2010 International Conference on Multimedia Information Networking and Security

An approach of Inconsistency Verification of Use Case in XML and the Model of Verification Tool

Ning Jin
College of Computer Science & Information
Guizhou University
Guiyang, China
Email: gzgyjinning@126.com

Jing Yang
College of Computer Science & Information
Guizhou University
Guiyang, China
Email: yangjing6646@yahoo.com.hk

***Abstract*—In order to solve the problems of verifying the inconsistency of the requirement specification in the natural language, and automatically transforming the requirement specification in natural language description to formal models, this paper proposes a modeling method of use case specification in XML and a method of verifying inconsistency in the specification. Furthermore, we design a corresponding model of auxiliary tool in our framework. The requirement analysts can use the tool to transform requirement document in the natural language into the formal specification automatically. The formal specification can be used for further automatic requirement verification.**

***Keywords*-use case; inconsistency; verification; formal methods; XML;**

I. Introduction

In recent years, people have paid much attention to the correctness, reliability, secrecy and security [1]–[4], etc. The high confidence of software system has become an important research field in the theory and technology of software. Through practicing in requirements engineering, we found that the quality of requirements engineering fundamentally depends on whether we formulate completely trusted requirements engineering process before starting the project. Only under the unifying guidance of the standard requirements process, and selecting the appropriate formal methods and supporting tools, can we guarantee the quality of the requirements engineering implementation and its products(i.e. the software requirements specification(SRS)). The core idea of RUP(Rational Unified Process) [5] is use case driven, architecture centric and iterative and incremental development process. In the RUP, use cases are used to capture the functional requirements and to define the contents of the iterations. Each iteration takes a set of use cases or scenarios from requirements all the way through implementation, test and deployment. This article uses XML to format the use case description, and proposes an approach of inconsistency verification of use case. The corresponding model of verification tool(UCBuilder) is also designed.

∗ Supported by NSF Project 90718009, Scientific research fund for talents recruiting of Guizhou University; QianKeHeJZi[2010]No.2094.

II. Related Works

There are many works about requirement engineering. The article [6] proposes a formal transformation model of the software requirements, which is used to convert the software requirement analysis to the formal specification directly and automatically. The articles [7]–[9] give the formal semantics of several kinds of UML diagrams, and verify the consistency between these diagrams. The article [10] proposes a XML-based method of use case specification, but it is lack of verification in semantics.

The majority of above research results formalize the existing model through formal methods, give its semantics, and then verify the inconsistency of the model manually. So, requirement analyst must master some formal methods and ensure the correctness of the transformation models from natural language to formal language. Now there is no research achievement of automatically transforming the natural language description to the UML model by the supporting tools in a simply manner. This paper describes use cases in XML, and proposes a method of the inconsistency verification in use cases specification, finally designs a model of supporting tool UCBuilder for verifying automatically.

III. Building Use Case XML Document

A. Use Case Body

Use case description plays a significant role in requirements document. Cockburn's work about use case is very meaningful. He proposed that a use case is a description of the interactions and responsibilities of a system with external agents, or actors. The use case is associated with the goal of one particular actor, who is called *primary actor* for that use case. Each possible sequence of interactions is called a *scenario* [11], each scenario is composed of several interactive actions called *steps*. According to Cockburn, a use case body includes “Name”, “Level”, “Primary actor”,“preconditions”,“Postconditions”,“Main Success Scenario”,“Extensions”,and “Priority” etc.(see table I). Consequently, we can create a meta-model of the use case, as

978-0-7695-4258-4/10 $26.00 © 2010 IEEE
DOI 10.1109/MINES.2010.160

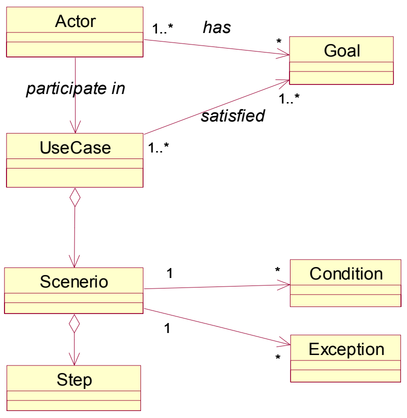


Figure 1. Use Case Meta-Model

shown in Fig.1. In Fig.1, the *Actor* class has one or more goals that are realized by use case.The uss case achieves goals by the execution of scenarios.

Table I
PROPERTIES IN USE CASE BODY

Use case property	Description	Related XML tag
ID	unique identifier	<ID>
Name	name of use case	<Name>
Description	simple description of use case	<Description>
Level	summary, core, supporting, or internal use case?	<Level>
Primary actor	role names of people or external systems initiating this use case	<Actor>
Preconditions	what must be true before a use case can begin	<Precondition>
Postconditions	what must be true about the system after a use case completes	<Postcondition>
Main success scenario	everything goes right	<MainStep>
Extensions	different ways that occur during the execution of a step	<Extension>
Priority	how important is this use case?	<Priority>
Goal	goals the use case will achieve	<Goal>

B. Definition of XML Schema of Use Case Specification

An obvious feature of the Extensible Markup Language(XML) [12] document is that we can use XML Schema Definition(XSD) to test and verify the syntax [12]. XML Schema defines the various elements, properties and other syntaxes appeared in a class of XML documents. Here we use the W3C XML Schema Definitions(XSD) [12] to define the elements, attributes, sub-elements, the number and sequence of sub-elements, the data types of elements and attributes in use case body. so as to make the use case specification more standard and unified. Now we can define the XSD model of use case. Fig.2 shows the segments of xsd document of use case. The semantic of each element in Fig.2 is shown in table I.

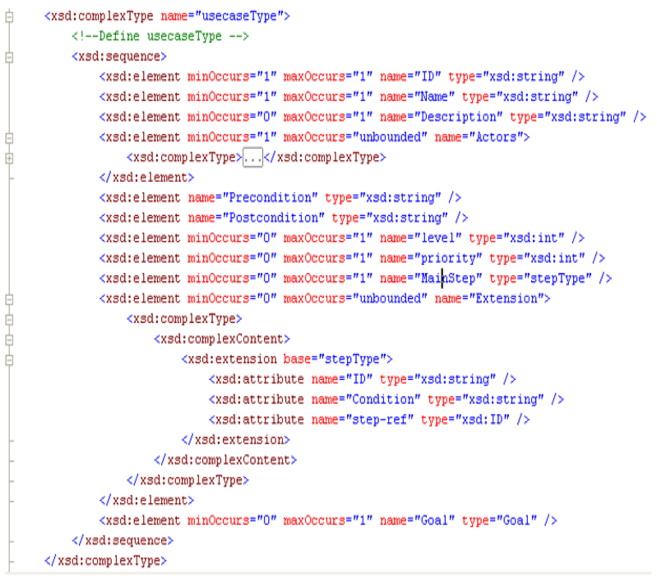

```
<xsd:complexType name="usecaseType">
    <!--Define usecaseType -->
    <xsd:sequence>
        <xsd:element minOccurs="1" maxOccurs="1" name="ID" type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="1" name="Name" type="xsd:string" />
        <xsd:element minOccurs="0" maxOccurs="1" name="Description" type="xsd:string" />
        <xsd:element minOccurs="1" maxOccurs="unbounded" name="Actors">
            <xsd:complexType>...</xsd:complexType>
        </xsd:element>
        <xsd:element name="Precondition" type="xsd:string" />
        <xsd:element name="Postcondition" type="xsd:string" />
        <xsd:element minOccurs="0" maxOccurs="1" name="level" type="xsd:int" />
        <xsd:element minOccurs="0" maxOccurs="1" name="priority" type="xsd:int" />
        <xsd:element minOccurs="0" maxOccurs="1" name="MainStep" type="stepType" />
        <xsd:element minOccurs="0" maxOccurs="unbounded" name="Extension">
            <xsd:complexType>
                <xsd:complexContent>
                    <xsd:extension base="stepType">
                        <xsd:attribute name="ID" type="xsd:string" />
                        <xsd:attribute name="Condition" type="xsd:string" />
                        <xsd:attribute name="step-ref" type="xsd:ID" />
                    </xsd:extension>
                </xsd:complexContent>
            </xsd:complexType>
        </xsd:element>
        <xsd:element minOccurs="0" maxOccurs="1" name="Goal" type="Goal" />
    </xsd:sequence>
</xsd:complexType>
```

Figure 2. XSD Document of Use Case

C. Transforming the Use Case description into XML Specification

Here is a sample of use case “Edit User Account”(see table II). The related xml docunment can be written like table III. According to the XSD document shown Fig.2, we can easily transform the original requirements to the formal specification in XML.

Table II
THE SEGMENTS OF USE CASE SAMPLE

Use case: Edit User Account
Primary actor:administrator
Precondition:the administrator login the system successfully.
Postcondition:saving the information of use account successfully.
Level:core
Priority:2
Main Success Scenario
1.The administrator selects an option to list users accounts on the system.
2.The system displays all users accounts information.
3.The administrator selects a user account he want to modify and selects the “Edit” option.
4.The system displays the user account detail.
5.The administrator modify the user account information.
6.The administrator submits the user account information he modified.
7.The system valid the information.
8.The system save the modified user account information successfully.
Extension
4a. Unable to connect to the network.
4a1.The system displays a errors about connection.
. . .
7a.Not pass the validation.
7a1.The system displays a errors about invalid input.
. . .

Table III
THE SEGMENTS OF USE CASE XML DOCUMENT SAMPLE

```
<?xml version="1.0" encoding="utf-8" ?>
<usecase ID="UC1" Name="Edit User Account">
  <Actor IsPrimary=true >administrator</Actor>
  <Precondition>the administrator login the system successfully.
  </Precondition>
  ...
  <MainStep>
    <Step ID="uc1ms1" >The administrator selects an option to list
users accounts on the system.</Step>
    ...
    <Step ID="uc1ms4" >The system displays the user account
detail.</Step>
    ...
  </MainStep>
  <Extension ID="uc1ms4ext1" Step-ref="uc1ms4" Condi-
tion="Unable to connect to the network."" >
    <Step ID="uc1ms4" >The system displays a errors about con-
nection.</Step>
    ...
  <Extension ID="uc1ms7ext1" Step-ref="uc1ms7" Condition="Not
pass the validation."" >
    <Step ID="uc1ms4" >The system displays a errors about invalid
input.</Step>
    ...
  </Extension>
  ...
</usecase>
```

IV. THE INCONSISTENCY VERIFICATION OF THE USE CASE SPECIFICATION

Any requirement modeling method involves a set of rules to maintain the consistency. when the requirement description has a predicate which does not satisfy these rules, we think there is inconsistency in the requirement description, and this issue needs further studies [13]. Inconsistency in the requirements description mainly comes from the redundant requirements [14],while the vague requirements [14] may imply inconsistency. Suppose S is the basic requirements specifications, D represents the facts related to the specific application scenario, E is a set of expected goals in the corresponding application scenario of the target system. Then the inconsistency requirements and redundancy requirements can be defined as follows:

- **inconsistency requirements**: check whether inconsistency exists in S corresponding to the application scenario which means whether $\exists\alpha \in \mathbf{E}$ satisfies $\mathbf{S} \cup \mathbf{D} \vdash \alpha \wedge \neg\alpha$
- **redundancy requirements**: if $\mathbf{S} \cup \mathbf{D} \vdash \mathbf{E}$ and $(\mathbf{S} - \mathbf{S_1}) \cup \mathbf{D} \vdash \mathbf{E}$ or $\mathbf{S} \cup \mathbf{D} \vdash \alpha \wedge \neg\alpha$ and $(\mathbf{S} - \mathbf{S_1}) \cup \mathbf{D} \vdash \mathbf{E}$ (for some $\mathbf{S_1}$:$\varnothing \subset \mathbf{S_1} \subset \mathbf{S}$), then $\mathbf{S_1}$ is the possible set of the redundancy requirements corresponding to the application scenario D.

Based on the above theory, we verify the requirement specification in the XML format from the following perspectives:

1) Verifying the structure and validity of the use case document

   We mainly verify the use case description through syntax. If a XML specification satisfies the definition of the XML syntax, this specification is well-defined. A valid XML specification means that it conforms with the definition of XML Schema [12]. Through defining XSD specification of the use case, we can implement the automatic verification easily with tools.

2) The redundancy verification

   In original use case statement,there are same or similar steps among use cases. For large projects, the manual checking may be more complex and time-consuming. In this paper, after the use case having been formatted in XML, the aided tools will be developed to do redundant automatic validation,here the smallest unit is step. By comparing the steps among use cases, a series of steps with the same or similar semantics among use cases can be extracted which can be called by those use cases. Through this way, the redundant steps in the description of use case can be eliminated at the same time. What's more, the inclusion relation between new use case and the original use case can also be determined.

   According to the definition of the demand redundancy in [14], the validation rules of redundancy can be obtained. If UC is the set of use cases for target systems, $Goals$ is the target set of all external entities. For a $\varnothing \subset \mathbf{UC_1} \subset \mathbf{UC}$, if $\mathbf{UC} \vdash \mathbf{Goals}$ and $\mathbf{UC} - \mathbf{UC_1} \vdash \mathbf{Goals}$ are both right. This set of use cases of $\mathbf{UC_1}$ may be redundant. By inputting the context diagram by the aided tool $UCBuilder$, we can easily obtain the target set of all the external entities. Then it will be compared with the target set achieved by all use cases obtained from the XML document by the parser. According to the redundant validation rules, a group of redundancy use cases can be automatically ruled out. According to the results returned by UCBuilder, requirement analyst could further refine the description of the use case.

3) The integrity verification

   Generally speaking, a requirement model contains at least one *context diagram*, *class model*, *glossary*, and *use case model*. *context diagram* defines the boundary of the system and describes the interaction between external entities for a certain aim. *class model* describes the data structure of use case operation. *glossary* lists domain specific terms. *use case model* is collection of use cases and relationship among them. In the later analysis and design stages, all external entities will be defined in the class model and their goals will be finished by designing appropriate use case. Thus, we can conduct the integrity verification from the following two aspects:

   I. Through checking whether there is at least one use case corresponding with the goal of all external entities, we can decide the integrity of the use case model.

II. Through checking whether all participants have got definition in class model, we can decide the integrity of the class model.
Here,we define the following concepts: CD is the context diagram in requirement model. C is the collection of classes. UCM stands for use case model. Then, the requirement model of a system is a triple $RM = < CD, C, UCM >$. Consequently,there is a formal deduction as following: where

CD represents the system boundary, the external entities that interact with the system and the nature of the interactions with each external entity. We define context diagram as a triple $CD = < EE, G, S >$ where EE is the external entities. G is the collection of goals of the external entities. S represents target system which is to be developed.

UCM, as use case model in requirements, it often contains a term-list called $Glossary$, a collection of all actors represented as $Actor$, a collection of all use cases represented as UC, a collection a collection of non-functional requirements related to use cases represented as NFR. Thus,we can defined as UCM a tuple UCM=$< Glossary, UC, Actor, NFR >$

Let $getActors(UC)$ denotes to get the collection of all actors in UC, $getGoals(UC)$ denotes to get the collection of goals in UC, then we can deduct the rules: $getActors(UC) \subseteq \mathbf{C}, G \subseteq getGoals(UC)$.

4) The inconsistency verification

According to the definition of inconsistency in [14], the expected operating results of a system in the application scene is considered as a set of goals of the external entities. In a context diagram, use case is inconsistency if $\mathbf{UC} \vdash g \wedge \neg g$ where $g \in \mathbf{Goals}$. Describing use cases in XML, we can read all $Goals$ for semantic analysis through the programming interface. In fact, conflicting goals indicate the inconsistency of the use case description.

5) The conflict detection of the use case priority setting

In practice, the improper setting of use case priority will affect the implementation of the entire project. If the priority of a use case is lower than the priority of those including it, logical conflicts about the implementation sequence of the use case will occur. So we should detect whether there is a conflict in the priority setting between correlative use cases.For example, if use case UC_1 includes use case UC_2, then UC_1 takes priority over UC_2, represented as $UC_2.Priority \geq UC_1.Priority$.

V. The Design Model of Verification Tool ***UCBuilder***

Using XML to describe use cases, our goal is to develop appropriate tool for the automatic verification. Now a lot of parsers provide programming interfaces. These parsers support the verification of structure and the validity of XML document structure, such as DOM, SAX, JAXP and so on [12]. The model of UCBuilder that we want to develop is shown in Fig.3.

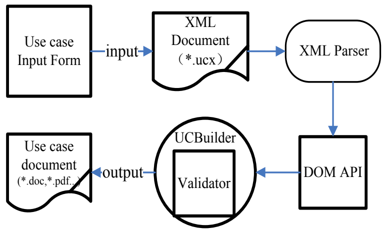
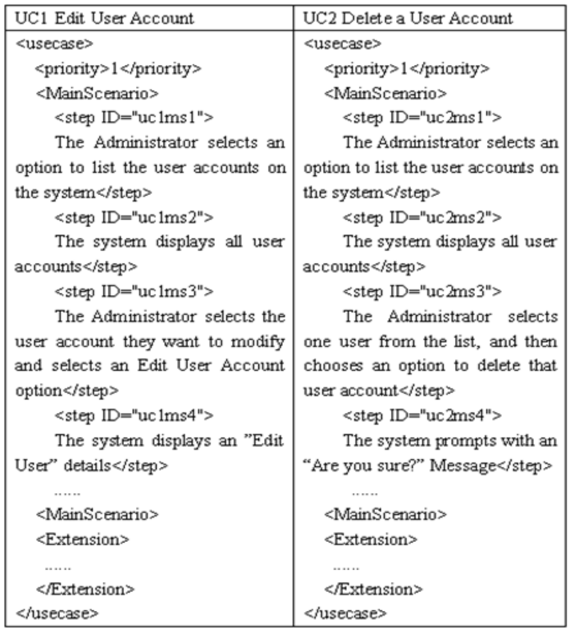


Figure 3. UCBuilder Model



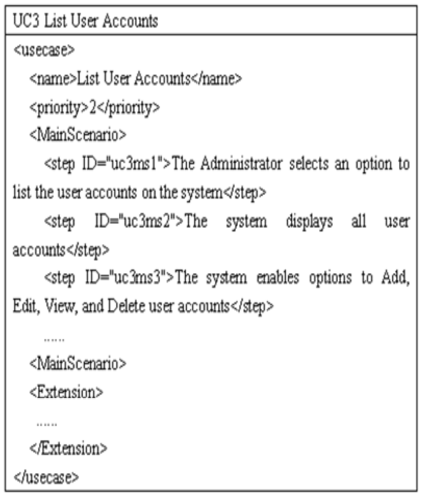
UC1 Edit User Account	UC2 Delete a User Account
<usecase>	<usecase>
<priority>1</priority>	<priority>1</priority>
<MainScenario>	<MainScenario>
<step ID="uc1ms1"> The Administrator selects an option to list the user accounts on the system</step>	<step ID="uc2ms1"> The Administrator selects an option to list the user accounts on the system</step>
<step ID="uc1ms2"> The system displays all user accounts</step>	<step ID="uc2ms2"> The system displays all user accounts</step>
<step ID="uc1ms3"> The Administrator selects the user account they want to modify and selects an Edit User Account option</step>	<step ID="uc2ms3"> The Administrator selects one user from the list, and then chooses an option to delete that user account</step>
<step ID="uc1ms4"> The system displays an "Edit User" details</step>	<step ID="uc2ms4"> The system prompts with an "Are you sure?" Message</step>
……	……
<MainScenario>	<MainScenario>
<Extension>	<Extension>
……	……
</Extension>	</Extension>
</usecase>	</usecase>

Figure 4. The Segments of Tow Use cases

As shown in Fig.3, we define the suffix of the use case XML document is ucx. UCBuilder uses the API provided by XML parser to interact with the ucx file, and reads all the nodes information of the ucx file, then uses the $validator$ component to verify the use cases in various perspectives. For instance, after reading all nodes information, we can compare the steps of each use case with others and find the same steps.

VI. The Case Study

This section illustrates our method through a case study. Fig.4 shows two segments of use cases in Office Automation System. It lists two segments $EditUserAccount$ and $DeleteaUserAccount$. When we use the verification tool $UCBuilder$ to search finding, $UCBuilder$ may find the steps of ID as $uc1ms1$ and $uc1ms2$ in UC_1 are same to



```
UC3 List User Accounts
<usecase>
   <name>List User Accounts</name>
   <priority>2</priority>
   <MainScenario>
      <step ID="uc3ms1">The Administrator selects an option to
list the user accounts on the system</step>
      <step ID="uc3ms2">The system displays all user
accounts</step>
      <step ID="uc3ms3">The system enables options to Add,
Edit, View, and Delete user accounts</step>
      ......
   <MainScenario>
   <Extension>
      ......
   </Extension>
</usecase>
```

Figure 5. The Segments of UC_3

the steps of $uc2ms1$ and $uc2ms2$ in UC_2, so it shows these steps to the requirement analyst, who will redefine these steps as a new use case named as $ListUserAccounts$, as shown in Fig.5. So $UCBuilder$ will automatically replace these repeated steps with call to UC_3.

When the analyst add UC_3 and sets the priority of UC_3 as 2 which is low to UC_1 and UC_2, $UCBuilder$ will indicate that requirement analyst should promote the priority of UC_3 to 1. The new priority of UC_3 will be used for further verification. After the first refinement of UC_1 and UC_2, both of them include UC_3, and UC_3 takes priority over UC_1 and UC_2.

VII. Conclusions

There are many research works [15]–[17] focused on verification and automatic code generation of the software requirement model. However, it is a issue needed to be further studied to smoothly transform the requirement specification in natural language to formal specification for them. In this paper, we explore a transforming model of the use case specification in XML, and propose an approach of inconsistency verification, finally design the corresponding tool to support the automatic verification.

In the future work, we will explore the semantics of XML Schema, then study the inconsistency verification of the requirement model, and implement $UCBuilder$,finally test the functionality of $UCBuilder$ through some case studies.

References

[1] Huowang Chen *et al*, "High confidence software engineering technologies," *Acta Electronica Sinica*, 2003.

[2] JNSTC, "Research challenges in high confidence systems," http://www.hwc.gov/pubs/hcs-Aug97/intro.html, 1997.

[3] High Confidence Systems Working Group, NSTC, "Setting all interagency high confidence systems(HCS) research agenda," Arlington,Virginia, USA, 1998.

[4] High Confidence Software and Systems Coordinating Group, "High confidence software and systems research needs," High Confidence Software and Systems Coordinating Group, http://www.ccicgov/pubs/hcss-research, Tech. Rep.

[5] P. Kruchten, *The Rational Unified Process – An Introduction (2nd Edition)*. Addison-Wesly, 2000.

[6] Lizhen Hou *et al*, "Digestion-based software formal transformation model," *Computer Engineering*, no. 5, 2007.

[7] Yaoxin Shi (School of Computer Science and Technology; Nanjing University; Nanjing210093) *et al*, "MDA-based transformation between UML models–from sequence diagrams to statecharts," *Computer Engineering and Applications*, no. 13, pp. 40–45, 2004.

[8] Xueyang Zhu *et al*, "A temporal logic semantics for UML activity diagrams," *Journal of Computer Research and Development*, no. 9, 2005.

[9] Xiaomeng Zhang *et al*, "Model consistency checking in mda development," *Journal of Chongqing Institute of Technology*, no. 11, 2007.

[10] Jiancheng Li *et al*, "Description of usecase specification based on xml," *Journal of Xi'an Polytechnic University.*, vol. 23, no. 01, 2009.

[11] Alistair Cockburn, *Writing Effective Use Cases*. Addison-Wesley, 2001.

[12] W3C, *XML Schema Part 1: Structures Second Edition W3C Recommendation*, 10 2004, http://www.w3.org/TR/xmlschema-1.

[13] Zhi Jin, "Reliability of software requirements engineering," 2008, http://sei.pku.edu.cn.

[14] Kedian Mu, Zhi Jin, Ruqian Lu, "Managing the non-canonical software requirements," *ACTA Electronica Sinica*, no. 16, pp. 1221–1231, 2005.

[15] Shuhang Guo *et al*, "Research of software dependability requirement specification method based on goal," *Computer Engineering*, no. 33, pp. 37–41, 2007.

[16] Xuefeng Zhu, Zhi Jin, "Managing the inconsistency of software requirements," *Journal of software*, no. 16, pp. 1221–1231, 2005.

[17] Jing Yang *et al*, "A predicative semantic model for integrating uml models," in *LNCS 3407*, Z.Liu and K.Araki, Eds. Verlag Berlin Heidelberg: Springer, 2005.