Customizable Rule-Based Verification of Requirements Ontology

Dang Viet Dzung
Graduate School of Science and Engineering
Ritsumeikan University
Kusatsu, Shiga, Japan
dungdv@selab.is.ritsumei.ac.jp

Atsushi Ohnishi
Department of Computer Science
Ritsumeikan University
Kusatsu, Shiga, Japan
ohnishi@cs.ritsumei.ac.jp

***Abstract*—In using ontology to support requirements engineering, quality of elicited requirements depends on quality of requirements ontology, so a rule-based verification method of the correctness of requirements ontology has been proposed. However, in recent evaluation experiments, users of the method (ontology verifiers) described only a few new rules based on rule grammars and rule examples. That led to the number of correctly detected errors were not so high (less than 50% of the total number of errors). To improve our method, in this paper, we propose a rules customization mechanism in which simple specific rules are generated using pre-defined and customizable meta-rules. We expect that by using the improvement, ontology verifiers can easily and effectively generate and customize rules for verification of requirements ontology. The customization mechanism is illustrated through examples and a case study.**

***Index Terms*—verification of requirements ontology, ontology-based requirements elicitation.**

I. Introduction

Requirements elicitation is still a difficult task in requirements engineering because of the possibility of requirements elicitors' lack of domain knowledge. Knowledge-based requirements elicitation, especially recent ontology-based researches [1][2][3], is a promising solution to the problem. In ontology-based requirements elicitation, quality of elicited requirements depends on quality of requirements ontology. Though there are many researches about ontology-based requirements elicitation, there are not much studies about quality of requirements ontology. Therefore, we have proposed a rule-based verification method of the correctness of requirements ontology [4].

In [4], we have evaluated our proposed method through experiments. However, the results of using rule-based verification method of requirements ontology were still not so good: the number of correctly detected errors was less than 50% of the total number of errors, and subjects who participated in the experiments only described a few new rules. Since it was difficult for users of the method to specify new rules, in this paper, we propose a customization mechanism of rules in which rules can be generated automatically and suitably for specific requirements ontology. We expect that by applying rules customization, ontology verifiers would define rules more easily and would find more correctly detected errors.

The paper is organized as follows. The next section will briefly introduce our previous research in requirements ontology, then Section III presents rules customization mechanism. Section IV evaluates the improvement of our method through a case study. After that, Section V discusses related researches and compares with our work, and finally, Section VI arrives at a conclusion.

II. Background

In this section, we briefly touch upon the previous researches: requirements ontology model, ontology-based requirements elicitation method, and rule-based verification method of requirements ontology.

A. Requirements Ontology

Requirements ontology includes a hierarchy of functional requirements of softwares in a domain. Each functional requirement that includes a verb and several nouns becomes a node in the hierarchy. Attributes of functional requirements are: who (agent), when (time), where (location), why (reason or purpose), and how (method). The five attributes: who, when, where, why, and how are abbreviated as 4W1H attributes. Relations among functional requirements include: complement, supplement, inconsistency, redundancy, aggregation, and inheritance. Fig. 1 displays the meta-model of requirements ontology.

Fig. 2 shows an example of a part of requirements ontology of online web store softwares. The requirements ontology contains a hierarchy of functions of web stores; some of the

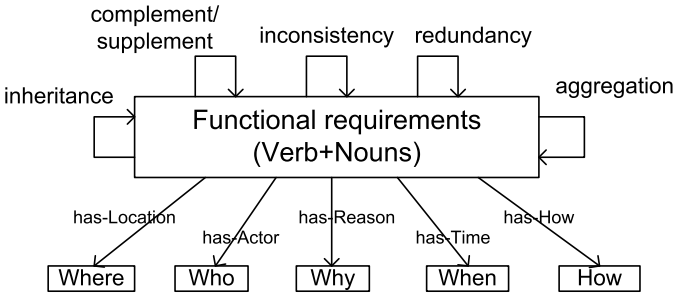

Fig. 1: Meta-model of requirements ontology[5].

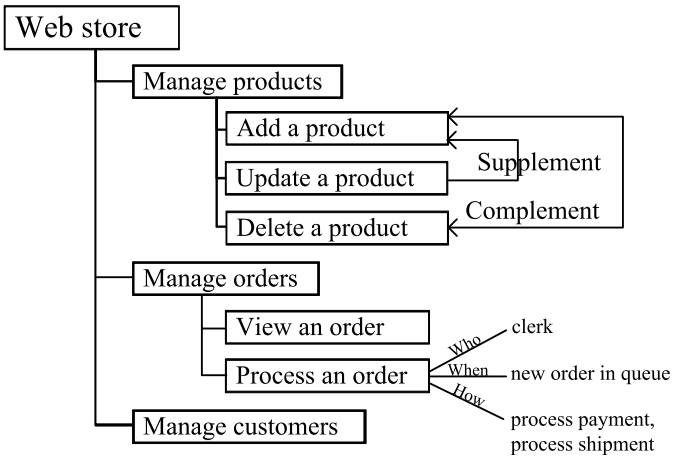

Fig. 2: A part of the requirements ontology of web stores

functions are: 'manage products', 'manage orders', and 'manage customers'. The function 'manage products' has several sub-functions such as: 'add a product', 'update a product', and 'delete a product'. The two functions 'add a product' and 'delete a product' have a relation of complement which means that they should both exist in a requirements specification. More detail descriptions of relations among functional requirements can be found in [5]. Fig. 2 also presents who, when, and how attributes of the function 'process an order', e.g., 'clerk' is who information of that function.

Requirements ontology can be built by analyzing user guide documents of existing software systems. To support construction and management of requirements ontology, we have developed a requirements ontology editor [4]. In the tool, requirements ontology is described using OWL (web ontology language) [6]. The detail description of construction of requirements ontology is out of scope of this paper.

B. Ontology-Based Requirements Elicitation

Requirements ontology can be used to support requirements elicitation [5][7]. We will explain concisely ontology-based requirements elicitation method through examples below.

Based on a hierarchy of functional requirements and relations among functional requirements in a domain, requirements of a new software in the same domain can be reasoned [5]. For example, in eliciting requirements for a new web store system, using the web store ontology as in Fig. 2, requirements elicitors can find that a web store software includes functions such as: 'manage products', 'add a product', 'manage orders', 'process an order', and so on. However, if the software has the function 'add a product', it should also have the function 'delete a product', because of the complementary relation between the two functions.

In addition, by referring to 4W1H attributes of functions in requirements ontology, we can detect lack or wrong description of 4W1H in elicited requirements [7]. For example, the function 'process an order' in the web store ontology has who attribute as 'clerk', so elicited requirements of the function of processing orders should have similar agent information.

C. Verification of the Correctness of Requirements Ontology

In order to elicit requirements of high quality using ontology-based method, we should guarantee the correctness of requirements ontology. In [4], we have proposed three types of rules for verification of requirements ontology. The three types of rules are: attribute rules, relation rules, and inference rules. Attribute rules and relation rules are for verification of the correctness of attributes of functions and relations among functions in requirements ontology, respectively. Inference rules are for reasoning an existence/ non-existence of a relation from two existing relations.

However, as described in Section I, the fact that subjects who participated in the evaluation experiments only described a few new rules led to the not so good results [4]. To overcome the difficult of specifying new rules, in this paper, we separate rules into two categories. One category contains simple rules and for verifiers' usage, and another category includes rules for generation of simple rules. Rules of the first category should be described easily and specifically suitable for each requirements ontology. We adopt attribute rule and relation rule from our previous research [4] into the first category, while inference rule is adjusted and put in the second category. The two categories of rules will be described in the next section.

III. Rules Customization

In rules customization mechanism, we focus on two categories of rules: meta-rules and specific rules. Meta-rules are used for generation of specific rules; specific rules are used for verification of specific requirements ontology. Both meta-rules and specific rules follow certain rule patterns. Patterns for specific rules should be simple so that users of the method (ontology verifiers) can easily apply specific rules. We define some terms related to rules as follows.

- Meta-rules: are used to generate specific rules.
- Specific rules: are used to verify specific ontology.
- Meta-rule patterns: are templates of meta-rules.
- Specific rule patterns: are templates of specific rules.
- Ontology verifiers: are people who are assigned to check quality of requirements ontology during ontology development. Ideally, ontology verifiers should have domain knowledge. However, if a domain expert has participated in development of requirements ontology, (s)he should not be assigned to verify the ontology. Either ontology verifiers have domain knowledge or not, they can use rules to support verification.

Rules customization mechanism is as follows. Meta-rules are specified in advance. Specific rules are generated using meta-rules with reference of requirements ontology and thesaurus in a domain. Ontology verifiers then can add, remove, or change the generated specific rules, providing that the customized rules follow specific rule patterns. Finally, specific rules are used to verify the correctness of requirements ontology as in [4]. Fig. 3 displays an overview of the new method.

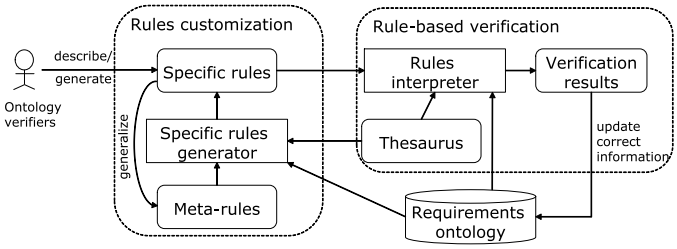

Fig. 3: Overview of rules customization.

Meta-rules can be formed from several example of specific rules. Based on several facts and/or specific rules, we propose a meta-rule. The meta-rule is then used to generated other specific rules.

A. Specific Rules

This sub-section presents three specific rule patterns and some examples of specific rules which follow the patterns.

Requirements ontology consists of functions, attributes of functions, and relations among functions. Therefore, errors in requirements ontology are: lack or redundancy of functions, lack or wrong description of function attributes, lack or wrong relations among functions. To support description of specific rules to detect such errors, we use three specific rule patterns: function rule pattern, attribute rule pattern, and relation rule pattern. Attribute rules and relation rules have been described in [4].

The first specific rule pattern ⟨SP1⟩ (function rule pattern) specifies the existence of functional requirements. The grammar of the first pattern of specific rules is shown below. Labels of bold fonts correspond to reserved words, while labels of italic fonts correspond to variables or values: [*Function*] is in square brackets, and (*Attribute*) is in parentheses. "{A|B|C}" means selection of items. "⟨ID⟩" is an identifier of a pattern or a rule.

[*Function*] **{should| should not} exist in ontology** ⟨SP1⟩

For example, a rule description, "[Process an order] should exist in ontology" ⟨SR1⟩, specifies the functional requirements of processing an order should exist in requirements ontology. With this pattern of rules we can detect both lack and redundancy of functional requirements in an ontology.

The second specific rule pattern ⟨SP2⟩ (attribute rule pattern) specifies the correctness of attributes of functional requirements. The grammar of the second pattern of specific rules is shown below. The term "4W1H" denotes "where| who| why| when| how".

(*Attribute*) **{should| should not} be** {*4W1H*} **information of** [*function*] ⟨SP2⟩

For example, a rule description, "(Clerk) should be who information of [process an order]" ⟨SR2⟩, specifies agent of the functional requirement of processing an order. With this pattern of rules we can detect both wrong attributes and lack of attributes of functional requirements in an ontology.

The third specific rule pattern ⟨SP3⟩ (relation rule pattern) specifies the correctness of relations between functional requirements in an ontology. The grammar of the third pattern of specific rules is shown below. The term "relation-name" denotes "complement| supplement| inconsistency| redundancy| inheritance| aggregation".

There {exists| does not exist} a relation of {*relation-name*} **between** [*function1*] **and** [*function2*] ⟨SP3⟩

For example, a rule description, "There exists a relation of complement between [add a product] and [delete a product]" ⟨SR3⟩, specifies a complement relation between the two functional requirements [add a product] and [delete a product]. With this pattern of rules we can detect wrong relations and lack of relations between functional requirements in an ontology.

We can use wildcard symbols and variables in rules. For example, a rule description, "(Customer) should not be who information of [process *]" ⟨SR4⟩, specifies that customer is the wrong agent of the group of functions of processing somethings in web store software. Another rule description, "There exists a relation of complement between [register X] and [cancel X]" ⟨SR5⟩, specifies a complementary relation between the two groups of functions: registration of something and cancellation of something (e.g., [register an account] and [cancel an account], [register an order] and [cancel an order]).

Table I lists examples of specific rules. The first five examples (⟨SR1⟩–⟨SR5⟩) have been explained above. The latter seven examples (⟨SR6⟩–⟨SR12⟩) are specific rules which are generated by meta-rules which will be explained in the subsection below.

B. Meta-Rules

We propose a meta-rule pattern for specification of meta-rules as follows.

If *condition-1* **and** *condition-2* **and** ... *condition-n* **then GENERATE** *a-specific-rule* ⟨MP1⟩

where *condition-i* is of the following types:

- an existence of a function in requirements ontology: "[*function1*] **{exists| does not exist} in ontology**"
- a relation of synonym or antonym between two verbs or two nouns: "**{synonym| antonym}***(word1, word2)*"
- a relation between two functions: "{*relation-name*}([*function1*], [*function2*])"
- an attribute of a function: "(*attribute1*) **is** {*4W1H*} **information of** [*function1*]"

a-specific-rule is a rule which follows one of the three specific rule patterns described above.

We provide some examples of meta-rules which follow the meta-rule pattern ⟨MP1⟩. Each below meta-rule is presented

TABLE I: Examples of specific rules.

ID	Specific rule	Pattern	Gen. by
⟨SR1⟩	[Process an order] should exist in ontology	⟨SP1⟩	
⟨SR2⟩	(Clerk) should be who information of [process an order]	⟨SP2⟩	
⟨SR3⟩	There exists a relation of complement between [add a product] and [delete a product]	⟨SP3⟩	
⟨SR4⟩	(Customer) should not be who information of [process *]	⟨SP2⟩	
⟨SR5⟩	There exists a relation of complement between [register X] and [cancel X]	⟨SP3⟩	
⟨SR6⟩	[Return books] should exist in ontology	⟨SP1⟩	⟨MR1⟩
⟨SR7⟩	[Insert a product] should not exist in ontology	⟨SP1⟩	⟨MR2⟩
⟨SR8⟩	[Add a commodity] should not exist in ontology	⟨SP1⟩	⟨MR3⟩
⟨SR9⟩	(Clerk) should be who information of [add a product]	⟨SP2⟩	⟨MR4⟩
⟨SR10⟩	There exists a relation of complement between [reserve a product] and [purchase a product]	⟨SP3⟩	⟨MR5⟩
⟨SR11⟩	There exists a relation of complement between [delete a product] and [add a product]	⟨SP3⟩	⟨MR6⟩
⟨SR12⟩	There exist a relation of complement between [select X] and [remove X]	⟨SP3⟩	⟨MR7⟩

in a box and is followed by an explanation and an example of a generated specific rule.

In the requirements ontology of online web stores, we see that there exist functions [add a product] and [delete a product], and the two verbs 'add' and 'delete' have contradict meaning. We form a meta-rule ⟨MR1⟩ which specifies a condition when a function exists in a requirements ontology and there exists a contradict action of the function, then a specific rule of the presence of the contradict function in ontology is generated.

If antonym(*verb1, verb2*) **and** [*verb1 noun1*] **exists in ontology and** [*verb2 noun1*] **does not exist in ontology then GENERATE** "[*Verb2 noun1*] **should exist in ontology**" ⟨MR1⟩

We show an example of using the above meta-rule ⟨MR1⟩ to generate other specific rules. Since two verbs 'borrow' and 'return' are antonym, if function [borrow books] exists in a library ontology, but function [return books] does not exist in the library ontology, ⟨MR1⟩ generates a specific rule: "[Return books] should exist in ontology" ⟨SR6⟩.

Another meta-rule ⟨MR2⟩ specifies that two similar functions (having synonym verbs and the same noun) should not exist concurrently in an ontology as follows.

If synonym(*verb1, verb2*) **and** [*verb1 noun1*] **exists in ontology and** [*verb2 noun1*] **exists in ontology then GENERATE** "[*Verb2 noun1*] **should not exist in ontology**" ⟨MR2⟩

For example, because 'add' and 'insert' are synonym, if function [add a product] exists in a web store ontology, function [insert a product] should not exist in the web store ontology. Therefore, by applying the above meta-rule to requirements ontology of online web store, a specific rule "[Insert a product] should not exist in ontology" ⟨SR7⟩ is generated. It illustrates the customization of rules through generation of specific rules because applying the above meta-rule to another requirements ontology will result another specific rule.

The above meta-rule ⟨MR2⟩ describes the redundancy of functions in ontology because of similar verbs. A meta-rule ⟨MR3⟩ below specifies such redundancy in case of synonymic nouns.

If synonym(*noun1, noun2*) **and** [*verb1 noun1*] **exists in ontology and** [*verb1 noun2*] **exists in ontology then GENERATE** "[*Verb1 noun2*] **should not exist in ontology**" ⟨MR3⟩

The above meta-rule ⟨MR3⟩ states that two functions having the same verb and synonym nouns should not both exist in a requirements ontology. For example, since two nouns 'product' and 'commodity' are synonym, if function [add a product] has existed in a web store ontology, function [add a commodity] should not exist in the ontology. In this case, a specific rule "[Add a commodity] should not exist in ontology" ⟨SR8⟩ is generated.

The above three meta-rules ⟨MR1⟩–⟨MR3⟩ are capable of generation of function rules. For an example of generation of attribute rules, we see that in requirements ontology, who attribute (agent) of parent function is often similar to who attribute of children functions. We form a meta-rule ⟨MR4⟩ as follows.

If (*attribute1*) **is who information of** [*function1*] **and aggregation**([*function2*], [*function1*]) **then GENERATE** "(*Attribute1*) **should be who information of** [*function2*]" ⟨MR4⟩

The above meta-rule ⟨MR4⟩ specifies similar who attributes of aggregated functions in requirements ontology. For example, if (clerk) is who information of function [manage products] in web store ontology, and function [manage products] includes several sub-functions such as: [add a product], [update a product], and [delete a product] (these three sub-functions aggregate to the parent function), then (clerk) should also be who information of the three sub-functions. In this case, attribute rules such as "(Clerk) should be who information of [add a product]" ⟨SR9⟩ are generated (supplier/customer are not who information of the function [add a product] because they are not allowed to change list of products in online store).

In following examples, we present some meta-rules which are capable of generation of relation rules. Firstly, we note that the relation of complement is transitive, which means that complement(X,Y) and complement(Y,Z) will lead to complement(X,Z). We describe a meta-rule of transitiveness of complementary relation as follows.

If complement([*function1*], [*function2*]) **and complement**([*function2*], [*function3*]) **then GENERATE "There exists a relation of complement between** [*function1*] **and** [*function3*]" ⟨MR5⟩

For example, if a function [reserve a product] is complementary to a function [notify of availability of a product], and the function [notify of availability of a product] is complementary to a function [purchase a product], then a relation rule "There exists a relation of complement between [reserve a product] and [purchase a product]" ⟨SR10⟩ is generated.

Secondly, we also see that the complementary relation is a two-way relation: complement(X, Y) will lead to complement(Y, X). We form a meta-rule of two-way relation below.

If complement([*function1*], [*function2*]) **then GENERATE "There exists a relation of complement between** [*function2*] **and** [*function1*]" ⟨MR6⟩

For example, if the function [add a product] is complementary to the function [delete a product], then the reversed relation also hold; a relation rule "There exists a relation of complement between [delete a product] and [add a product]" ⟨SR11⟩ is generated.

Thirdly, we note that in requirements ontology of online web store, there exists a relation of complement between two functions [add a product] and [delete a product], and there exist a contradict meaning between two verbs 'add' and 'delete'. We propose a meta-rule that a relation of antonym between two verbs will lead to a relation of complement between two functions as follows.

If antonym(*verb1, verb2*) **and** [*verb1 noun1*] **exists in ontology and** [*verb2 noun1*] **exists in ontology then GENERATE "There exists a relation of complement between** [*verb1 noun1*] **and** [*verb2 noun1*]" ⟨MR7⟩

For example of using above meta-rule to generate other specific rules, since the two verbs 'select' and 'remove' are antonym, a rule "There exist a relation of complement between [select X] and [remove X]" ⟨SR12⟩ is generated.

This sub-section has presented a meta-rule pattern and seven examples of meta-rules (We do not list all our proposed meta-rules in this paper for space limitation). Examples of specific rules which are generated using the meta-rules were also illustrated. Generated specific rules are then used to verify the correctness of requirements ontology [4].

C. Generation of Specific Rules Using Meta-Rules

To apply a meta-rule, which follows the meta-rule pattern ⟨MP1⟩, we firstly extract all *conditions* in the if clause. We secondly search for objects (functional requirements, their attributes, and their relations in requirements ontology; words in domain thesaurus) that satisfy the *conditions*. If satisfied objects are found, we thirdly generate *specific rules* based on information from the objects. If satisfied objects are not existed, we exit without generation of *specific rules*. The steps for generation of *specific rules* using a meta-rule are as follows.

1) Extract *conditions* in the if clause.
2) Determine search criteria of the *conditions*: types of objects to be searched, locations to search (thesaurus or ontology), and matching values.
3) Search for objects which match the search criteria of one *condition*. If objects are found, go to step 4. If not found, go to step 6.
4) If there are more *conditions*, update the search criteria of other *conditions* (update the matching values), then repeat step 3 with another *condition*. If there is no more *condition*, go to step 5.
5) Generate *specific rules* using information of the found objects.
6) Exit.

There are two ways to implement the searching of objects which match *conditions*. One way is to traverse through hierarchy tree of requirements ontology which is constructed by requirements ontology editor [4]. Another way is to use Prolog to support reasoning of satisfied objects.

We have been upgrading a rule-based verification tool of requirements ontology in [4] to a prototype of customizable rule-based verification tool with implementation using Java and Prolog. Fig. 4 shows a screenshot of the tool with verification of requirements ontology of web store. The left side of the figure depicts a list of meta-rules. When a meta-rule is selected, the right side shows a list of generated specific rules. Under each specific rule, verification results of that specific rule is displayed. For example, in Fig 4, the meta-rule ⟨MR2⟩ is selected, and specific rules in the form of function rule pattern ⟨SP1⟩ are generated. In this example, requirements ontology of web store in Fig. 2 was added with several redundant functions such as: [insert product], [edit product]. Using meta-rule ⟨MR2⟩, the tool generated rules that those functions should not exist in requirements ontology, as illustrated in the right side of Fig. 4.

IV. Case Study

This section illustrates the potential effectiveness of rules customization to rule-based verification method of requirements ontology through a case study. Rules customization is applied to generate specific rules for verification of a requirements ontology. Results by new mechanism are then analyzed and compared with results by previous method [4].

We used rules customization mechanism to describe rules for verification of a requirements ontology of library systems. The library ontology included 104 function nodes, 44 relations among nodes (not including aggregation relations), and 89 information of 4W1H attributes. The ontology was constructed from manuals of several existing library systems, and intentionally contained errors such as duplication of functions, lack or wrong descriptions of 4W1H attributes, lack or wrong relations among functions. The ontology has been used in an experiment

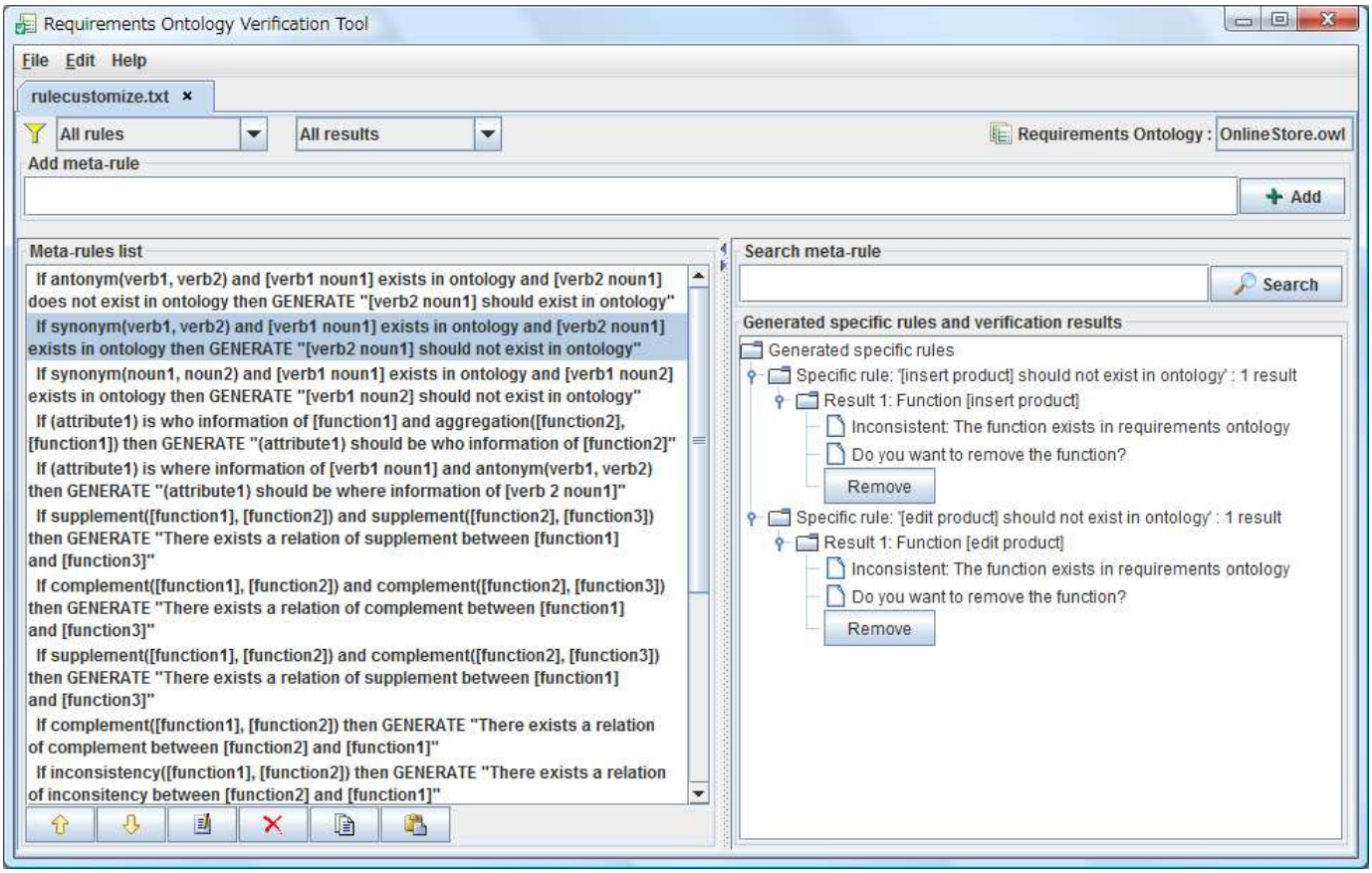

Fig. 4: Screenshot of verification tool of requirements ontology.

in the past which evaluated rule-based verification method of requirements ontology (without rules customization) [4]. Sixty minutes was assigned to generate specific rules and find errors in the ontology.

A. Results of Applying Rules Customization

A list of 15 meta-rules (including ⟨MR1⟩–⟨MR7⟩) was used to generated specific rules. By referring the library ontology and a thesaurus, 42 specific rules were generated using meta-rules. Some of the meta-rules and generated specific rules are listed in Table II. For example, the meta-rule ⟨MR2⟩ generated a specific rule: "[Delete a book] should exist in ontology", and we used that specific rule to check whether the function [delete a book] exist in library ontology. If the function did not exist, an error would be reported. Similarly, the meta-rule ⟨MR4⟩ generated another specific rule: "(User) should be who information of [reserve books]". That specific rule was used to check the correctness of who attribute of the function [reserve books] in library ontology.

Forty two specific rules are generated. They can be classified into 5 function rules, 19 attribute rules, and 18 relation rules. Table III summarizes the number specific rules and the number of errors detected by using the specific rules. Totally, using the 42 generated specific rules, we could propose 39 problems (potentially detected errors). In the experiment in the past which used the same ontology, there were 47 correctly detected errors. In the 39 potentially detected errors by new rules customization mechanism, 31 of them were correctly detected errors, but there are still eight wrongly detected errors and

TABLE II: Examples of meta-rules and generated specific rules for the library ontology.

Meta-rules	Generated specific rules
If antonym(verb1, verb2) and [verb1 noun1] exists in ontology and [verb2 noun1] does not exist in ontology then GENERATE "[Verb2 noun1] should exist in ontology" ⟨MR2⟩	[Delete a book] should exist in ontology
	[Delete an user] should exist in ontology
If (attribute1) is who information of [function1] and aggregation([function2], [function1]) then GENERATE "(Attribute1) should be who information of [function2]" ⟨MR4⟩	(User) should be who information of [reserve books]
	(Staff) should be who information of [loan books]
	(Administrator) should be who information of [create a group]
If supplement([function1], [function2]) and complement([function2], [function3]) then GENERATE "There exists a relation of supplement between [function1] and [function3]" ⟨MR8⟩	There exists a relation of supplement between [extend books] and [return books]
	There exists a relation of supplement between [print receipt] and [take back books]

TABLE III: The number of generated specific rules and the number of detected errors in library ontology by applying rules customization.

	# specific rules	# potentially detected errors	# correctly detected errors
Function rules	5	2	1
Attribute rules	19	16	14
Relation rules	18	21	16
Total	42	39	31

TABLE IV: Precision metrics and recall metrics of results.

	Subject	Precision		Recall	
		Metrics	%	Metrics	%
with rules customization	A	31 / 39	79.4	31 / 47	65.9
without rules customization	B	14 / 17	82.3	14 / 47	29.7
	C	11 / 12	91.6	11 / 47	23.4

16 undetected errors (we discuss the reasons later in section IV-C).

B. Comparison with Previous Method

We compared the above results with results by not using rules customization in the past experiment. In that experiment, two subjects who were undergraduate students had checked the same library ontology. Two subjects (denoted as B, C) had been trained of the rule-based verification method of requirements ontology. The subject were allowed 60 minutes to define rules and find errors in the library ontology, using previous verification method in [4].

We use two metrics to compare the results of detected errors in requirements ontology:

- Recall: the number of correctly detected errors / the total number of errors.
- Precision: the number of correctly detected errors / the number of detected errors.

Table IV summarizes recall and precision metrics of results. The table shows that recall metrics of results by using rules customization was higher than recall metrics of results by not using rules customization. However, the precision metrics of results with rules customization was less than precision metrics of results without rules customization. It is said that when recall tends to increases, precision would be decreased, and vice versa [8].

The above analysis shows that in this case study, rules customization has positive effect in rule-based verification method of requirements ontology. Our usage of the mechanism helps improve the percentage and total of correctly detected errors. Based on errors found, ontology verifiers will revise and improve the quality of requirements ontology.

C. Discussion

This sub-section presents some issues in current rules customization mechanism. We discuss types of specific rules which are difficult to be generated and weakness of the method in the case study.

(Issue 1: we should use domain thesaurus). Some meta-rules use relations between words (synonym, antonym) to reason relations between functional requirements. A word can have a lot of synonyms and antonyms, so if we used common thesaurus, the number of generated specific rules would be huge. That is why we should use a domain thesaurus which contains only limited words which correspond to the domain.

(Issue 2: lack of word relations). Synonym can reveal relation of redundancy between functions; antonym can reveal relation of complement or inconsistency between functions. However, there are function relations which cannot be reasoned from synonym or antonym of verbs or nouns. For example, there is a supplementary relation between function [update X] and function [add X], but there is not a word relation between the two verbs 'add' and 'update' in common thesaurus. To solve this problem, there should exist a new type of word relation between the two verbs in domain thesaurus, instead of synonym or antonym. This word relation can be retrieved from an upper ontology such as WordNet [9]. In addition, sometimes it is difficult to reason function relations such as redundancy between [reserve books] and [make reservation of books] because the two verbs 'reserve' and 'make' have no relation. In that case, we can use natural language parsers (e.g., OpenNLP library [10]) to identify dependencies between words in a sentence.

(Issue 3: lack of information in ontology). Specific rules are generated using meta-rules with reference to domain thesaurus and requirements ontology, so if there is no relevant information in requirements ontology, specific rules cannot be generated.

(Issue 4: wrongly generated specific rules). Wrong information in requirements ontology could lead to semantically wrongly generated specific rules. For example, the meta-rule ⟨MR5⟩—an inference rule—generates a relation rule of required relation from two existing relations. If the existing relations are incorrect, the generated rule will be semantically incorrect. That is a reason why we need human assessment of potentially detected errors, and not all of them are correctly detected errors, as in the case study. Metamorphic testing [11] of requirements ontology would be a solution to this issue in our future research.

(Issue 5: lack of attribute rules). Of the 15 meta-rules which were used to generate specific rules in the case study, only two meta-rules were for generation of attribute rules. In our opinion, writing meta-rules for generation of attribute rules is not easy. It could be due to the complexity of natural languages which are used to described 4W1H information. Functional requirements contains only verbs and nouns, and relations among verbs and nouns can be used to reason out relations among functions. Instead, 4W1H attributes is described with many types of words: verbs, nouns, prepositions, adjectives, and adverbs, so it is difficult to retrieve such many types of relations among many types of words. Therefore, finding ways

to describe meta-rules for generation of attribute rules would also be one of our future research topics.

(Issue 6: weakness of method). Though applying rules customization in the case study, there were still 16 undetected errors and eight wrongly detected errors. The reasons of this weakness are in the issues discussed above.

For the 16 undetected errors, obviously it was due to the lack of specific rules to detect them. The lack of specific rules was because of three reasons: lack of relations among words (issue 2), lack of meta-rules (partly issue 5), and lack of information in requirements ontology (issue 3). We can solve issue 2 as described above. By studying the undetected errors, we can form new meta-rules to use in the future. Issue 3 is clearly a weak point of our current method, and should be studied more in the future.

The eight wrongly detected errors were due to issue 4 (wrong information in requirements ontology leads to semantically wrongly generated specific rules). Although all 42 generated specific rules were grammatically correct, but eight of them were semantically incorrect. As previously described, metamorphic testing would be a solution to this problem in future work.

(Issue 7: why rules customization?). The question "Why we need to separate generation of rules and verification of ontology?" would be raised. We can use meta-rules to detect errors directly, instead of using specific rules. One reason is that we should control verification process in order to prevent wrongly detection of errors. Rules such as meta-rules are complicated. It is difficult for ontology verifiers to describe meta-rules and to control verification process using meta-rules. Another reason is simplicity. Specific rules are simple, and can be generated quickly and automatically by tool. Verification task using specific rules is simple, regardless of doing by hand or by tool. In short, we divide a complicated task (verify with general rules) into two simple tasks (generate specific rules, verify with specific rules).

V. Related Works

A number of researchers have explored the usage of ontology to support requirements elicitation, notably Kaiya and Saeki proposed ontology-based requirements elicitation method [1]. Kluge et al. focused on helping business people to choose suitable software products using ontology [12]. The elicitation method by Zong-yong and colleagues divides ontology into multiple stages: domain ontology, task ontology, and application ontology [2]. Dobson, Hall, and Kotonya proposed non-functional requirements ontology to be used to discover non-functional requirements [13]. Xiang et al. divided initial requirements to a list of goal; each goal is narrowed down to a list of sub-goals [3]. Similarly, Liu and colleagues used ontology model consisted of actor, goal, task to do reasoning [14]. The above researches focused on requirements elicitation using ontology, but their methods did not support detection of errors in requirements ontology.

Some researches provided methods to detect errors in OWL ontologies [15][16]. In our opinion, requirements ontology should be described in OWL format—a common knowledge description language [6]. Therefore, requirements ontology can be verified by methods which support reasoning with OWL such as in [15][16]. However, we do not focus on implementation ways; instead, our research focuses on support requirements engineers to improve the quality of requirements ontology.

VI. Conclusion

We proposed a rules customization mechanism for improvement of the rule-based verification method of requirements ontology. The customization mechanism is mainly based on generation of specific rules suitably for each domain from meta-rules. The comparison of detection of errors in a requirements ontology by the new improvement and by the previous method shows that rules customization contributes positive effect to the rule-based verification method. In the future, we plan to study on existing issues of the method (discussed in Sub-sect. IV-C) and evaluate the method through experiments.

References

[1] H. Kaiya and M. Saeki, "Using domain ontology as domain knowledge for requirements elicitation," in *Proc. RE'06*, 2006, pp. 186–195.

[2] L. Zong-yong, W. Zhi-xue, Y. Ying-ying, W. Yue, and L. Ying, "Towards a multiple ontology framework for requirements elicitation and reuse," in *Proc. COMPSAC'07*, 2007, pp. 189–195.

[3] J. Xiang, L. Liu, W. Qiao, and J. Yang, "Srem: A service requirements elicitation mechanism based on ontology," in *Proc. COMPSAC'07*, 2007, pp. 196–203.

[4] D. V. Dzung, B. Q. Huy, and A. Ohnishi, "Rule-based verification method of requirements ontology," *IEICE Transactions on Information and Systems*, vol. E97-D, no. 5, pp. 1017–1027, 2014.

[5] D. V. Dzung and A. Ohnishi, "Ontology-based reasoning in requirements elicitation," in *Proc. SEFM'09*, 2009, pp. 263–272.

[6] World Wide Web Consortium (W3C), "OWL Web Ontology Language Overview," http://www.w3.org/TR/owl-features/, accessed: 2012-1-16.

[7] D. V. Dzung and A. Ohnishi, "A verification method of elicited software requirements using requirements ontology," in *Proc. APSEC'12*, 2012, pp. 553–558.

[8] R. A. Baeza-Yates and B. Ribeiro-Neto, in *Modern Information Retrieval*. Addison-Wesley, 1999, p. 81.

[9] WordNet: A lexical database for English, http://wordnet.princeton.edu/, accessed: 2014-3-10.

[10] OpenNLP library, http://opennlp.sourceforge.net/, accessed: 2011-10-25.

[11] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Tech. Rep. HKUST-CS98-01, 1998.

[12] R. Kluge, T. Hering, R. Belter, and B. Franczyk, "An approach for matching functional business requirements to standard application software packages via ontology," in *Proc. COMPSAC'08*, 2008, pp. 1017–1022.

[13] G. Dobson, S. Hall, and G. Kotonya, "A domain-independent ontology for non-functional requirements," in *Proc. ICEBE'07*, 2007, pp. 563–566.

[14] L. Liu, Q. Liu, C. hung Chi, Z. Jin, and E. Yu, "Towards a service requirements ontology on knowledge and intention," in *Proc. QSIC'06*, 2006, pp. 452–462.

[15] A. Kalyanpur, B. Parsia, E. Sirin, and J. A. Hendler, "Debugging unsatisfiable classes in owl ontologies," *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 4, pp. 268–293, 2005.

[16] H. Wang, M. Horridge, A. L. Rector, N. Drummond, and J. Seidenberg, "Debugging owl-dl ontologies: A heuristic approach," in *Proc. ISWC'05*, 2005, pp. 745–757.