Model-based Object-oriented Requirement Engineering and its Support to Software Documents Integration

William C. Chu, Chih-Hung Chang, Chih-Wei Lu
1Dept. of Computer Science and Information Engineering, Tunghai University, Taiwan No.181, Sec. 3, Taichung Port Rd., Situn District, Taichung City 407, Taiwan (R.O.C.)
2Dept. of Information Management, Hsiuping Institute of Technology, Taiwan No.11, Gongye Rd., Dali City, Taichung County 412, Taiwan (R.O.C.)
1cchu@thu.edu.tw, 2{chchang, cwlu}@mail.hit.edu.tw

Abstract - Maintaining uniformity of software requirement documents with artifacts from other phases of software life cycle is a very important, however, still a difficult and time-consuming task. Most requirement documents were written in ambiguous natural language which is less formal and imprecise. Without proper modeling, the requirement knowledge is laboriously captured and thus the following formal integration with other artifacts is infeasible. In this paper, we propose a Model-based Object-oriented approach for Requirement Engineering to support and improve the maintenance and consistency of software requirement documents, as well as the consistency with other artifacts through software life cycle. By applying modeling and Object-Oriented technologies to requirement, the domain knowledge can be captured in a well-defined model. The completeness, consistency, traceability and reusability of requirement and its integration with the artifacts of other phases thus can be improved.

Keywords: Software Maintenance, Object-oriented, Requirement Document, UML

1 Introduction

By the inundation of computers and networks from the end of the 20th century, informatization has generally acknowledged as the main trend toward modern business operations and enterprise management. Yet this population brings more challenges to the software systems development. Software now needs to be more flexible for enterprise mutability, and quicker but with more accurate analyses, as well as the designs, for the least time-to-market commercial essentiality. In practice, most of software requirement documents are represented in natural language, which are usually ambiguous, imprecise, incomplete, and informal, in spite of containing some concrete descriptions such as use-case diagrams. Without proper modeling, the knowledge of the requirement is hard to be captured formally and its integration with other artifacts is infeasible. Moreover, the modeling process of software requirement needs to face a challenge of representing these informal and incomplete requirement documents into a well-defined model, so the modeled requirements can be integrated with other much formal software artifacts, such as the diagrams in UML from the upcoming phases analysis, design, implementation, and testing. By observation to the companies who have practiced CMMI and passed level 2 and 3, one of the biggest costs is to maintain the traceability and the consistency of related documents.

On the other hand, for the last decade the growing influence of object-oriented programming (OOP) has led to the rise of new paradigms for software development, generally known as object-oriented analysis and design (OOA/OOD). The most promise of OOA/D and OOP is that the software development process can be efficient and simplified through having the common building blocks [1] (e.g., classes, objects, methods, and inherences), used in phases of software development, from software analysis to implementation. These paradigms adopt concept from object-oriented programming and blend it with semantic data modeling and knowledge representation into modeling frameworks such as UML, which makes the software development process more concrete and manageable and therefore OOA/D have been accepted as the general formal methodologies used in current software development.

However, there are still insufficient of corresponding consistency that OO complies for the requirement process. Defining and specifying the requirements of a software system from the very early stage is essential of the software development lifecycle since meeting stakeholder’s needs is always the principal part for any project. Requirement phase activities, which indeed need the professional and formal treatment the most, usually were treated the less in an informal way. Practically, in requirement elicitation and elaboration stages, natural language has still remained as the general choice for the software developers to specify the software requirements for higher flexibility, since most of the stakeholders
involved were either poor-trained or lack of domain knowledge and experiences[2].

In this paper, we propose a Model-based Object-oriented approach of Requirement Engineering (MORE) to support and improve the maintenance and consistency of software requirement documents and its consistency with other artifacts of software life cycle.

Applying modeling and OO technologies to requirement, the domain knowledge can be captured in a well-defined model, so the completeness, consistency, traceability and reusability of requirement and its integration with the artifacts of other phases can be cost effectively improved.

This paper is organized as follows. In section 2, we list the related works and the summaries of our earlier research. Next, the XML-based unified model (XUM) and MORE integration is shown in section 3. In the section 4, we conduct an industrial case study to demonstrate the mechanism. Conclusion and future works are given in section 5.

2 Related work

XML [3] is a standard language supported by W3C (World Wide Web Consortium). It offers application neutrality (vendor independence), user extensibility, ability to represent arbitrary and complex information, validation for data structure scheme, and human readability. In our earlier research [4], we apply XML meta-model to describe and integrate related software paradigms of a system into an XML-based unified model (XUM). XUM is a mechanism to integrate and unify sets of models/paradigms (e.g., UML or design patterns) in well accepted software standards (e.g., OOA, OOD, or OOP). The mechanism utilizes a XML-based unified meta-model (XUMM) to construct the corresponding XUMs which present the schema of adopted models in software development process. The mechanism also creates explicitly the relations between these adopted models. The connective models assist the maintenance of consistency by reflecting the changes to all affected models. The mechanism handles ripple effects systematically to reduce the difficulty of software maintenance.

Many approaches have been proposed for requirement engineering. Object-oriented requirement engineering, which takes the good features of object-oriented paradigms, has been proposed to assist the activities of requirement phase, such as requirement elicitation, analysis, negotiation, documentation, verification and validation, and management. Many researches focus on use cases and scenarios to systematically model the requirement artifacts. In our early work of MORE [5], it demonstrated the feasibility of the OO modeling for informal text-based documents.

In past decades, design patterns [6] propose solution skeletons for common design problems. The solution skeleton is described the design can be reuse for other projects. Based on the concept of design pattern, more researches are aware of patterns applicable to other parts of the software development process. Gross and Yu [7] discussed the relationship between non-functional requirements and design patterns. Sutcliffe et al. [8] described how scenarios of use cases can be investigated to identify generic requirements for different application classes. Konrad et al. [9][10] applied patterns to requirement for embedded system.

For official requirement specification, IEEE presented several sample about SRS outlines [11]. However, they still used natural language to describe the requirement of system.

3 The XML-based unified model and MORE

In MORE, information of requirement documents are generally collected and written in naturally language, and then requirement engineer can objectize key concepts into corresponding objects and class, which are called ROMs (Requirement Object Models). The objects are kept in reusable repository, which can be reused for further requirement definitions. ROMs and the objectization of requirement documents lead to a formal system representation which can assist precise specification of the requirement semantics and then prevent incompleteness and inconsistency.

By the construction of ROMs and the related requirement domain knowledge, a set of model-based requirement templates are accumulated. A requirement template consists of actors, scenarios, constraints, and effects. The templates encapsulate the experiences and knowledge of domain experts, which can help and guide requirement engineer to elicit requirements more correctly and effectively.

![Figure 1: The concept process of MORE](image)

Figure 1 shows the concept process of our approach. With the support of ROM, the process of the design phase is assisted with reusable requirement objects, so the related design paradigms such as the class diagrams, activity diagrams, …etc, will be associated and consist with each other in the XML-based Unified Model (XUM).

To making up the space that XUM approach left in the requirement phase, for the following software development phases, software paradigms are represented and integrated as models of XUM. Figure 2 shows the relationship between the software paradigms and XUM.
During software maintenance, modification to any sub-model should be detected and should reflect the effects on the related sub-models; the semantics in each sub-model can then be updated appropriately according to the modification. This assists the consistency checking of modeling information of views. In addition, impact analysis can be applied to the entire software system, including the impact on related source codes, the impact on related design information, and the impact on related requirement information.

4 Case study

In order to demonstrate our approach from document-centered requirement development to MORE development, we conduct an industrial case study. The subject of the case is about developing the software requirements of a Book/Magazine distribution system.

Table 1: A set of questions support collecting original needs from customers

<table>
<thead>
<tr>
<th>Category/Questions</th>
<th>Example (B Corp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goals</td>
<td>What are the major goals of the project? B Corp. has an ERP purchase system but its document approval procedure is executed manually. The company wants to exploit the approval application of book/magazine to increase the performance and reduce error rate.</td>
</tr>
<tr>
<td>Functional Needs</td>
<td>What are the major functionalities of the targeted application? 1. The book distribution system can support automatic distribute procedure. 2. The book distribute system can support adjusting flexibly the authorized amount of book/magazine of each position.</td>
</tr>
<tr>
<td>Non-functional Needs</td>
<td>What are the non-functional expectations of the targeted application? The user must be authorized before he/she distribute.</td>
</tr>
<tr>
<td>Operation concepts</td>
<td>What are the operation procedures? 3. The staffs of B Corp. interact with the book distribute system to distribute the new book or magazine to each pathway by different levels of bookstore.</td>
</tr>
<tr>
<td>Constraints</td>
<td>What are the constraints of the targeted application? 1. Before book distribute, the related data must be input into database. 2. If the difference between the amount of distribute and the amount history is the large, system must email to notify related user.</td>
</tr>
</tbody>
</table>

In the development of customer requirement, developers elicit customer needs with the questionnaire, as shown in Table 1. The replies got from the questionnaire imply implicit software needs and explicit hardware demands. The design of software needs elicitation is emphasized on the understanding of project goals, functional or non-functional needs,
system operation concepts, and specific constraints. The design implies that the customer needs could include the concepts, which is informal at the early stage.

The case study involved with developing the software requirements of a workflow application adopted the requirement development process shown in Figure 5.

![Figure 5: requirement development process](image)

Table 2: The detailed identification of book distribution requirement

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Business process ID</th>
<th>Date</th>
<th>Business process ID</th>
<th>Designer</th>
<th>Actor</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Book distribution</td>
<td>P3.2</td>
<td>5.30.06</td>
<td>Bookman</td>
<td>Eric</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Module Description**

1. The distributor input the book_id, to check the data is existed?
2. Select using fixed or no,
3. Select the mode of distribute

**Pre-condition**

User had finished the basic materials of of new book in I1

**Post-condition**

1. successful: goto 1.2Transformation of pathway
2. unsuccessful: show the message “The data of new book are not exist”, and disable the button of 1.2Transformation of pathway and 1.3Select the pathway

**Main process**

1. User input the ID of new book which want to distribute
2. User input the amount of book
3. User select the mode of distribute
4. User click the [OK]
5. Check the data of new book are already finial in I1
6. If it has been existed, goto 1.2Transformation of pathway, 2.0 Distribute the level of new book, or 4.0Distribute of magazine
7. Else show the message “The data of new book are not exist”

**Alternate process**

N/A

**Screen:**

The customer requirement is developed from the needs of customers. The customer needs generated from the questionnaire are used to roughly identify the components in the application, including which business processes should be automated (workflow application), which documents are managed/executed on the processes (relevant data), and which legacy systems (ERP purchase system) will be integrated with. Table 2 shows the results elicited from the original needs.

Table 2 shows the results elicited from the original needs.

Based on the development procedure, the customer requirement of the project generated in this stage might contain the identifications of all coarse components in the workflow application (e.g., the business processes, extra applications, and relevant data). So, the components of workflow application might be the fundamental entities of the customer requirement. The design of customer requirement model is composed of the same entity model as customer needs model. The customer needs model and customer requirements model associating with each other are linked with association relation. The detail design is shown in Figure 4.

Figure 6 shows the corresponding Use Case diagram derived from system requirement specification. And Figure 7 is the corresponding activity diagram of system requirement specification.

Figure 6: The corresponding Use Case diagram of system requirement specification

![Figure 6](image)

Figure 7: The corresponding activity diagram

After system analysis design the related document of system, according MORE2MXUM transformer, these requirement documents and design documents can be represented as an XML-based file as summarize in figure 8. Beside the organized information, the integration link which the traceability are added into XUM.

![Figure 7](image)
According the integration links, any modify of requirement, developer can extract related paradigms easily. For example, if we change the content of customers_need, according unifiedLine we know that the requirement: 1.0 Distribute pre-operational maybe need to modify.

```xml
<xml version="1.0" encoding="UTF-8"/>
<XUMM>
  <customers_need id="CN1.0" name="Example (B Corp)"
  <unifiedLink link="Abstraction" xlink:type="simple" xlink:href="R1.0"/>
  <goal>...<goal>
  <function_need>...<function_need>
  <nonfunction_need>...<nonfunction_need>
</customers_need>

  <requirement id="R1.0" name="1.0 Distribute pre-operational"
    <unifiedLink link="Integration" xlink:type="simple" xlink:href="U1.0"/>
  >
    <activityDiagram id="CA001" name="class diagram of book distributer">....
    <process id="PA001" performer="Distributer">Input Book_id</process>
    <unifiedLink link="Abstraction" xlink:type="simple" xlink:href="U1.1"/>
    <goals>The pre-operational before distribute the books or magazines to retailers</goals>
    <description>1. Distribute the book_id, to check the data is existed</description>
  ...
</requirement>

  <requirement id="R1.1" name="1.1 Book distribution"
    <unifiedLink link="Integration" xlink:type="simple" xlink:href="U1.0"/>
  >
    <actor id="UA001">
      <description>1. The distribut er input the book_id, to check the data is existed</description>
  ...
</requirement>

  <requirement id="R1.2" name="2.0 Book distribution"
    <unifiedLink link="Integration" xlink:type="simple" xlink:href="U1.0"/>
  >
    <actor id="UA001">
      <description>1. Distribute the book_id, to check the data is existed</description>
  ...
</requirement>

  <requirement id="R1.3" name="3.0 Book distribution"
    <unifiedLink link="Integration" xlink:type="simple" xlink:href="U1.0"/>
  >
    <actor id="UA001">
      <description>1. Distribute the book_id, to check the data is existed</description>
  ...
</requirement>

  <requirement id="R1.4" name="4.0 Book distribution"
    <unifiedLink link="Integration" xlink:type="simple" xlink:href="U1.0"/>
  >
    <actor id="UA001">
      <description>1. Distribute the book_id, to check the data is existed</description>
  ...
</requirement>

  <requirement id="R1.5" name="5.0 Book distribution"
    <unifiedLink link="Integration" xlink:type="simple" xlink:href="U1.0"/>
  >
    <actor id="UA001">
      <description>1. Distribute the book_id, to check the data is existed</description>
  ...
</requirement>

</XUMM>
```

Figure 8: The correspond XUM

In order to prove our methodology, we implement a prototype called MOR Editor. MOR Editor provides classification to organize the defined objects in class hierarchy. Requirement engineer can apply inheritance techniques to increase the reusability. MOR Editor also provides association function to allow requirement engineer to link these informal and formal objects and classes together. Therefore, related document segments (informal ones) and formally defined objects can be associated with.

Through MOR Editor, the documents in requirement are modeled and candidate reusable documents are made as template. Our approach also adapts the concept of pattern, successful solutions to recurring problems, to the requirement modeling. Requirement engineer can define reusable requirements into templates. For example, a requirement specification about any data processing needs concern about its functionality the creation, deletion, modification, and query. Without considering them all, in practice, we have found many requirements are incomplete, which may cause the cost estimation wrongly. The template contains the experience and knowledge of domain expert, which can help and guide requirement engineer to solicit requirements more correctly and effectively. Java is adopted as the implementation language for the tool prototype, so the MOR Editor can be run across platforms such as Windows, Linux, or UNIX. There are five major toolsets collaborating one another in MOR Editor.

A) The toolbar of MOR Editor, to manipulate functions of the editor.
B) The requirement/object structure browser, to list all element objectization of the requirement in the MORE structure.
C) The requirement engineer viewer, to display the contents needed for requirement engineer assistances.
D) The requirement workspace, to edit requirement documentation. In here, requirement engineer can define the requirement details, requirement states, and requirement relationships.

Figure 9 shows the layout of MOR Editor toolsets in Windows.

![Figure 9: Layout of the tool MOR Editor](image)

5 Conclusion

Software requirement is the cardinal importance for developing and maintaining an information system because it is what a system is needed for. Requirement documents are usually written in natural languages. Most of them are informal and that leads to the difficulty for capturing complete/consistent domain knowledge. As a result, the following processes of software development and maintenance are costly and error prone.

Many disparate software analysis/design methods and tools promise fast, efficient software design/evolution, yet they are generally incompatible and therefore suffer from a lack of communication and integration, especially to the paradigms of requirement
process. Problems of implicit inconsistency, which are caused by making changes to requirement specifications, components of the models and designs, significantly increase costs and errors for maintenance.

In this paper, first we proposed a model-based requirement development framework to improve the quality of requirement analysis. Second we proposed a mechanism to integrate system paradigms from requirement to implementation. If system developers/maintainers modify any software paradigms, he/she can easily spot the related paradigm updates. Third we implement a requirement editor prototype MOR Editor.

For the future work of this research, we will improve MOR Editor more complete to support XUM which serves as a mechanism for the integration and maintenance through the whole process of software life cycle.

6 References


[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.


