9: break
10: end if
11: if subT has child tree then
12: continue
13: else
14: return subT
15: end if
16: end for
17: end while

3.2. Similar Action Cluster Generator

This step is intended to make groups of the main verbs identified from the previous step in terms of its semantics. Figure 3 shows the process of this step. First, the verbs are first separated into two groups according to the type of an agent (e.g., a use or a system) in a scenario. The agents are separated authored in writing scenarios in separated column of the scenarios. Each group is also separated into two groups depending on the number of occurrence in the use case specification. If the verb does not reach to the minimal threshold\textsuperscript{1} it is classified into Low Frequency Group, and others are classified into High Frequency Group.

Then, the verbs only in the high frequency group are clustered based on the semantic distance from Wordnet dictionary [3]. After clustering the verbs in the high frequency group, the verbs in the low frequency group are mapped into those in the high frequency group based on the similarity (see the last activity in Figure 3).

\textsuperscript{1} The threshold is decided in the case study section.
In order to calculate the similarity between all verbs in the high/low frequency groups, the following equation is used. The equation considers the semantic distance similarity between verbs, and relative frequency of a specific sense containing with a synonym \( v_2 \) over all summation of frequencies of senses of a word \( v_1 \). After proceeding this step, a verb clusters in the high frequency group linked with the verbs in the low frequency group is identified, also it is classified into a user as well as a system groups.

\[
sim(v_1, v_2) = \max_{1 \leq i \leq n_1, 1 \leq j \leq n_2} \left( \frac{\text{freq}(s_{1i}) + \text{freq}(s_{2j})}{\sum_{1}^{n_1} \text{freq}(s_{1i}) + \sum_{1}^{n_2} \text{freq}(s_{2j})} \right) \times \text{similarity}(s_{1i}, s_{2j})
\]

where \( n_x \) is number of sense of \( v_x \), \( \text{freq}(s_{xy}) \) is frequency of \( y_{th} \) sense of \( v_x \)
and similarity() indicates semantic similarity from Resnik[11]

**3.3. Extracting use Case Flow Pattern Graphs from the Clustered Verb Sets**

In this step, the use case flow pattern graph is initially produced by the transformation rules, and it is simplified based on the suggested equations. The use case flow graph consists of nodes obtained from agents and its corresponding verbs, and edges from the sequence of the scenarios. To transform the agent and verb groups according to its frequency from the previous step, first all clusters are transformed into a node and the sequence of scenarios in use case specifications become edges of the graph.

![Figure 3. Process of Generating Verb Cluster](image)

After depicting the graph, the minor relationships between nodes should be filtered out. We apply two equations to compute the strength of the relationships. *Relative Inter-connectivity (RI)* is an equation to compute the absolute interconnectivity of two clusters normalized by the internal connectivity of the clusters [10]. Figure 4 shows the concept of RI. The weight of Figure 4(ex: a-out, b-in) implies the occurrence number between two verbs sequentially. For example \( ab \) is total number of verb B next of the verb A in the entire scenarios in use case specifications. RI of the node A and B can be obtained from the following equation.

\[
RI = \frac{EC(A, B)}{\frac{1}{2}(EC(A) + EC(B))}
\]

where \( EC(A, B) = ab \) : weight of A to B,
\( EC(A) = a\text{-out}_1 + a\text{-out}_2 + a\text{-out}_3 \) : total weight of A out edge
and \( EC(B) = b\text{-in}_1 + b\text{-in}_2 \) : total weight of A in edge
The second equation is the frequency weight as a complement equation of RI. In order to compute the frequency weight, we suggest the Maximum based frequency (MF) as below. It is relative proportion between a specific edge and the maximum edge among all edges.

$$MF_{ei} = \frac{\text{occur.}(e_i)}{\max_{0 \leq i < n}(\text{occur.}(i))}$$

where \(\text{occur.}(n)\) indicates the occurrence number between two verb in a subsequent scenario.

Figure 4. The Concept of Relative Interconnectivity (RI)

If an edge is not reached to predefined threshold of each two values\(^2\), it removed from the graph. After filtering out the minor edges, each edge has a normalized weight. The MF equation is critical, because Relative Interconnectivity does not consider the absolute occurrence number of the edges. During processing these three step, a single use case pattern graph is created from the give use case specification. After that, we can extract use case flows patterns from the graph.

4. Case Study

In order to show the feasibility of the suggested method, this section demonstrates the steps to identify two use case flow patterns from the real-world industrial projects. Table 1 shows the list of the projects for this case study. We collected eight projects consisting of 83 use case specifications and 615 scenarios. Main verbs of all scenarios are detected at first through the algorithm introduced in the Extracting Main Verb step. Then, the main verbs are classified into the high and low frequency groups with the threshold 0.01. This threshold implies that if the main verb is occurred less than 1 % (7 main verbs of 615 verbs), the main verb is classified into the low frequency group. Others are classified into the high frequency group. We obtained the threshold 0.01 through the sensitivity analysis. In the case of the value is less than 0.01, too many number of verbs are joined in the clustering, which may cause to make very fine grained semantic clustering. Otherwise, the threshold value over 0.01 makes too small number of verbs to be joined in clustering. Thus we realized that the 0.01 is appropriate as a threshold.

After building two groups, the semantic clustering for the verbs in the high frequency group is carried out in association with the semantic distance between two verbs. After then, the verbs in the low frequency group is mapped with the equation \(\text{sim}(v_p, v_z)\). In this case study, 32 main verb clusters is created, including 12

\(^2\)see the case study for deciding the threshold
verbs for the user agent and 20 verbs for the system agent respectively. For example, the agent and verb set: “user select”, “user choose” and “system provide”, “system recommend” are examples of a semantic clustering and mapping two sets of verbs from two groups with the equation $\text{sim}(v_1, v_2)$.

**Table 1. Project List for Case Study**

<table>
<thead>
<tr>
<th>Project</th>
<th># of scenario</th>
<th># of action sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>4</td>
<td>38</td>
</tr>
<tr>
<td>Online Shopping System</td>
<td>4</td>
<td>32</td>
</tr>
<tr>
<td>Payroll System</td>
<td>13</td>
<td>132</td>
</tr>
<tr>
<td>Cafeteria Ordering System</td>
<td>26</td>
<td>176</td>
</tr>
<tr>
<td>University Information System</td>
<td>18</td>
<td>109</td>
</tr>
<tr>
<td>Traffic Management System</td>
<td>6</td>
<td>46</td>
</tr>
<tr>
<td>Stock Ordering System</td>
<td>7</td>
<td>52</td>
</tr>
<tr>
<td>Order Processing System</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>Total</td>
<td>83</td>
<td>615</td>
</tr>
</tbody>
</table>

**Table 2. Examples of Generated Verb Cluster**

<table>
<thead>
<tr>
<th>Agent</th>
<th>Cluster set</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>pay, submit, sent</td>
</tr>
<tr>
<td>user</td>
<td>choose, select</td>
</tr>
<tr>
<td>user</td>
<td>request, send, set</td>
</tr>
<tr>
<td>system</td>
<td>display, open</td>
</tr>
<tr>
<td>system</td>
<td>alert, notify, report</td>
</tr>
<tr>
<td>system</td>
<td>provide, recommend</td>
</tr>
</tbody>
</table>

We obtained the use case flow pattern graph as shown in Figure 5. Initially, the graph was very complex due to too many edges between nodes from the agent and verb set. Through the RI and MF, the edges can be removed to reduce the complexity of the group. In this case study, we decided the threshold of both $RI$ and $MF$ as 0.1 through the sensitivity analysis, which means that if one of them are smaller than 0.1, the edge is removed from the graph. Then, the final weight of edges is calculated with weighted multiplication of RI and MF. We decided weights for RI and MF are 0.7 and 0.3 respectively.

In Figure 5, the node indicates a set of the agent and verb (e.g., “u:choose” implies the agent is “user” and the verb is “choose”). The weight of edges from the node 1 to the node 2 denotes that the node 2 is likely to exist after the node 1, that is, the node 2 can be the subsequent scenario after the node 1 scenario. For example in Figure 5, the “system display” node (0.371) and “system request” node (0.135) exist around the “user choose” node. As the “system display” node (0.371) has a higher edge weight, the “system display” scenario is likely to happen after the “user choose” scenario.
Then we can extract scenario flow patterns from the generated graph. Although there are lots of possible routes, some meaningful route can be a use case flow pattern among them. Figure 6(a) illustrates the examples of extracting patterns from the use case flow graph. In this case study, we manually identified the use case flow pattern from the use case flow graph. However, once an appropriate start node is selected, then all possible routes can be automatically identified though the graph traversal algorithm. Figure 6(b) presents two use case flow patterns extracted from the use case flow graph.

5. Conclusion

In this paper, we have suggested an approach to automatically generating requirement scenario flow patterns from the use case specifications from real industrial projects. The approach consists of extracting main verbs by using the NLP parser, verb clustering through the semantic distance, and graph shrinking algorithm to manipulate the use case flow graph. Finally, we identified use case flow patterns from the use case flow graph. To demonstrate feasibility of our approach, we adopt the 8 projects including 83 use case specifications and extracted possible use case flow patterns from the scenarios. We currently are planning to research for fully automated technique for identifying use case flow patterns and its tool support.
Acknowledgements

This research was supported by the Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2014M3C4A7030503).

References


Authors

Deokyoong Ko, is Ph.D. candidate of the Department of Computer Science and Engineering at Sogang University. He received his B.S. degree in computer engineering from Myongji University in 2006 and the M.S. degree in Graduate School of Information and Technology at Sogang University in 2011. He worked in Seahan Information Systems as a software developer during 2006-2011.

Sooyoung Park, is a Professor of the Department of Computer Science and Engineering at Sogang University. He received his B.S degree in computer science and engineering from Sogang University in 1986 and the M.S. Degree from Florida States University. And he received his Ph.D. Degree in Information Technology from George Mason University. His research focuses on financial technology (FinTech), internet of things (IoT) and software dynamic analysis.

Suntae Kim, is an Assistant Professor of the Department of Software Engineering at Chonbuk National University. He received his B.S. degree in computer science and engineering from Chung-Ang University in 2003, and the M.S. Degree and Ph.D. Degree in computer science and engineering from Sogang University in 2007 and 2010. He worked in Software Craft Co. Ltd., as a senior consultant and engineer for financial enterprise systems during 2002–2004. Also, he developed Android based Smart TV middleware from 2009 to 2010. His research focuses on software architecture, design patterns, requirements engineering, and source code mining.
Mansoo Hwang, is a Professor of School of IT Convergence Engineering in Shinhan University. He received B.S. degree and M.S. degree in computer science and engineering from Chung-Ang University in 1984 and 1986. And he received Ph.D. degree in computer engineering in Soongsil University in 2001. His research focuses on Software Engineering, Requirement Engineering and Software Quality Management.