PAPER Special Section on Knowledge-Based Software Engineering

Rule-Based Verification Method of Requirements Ontology

Dang Viet DZUNG[†a)], **Bui Quang HUY**[††b)], *Nonmembers*, *and* **Atsushi OHNISHI**[†c)], *Member*

SUMMARY There have been many researches about construction and application of ontology in reality, notably the usage of ontology to support requirements engineering. The effect of ontology-based requirements engineering depends on quality of ontology. With the increasing size of ontology, it is difficult to verify the correctness of information stored in ontology. This paper will propose a method of using rules for verification the correctness of requirements ontology. We provide a rule description language to specify properties that requirements ontology should satisfy. Then, by checking whether the rules are consistent with requirements ontology, we verify the correctness of the ontology. We have developed a verification tool to support the method and evaluated the tool through experiments.

key words: *requirements ontology, ontology verification, requirements elicitation and analysis*

1. Introduction

It is difficult to elicit correct and complete software requirements because of the complexity of natural language in human to human communication, and the tactic to extract knowledge from software stakeholders. Experiences in requirements engineering will improve the ability of requirements analysts. Nevertheless, an analyst who has good skill of requirements techniques but does not have domain knowledge of target system will not elicit requirements of high quality and will not get sufficient requirements. Knowledge-based approaches, especially with regard to ontology-based methods, has the potential to facilitate requirements engineering. On using ontology to support requirements elicitation, quality of elicited requirements and requirements specification depends on quality of ontology. Therefore, after construction, ontology needs to be verified before using in requirements elicitation. Though there are many researches about ontology-based requirements elicitation (Sect. 2), there is no research about verifying quality of requirements ontology (ontology which supports requirements engineering). In this paper, we propose a verification method of the correctness of requirements ontology in order to improve the quality of the ontology.

The paper is organized as follows. The next section discusses related researches and compares with our work. After that, Sect. 3 presents our requirements ontology model and the usage of requirements ontology to support requirements engineering. Then Sect. 4 describes the verification method of requirements ontology using three types of rules. Section 5 presents two experiments to evaluate our proposed method, and finally, Sect. 6 arrives at a conclusion.

Manuscript received June 24, 2013.
Manuscript revised October 24, 2013.

[†]The authors are with the Department of Computer Science, Ritsumeikan University, Kusatsu-shi, 525–8577 Japan.
[††]The author is with the Rikkeisoft Co., Ltd, Hanoi, Vietnam.
a) E-mail: dungdv@selab.is.ritsumei.ac.jp
b) E-mail: huybq@rikkeisoft.com
c) E-mail: ohnishi@cs.ritsumei.ac.jp

DOI: 10.1587/transinf.E97.D.1017

2. Related Works

There are several related works in requirements engineering using ontology. Breitman and Leite used language extended lexicon (LEL) to represent terms and phrases in application language, and then proposed framework to construct ontology [1]. Similarly, Zhang, Mei, and Zhao provided feature diagram to represent domain knowledge [2]; Bao et al. proposed maintenance framework for domain ontology with focus on formal representation of process changes [3]. The above three researches focused on construction of ontology, but they did not focus on verification of quality of ontology with rules.

A number of researchers have explored the usage of ontology to support requirements elicitation, notably Kaiya and Saeki proposed ontology-based requirements elicitation method; their method measures the quality of elicited requirements [4]. Kluge et al. described business requirements and software characteristics in terms of ontology [5]. Their method helps business people to compare and match new functional requirements to functions of available commercial software products, and choose a suitable one. The method by Zong-yong and colleagues divides ontology into multiple stages: domain ontology, task ontology, and application ontology [6]. Domain users participate in the requirements elicitation by fill in the questionnaires directed by ontology. Dobson, Hall, and Kotonya proposed non-functional requirements ontology to be used to discover non-functional requirements [7]. Xiang et al. divided initial requirements to a list of goal; each goal is narrowed down to a list of subgoals [8]. Using relation among goals, they can refine initial requirements. Similarly, Liu and colleagues used ontology model consisted of actor, goal, task to do reasoning [9]. The above researches focused on requirements elicitation using ontology, but their methods did not support detection of errors in ontology.

Some researches provided methods to detect errors in OWL ontologies [10], [11]. In our opinion, requirements ontology should be described in OWL format—a common knowledge description language. Therefore, requirements

ontology can be verified by methods which support reasoning with OWL such as in [10], [11]. In addition, some parts of our method can also be implemented using existing tools [12]–[15]. However, we do not focus on implementation ways; instead, our research focuses on support requirements engineers to improve the quality of requirements ontology.

In concern for improvement of quality of requirements ontology, Huy and Ohnishi proposed a verification method of the correctness of requirements ontology using three types of rules [16]. A rule language and prototype system in Japanese was provided, but evaluation was not done. This paper extends Huy and Ohnishi's paper [16] with development of verification tool based on English and providing evaluation with the tool.

3. Requirements Ontology and Its Usage

This section will introduce requirements ontology model and describe ontology-based requirements elicitation method with a simple example.

3.1 Requirements Ontology

Domain ontology is an ontology of a certain domain knowledge. Requirements ontology is a type of domain ontology, but it is specialized for supporting requirements elicitation. It consists of functions structure and attributes of functions of software systems. Our requirements ontology meta-model is illustrated in Fig. 1. Each functional requirement that includes one verb and several nouns becomes a node in requirements ontology, and relations including inheritance and aggregation can be represented with a functional structure of systems of a certain domain. We provide other four relations: complement, supplement, inconsistency, and redundancy which are explained below.

- Complement: If functional requirements A and B have a relation of complement, when A is deleted, B also has to be deleted. In addition, when A is added, it is necessary to add B. The opposite also holds.
- Supplement: If functional requirement A supplements functional requirement B, when B is deleted, A also has to be deleted. In addition, B can appear independently in requirements specification, while A cannot appear independently.
- Inconsistency: When functional requirements A and B are inconsistent, both A and B should not exist in a software requirements specification (SRS). In other words, if A exists in an SRS, then B should not exist in the SRS and vice versa. Using this relation, we can detect inconsistent requirements.
- Redundancy: When functional requirements A and B are redundant, both A and B can exist in an SRS, but modification of only A may lead to be inconsistent between modified A and B. So, we can detect potential inconsistency with this relation between two functional requirements and modification of just one of them.

Functional requirements can include other factors such as the agent of the function (who), location of the function (where), time (when), reason (why), and method (how). These factors (who, where, when, why, how) are called 4W1H attributes of functional requirements. The 4W1H attributes are illustrated in bottom of Fig. 1.

Figure 2 shows an example of requirements ontology of library systems. In Fig. 2 we omit 4W1H attributes for space limitations. The ontology represents that a library system has several functions: searching books, borrowing books, returning books and so on. The function searching books has sub-functions, such as searching new available books, searching books by specified author name, etc. The function borrowing books and function returning books are complementary. In addition, the function renewing books supplements the function borrowing books.

Requirements ontology, such as in Fig. 2, can be built by analyzing manual documents of existing software systems in the application domain. User-guide documents usually includes list of functions of software systems and guideline about how to use the functions, so we can extract from these documents list of functions and usage context of functions: agent, method, time, location, reason. This information is used to build requirements ontology. As described above, a functional requirement in requirements ontology can be expressed with verb (action) and nouns (objects of the action). Specific nouns in manual documents cannot be

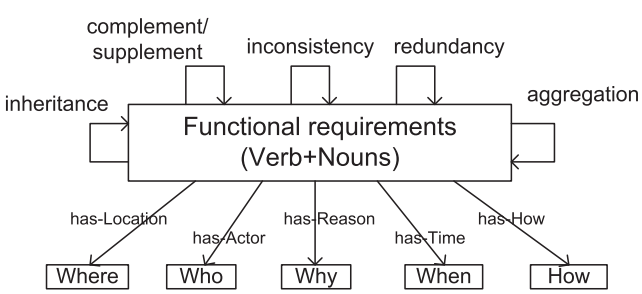

Fig. 1 Meta-model of requirements ontology.

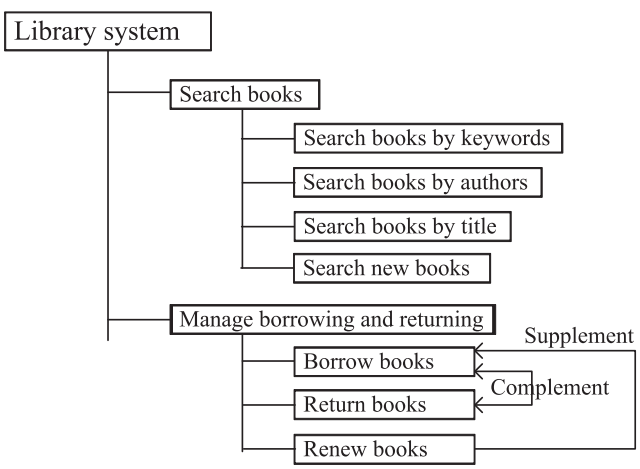

Fig. 2 A part of requirements ontology of library system.

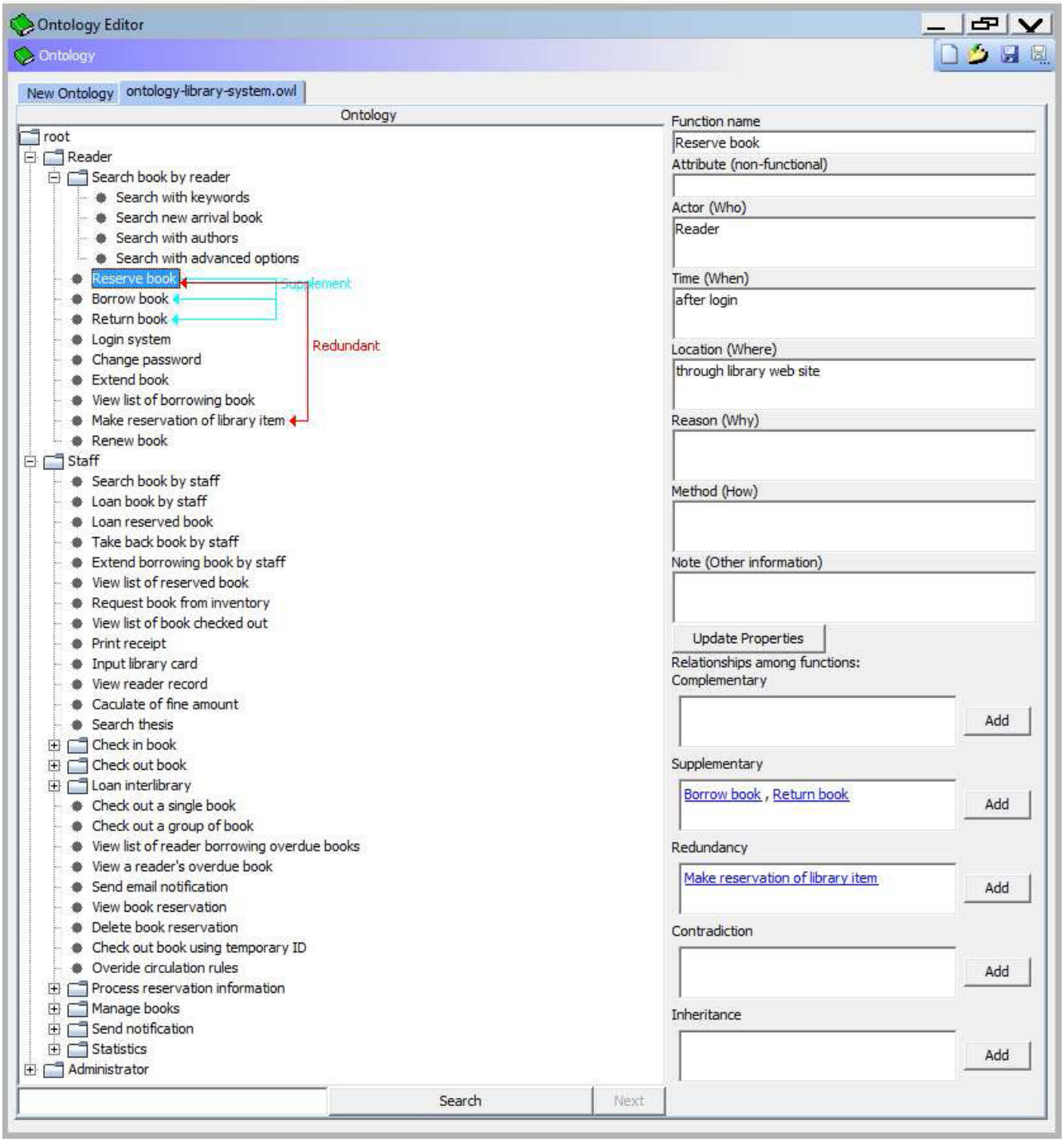

Fig. 3 Screenshot of requirements ontology editor.

adopted as nouns in functional requirements, because specific nouns may not be applied to another SRS. For example, "station staff" cannot be used except for a specific system's requirements; however, non-specific noun such as "operator" can be used. Hence, we introduce abstraction for specific nouns and abstracted nouns are used in functional requirements in ontology [17]. For instance, the noun "station staff" of "train ticket reservation system" can be abstracted such as "operator", so it can be reused in development of requirements ontologies for "reservation system".

To support construction and management of requirements ontology, an ontology editor tool has been developed. This tool was written with Java, and was two person-months product; the number of source code lines was 7,576. Requirements ontology is described with OWL [14]. Figure 3 shows a screenshot of the ontology editor which allows cre-

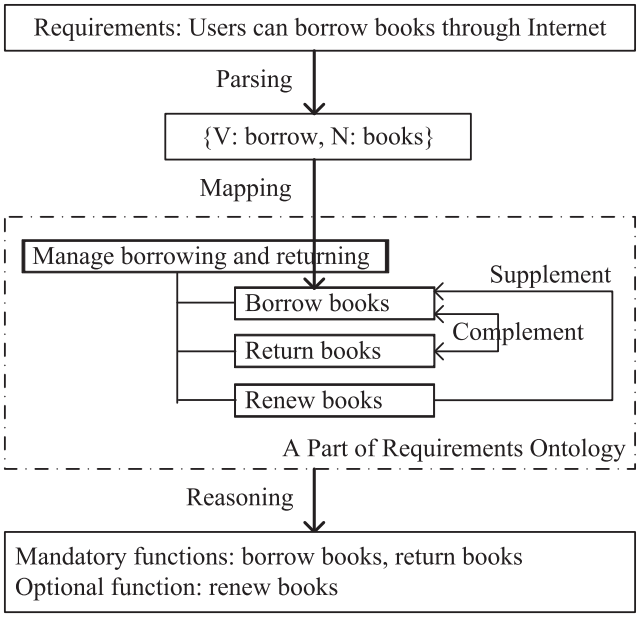

Fig. 4 An example of checking requirements with ontology.

ating and managing requirements ontology. In the figure, the visual view of requirements ontology is on the left, and the attributes of functions and relations among functions are on the right.

3.2 Usage of Requirements Ontology

Requirements ontology can be used to support requirements elicitation or requirements verification. Elicited requirements or requirements specification are compared with requirements ontology to find lacking functions or incorrect descriptions of functions. Based on detected errors, analysts can revise and improve quality of software requirements.

The reasoning process of errors in software requirements is as follows. Firstly, we map requirements to function nodes in ontology. For examples, a requirement "User can borrow books through Internet" is mapped to the node "Borrow books" in the ontology model, as illustrated in Fig. 4. After mapping, we use relations in ontology for reasoning errors in requirements list. In Fig. 4, using the relation of complement, the function "Borrow books" and the function "Return books" should be added to requirements list. Similarly, through relation of supplement, the function "Renew books" is optionally added. For optional functions, requirements team can discuss with customers whether adding them to requirements specification. After doing so, the requirements list will be more complete. In this way, we can detect lack of indispensable requirements in requirements elicitation and improve the correctness of software requirements specifications [18], [19].

Requirements ontology supports improving quality of software requirements, so quality of software requirements depends on quality of requirements ontology. After requirements ontology is built, it needs to be checked for correctness. The next section will present a method of verification of the correctness of requirements ontology.

4. Verification of Requirements Ontology with Rules

Requirements ontology is a representation of domain knowledge about requirements, so requirements ontology should be consistent with domain rules. Domain rules specify constraints and conditions that requirements belonging to each domain should follow. The increasing size of requirements ontology makes it difficult to guarantee the correctness of it (For example, in a team where each person develops a part of ontology, and then they merge ontology parts together). A verification method of requirements ontology and a supporting tool are a solution for this problem.

In this research, we propose a method of verification of the completeness and the consistency of a requirements ontology by checking the consistency between the requirements ontology and rule descriptions that specify the correctness of the ontology. Firstly, we describe our rule language to specify the correctness of a requirements ontology.

4.1 Rules Description Language

Requirements ontology consists of: functional requirements, attributes of functional requirements, and relations among functional requirements. Therefore, there are several types of errors that might occur with requirements ontology: (1) lack of functional requirements, (2) lack or wrong descriptions of attributes, (3) lack or wrong relations. To verify the correctness of requirements ontology, we provide three types of rules: attributes rules, relations rules, and inference rules. The first type of rule statement is to specify the correctness of attributes of functional requirements in an ontology. The second type is to specify the correctness of relations between functional requirements. The last type is to specify inference between functional requirements.

The first type of rules (attribute rules) specifies the correctness of attributes of functional requirements. The grammar of the first type of rules is shown below. Labels of bold fonts correspond to reserved words, while labels of italic fonts correspond to identifiers. "[A|B|C]" means selection of items.

Attribute-name **[should| should not] be [where| who| why| when| how] information of** *function*

For example, a rule description, "Users should be who information of borrow books" (#1) (examples of rules are shown in Table 1), specifies agent of the functional requirements of the borrowing books. With this type of rules we can detect both wrong attributes and lack of attributes of functional requirements in an ontology.

The second type of rules (relation rules) specifies the correctness of relations between functional requirements in an ontology. The grammar of the second type of rule is shown below.

Table 1 Examples of rules.

R#	Rule	Rule type
#1	Users should be who information of borrow books	first type
#2	There exists a relation of complement between borrow books and return books	second type
#3	If complement(X, Y) and complement(Y, Z) then ADD complement(X, Z)	third type
#4	If complement(X, Y) and supplement(X, Y) then REMOVE supplement(X, Y)	third type
#5	Student should not be who information of * scores	first type
#6	There exists a relation of complement between register * and cancel *	second type
#7	There exists a relation of complement between register X and cancel X	second type

There exists a relation of [complement| supplement| inconsistent| redundant] between *function1* **and** *function2*

For example, a rule description, "There exists a relation of complement between borrow books and return books" (#2), specifies a complement relation between the two functional requirements "borrow books" and "return books". With this type of rules we can detect wrong relations and lack of relations between functional requirements in an ontology.

The third type of rules (inference rules) specifies conditions and actions on relations among functional requirements. The grammar of the third type of rules is shown below.

If [complement| supplement| inconsistent| redundant] (*variable1, variable2*) **and [complement| supplement| inconsistent| redundant]** (*variable3, variable4*) **then [ADD| REMOVE] [complement| supplement| inconsistent| redundant]** (*variable5, variable6*)

For example, a rule description, "If complement(X, Y) and complement(Y, Z) then ADD complement(X, Z)" (#3), specifies that the complement relation is transitive. Another rule description, "If complement(X, Y) and supplement(X, Y) then REMOVE supplement(X, Y)" (#4), means that relation of supplement between two functional requirements which are having relation of complement will be deleted. With this type of rules we can detect lack of relations and remove wrong relations in an ontology.

We can use wildcard symbols in rules. For example, a rule description, "Student should not be who information of * scores" (#5), specifies the wrong agent of the group of functions accessing scores. Another rule description, "There exists a relation of complement between register * and cancel *" (#6), specifies a complementary relation between the two groups of functions: registration of something and cancellation of something. However, rule (#6) might request complementary relation of unrelated functions, e.g., "register account" and "cancel order". To fix this problem, we can use variables in rules (#5) and (#6). For example, rule (#6) can be rewritten such as "There exists a relation of complement between register X and cancel X" (#7).

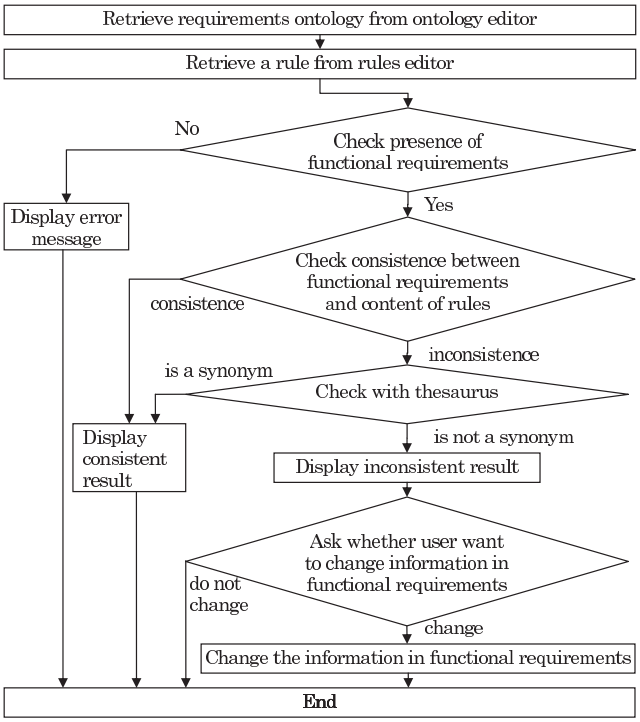

Fig. 5 Flow of checking first type of rules (Flows of checking second & third types of rules are similar).

This revised rule specifies complementary relation between functions "register account" and "cancel account", between functions "register order" and "cancel order", and so on.

When rules writers detect an error in a requirements ontology, they can define a new rule to detect similar errors. For example, when users can wrongly register books in a requirements ontology of library system, the rules writers can write a rule such as "Users should not be who information of register *." (It means that users should not register anything.) To support definition of rules, we provide examples of three types of rules, and these examples are easy to understand how to detect errors in an ontology, then rules writers can revise the examples to make their own rules. Of course, rules writers can also define new rules based on the rules grammars.

4.2 Verification of the Correctness of Requirements Ontology

We compare each rule with information in requirements ontology, and check the consistency between them. If consistent, we inform that the requirements ontology satisfies the rule. If inconsistent, we inform that the ontology does not satisfy the rule and counter example will be shown. The counter examples are useful to correct wrong parts of the ontology. Verification procedures are different for three types of rules.

The verification procedure for the first type of rules is described below, and is illustrated in Fig. 5.

1. Check whether functions specified in a rule exist in re-

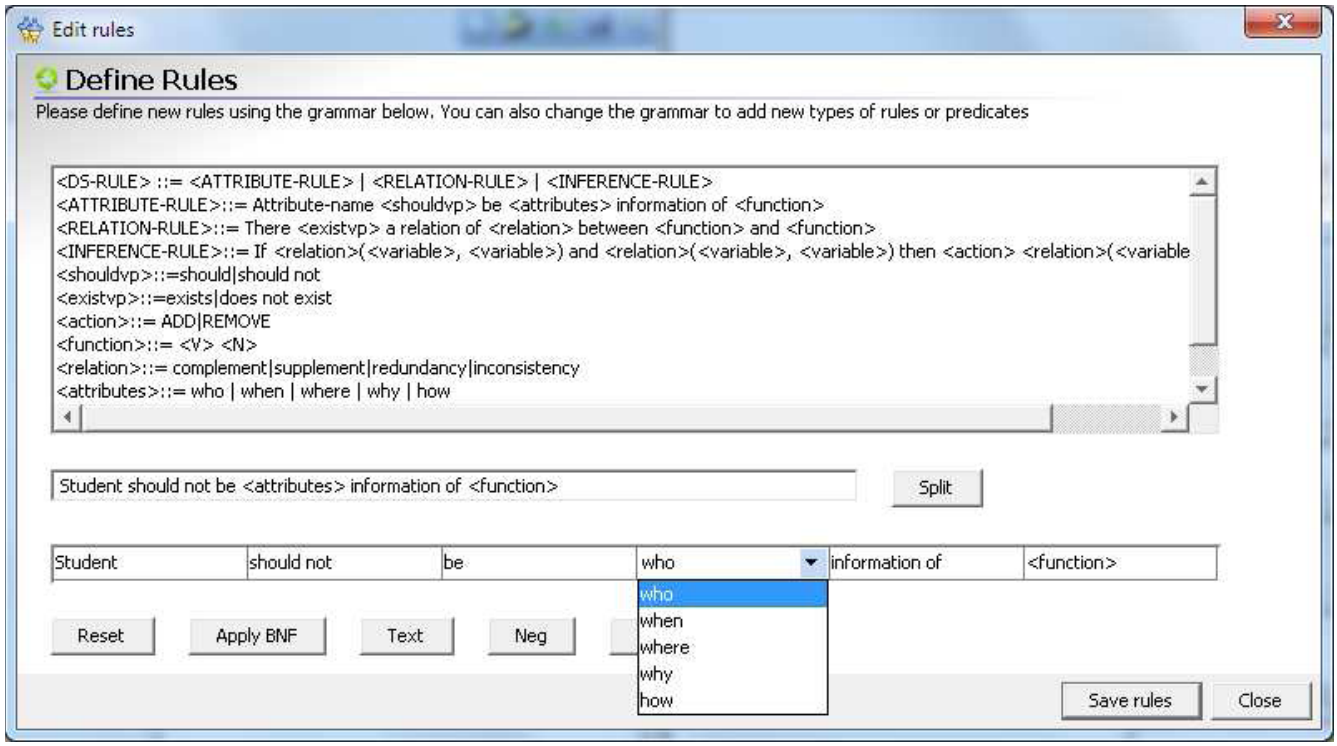

Fig. 6 Screenshot of rules definition tool.

quirements ontology. If not exist, error message will be given and verification will be ended unsuccessfully.
2. If exist, corresponding attribute of the functional requirements in the ontology will be derived and compared with specified attribute in the rule.
3. If same, verification will be ended successfully with a message: "successfully verified."
4. If different, transform the attribute in the rule using a thesaurus, then compare the transformed attribute with the corresponding attribute in the ontology.
5. If still different, error message and two different attributes will be given. If the rule is correct and the ontology seems wrong, the ontology will be automatically corrected after user's approval.

As for the second type of rules, the verification procedure is as follows.

1. Check whether two functions specified in a rule exist in requirements ontology. If not exist, error message will be given and verification will be ended unsuccessfully.
2. If the two functions exist, check whether relation between the two requirements in the ontology is same as the relation in the rule.
3. If same, verification will be ended successfully with a message: "successfully verified."
4. If different, error message and two different relations will be given. If the rule is correct and the ontology seems wrong, the ontology will be automatically corrected after user's approval.

The verification procedure the third type of rules is shown below.

1. Check whether functional requirements satisfied the two relations specified in the conditional part of a rule exist in requirements ontology. If not exist, error message will be given and verification will be ended unsuccessfully.
2. If exist, then list all of the functional requirements satisfied with the relation in the part of the rule. If "ADD" is specified in the rule, relations that do not exist in the ontology will be automatically added after user's approval. If "REMOVE" is specified in the rule, relations that exist in the ontology will be automatically deleted after user's approval.

4.3 A Supporting Tool for Verification of Requirements Ontology

We have developed a supporting tool for verification of requirements ontology with the three types of rules above. Figure 8 shows configuration of the verification tool. The requirements ontology editor and the verification tool cooperate through a shared memory. The requirements ontology editor builds an ontology tree in memory and the verification tool can access that ontology tree to traverse through the tree; retrieve functions, attributes; and updates functions and attributes if users approve. The verification tool was written with Java on an Eclipse 3.5.2, and was one person-month product; the number of source code lines was 3,652. The verification tool provides six functions, namely

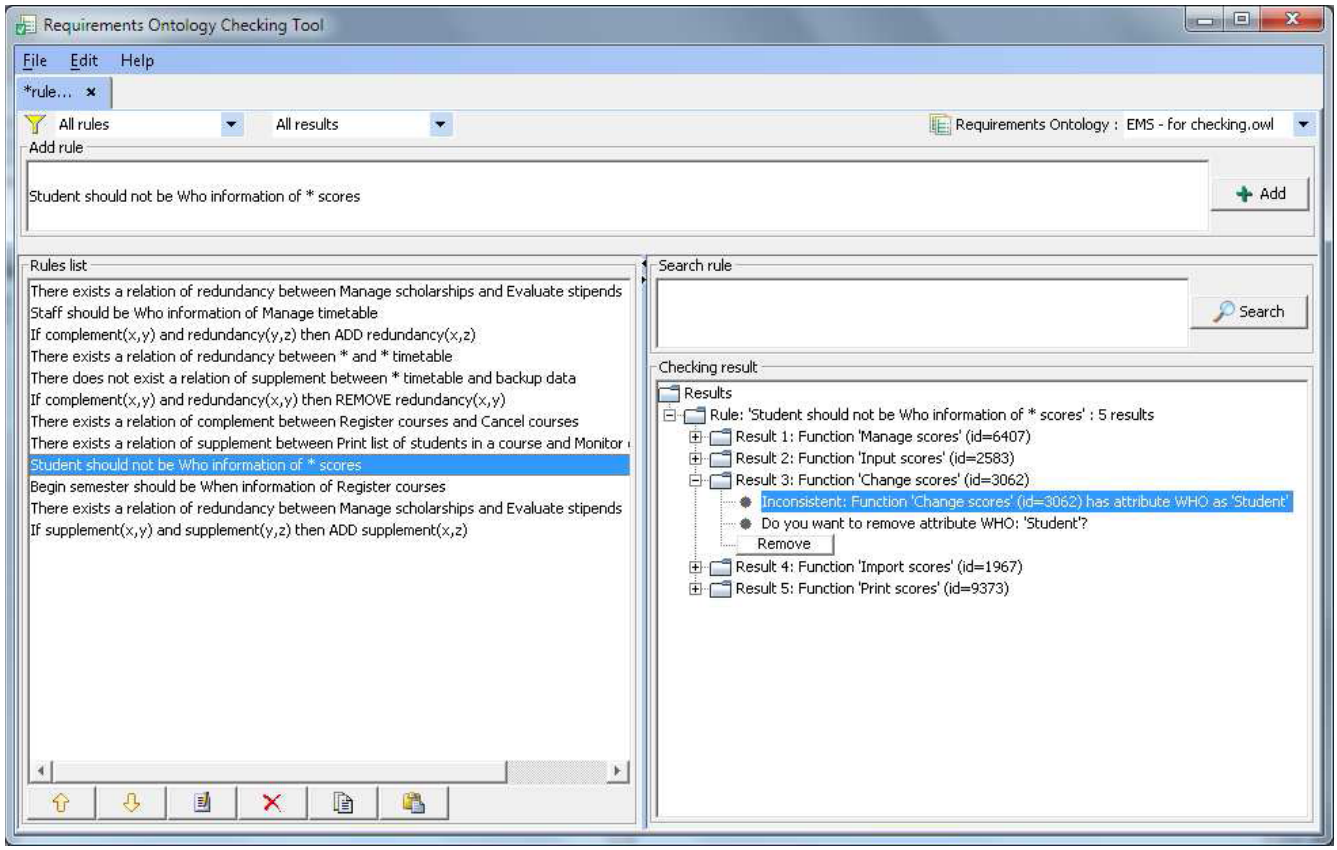

Fig. 7 Verification of "who information" of a functional requirement.

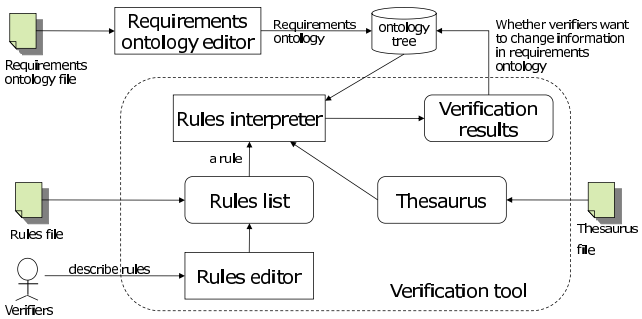

Fig. 8 Configuration of the verification tool.

to say, (1) input/saving of rule description files, (2) editing thesaurus, (3) editing rule descriptions and rule attributes, (4) analyzing rules, (5) retrieving rules, and (6) verifying requirements ontologies with rules. All implementations of the above functions are newly developed.

Figure 6 shows an image of rules definition interface. Rules can be described gradually by the rule grammar: each non-terminal symbol can be replaced by a selection from a list of expressions, according to the production rules defined in rule grammar. For example, in Fig. 6, in the rule which is defining, the symbol <attributes> is going to be replaced by the symbol "who". Users can also copy and revise existing rules to make new rules.

We will explain more about processing of rules by the rules interpreter (a module in Fig. 8). Firstly, a selected rule is classified into one of the three types of rules. Secondly, functions, attributes, and variables are extracted from the rule with regular expressions. Thirdly, the rules interpreter traverses through requirements ontology tree to compare functions, attributes in the tree with functions, attributes, and variables in the rule. The traversal and comparison are separated for each type of rules: for example, steps for the first type of rules are as in Fig. 5. The rules interpreter was written with Java, in 1,396 lines of code.

Figure 7 shows verification with a type 1 rule. Attributes of a functional requirement specified in a rule will be compared with attributes of the requirements in the applied ontology. In Fig. 7, the highlighted rule in the left box states that agent of the group of function of "* scores" should not be "student". In the right box, verifier shows a verification result that the ontology is inconsistent with the rule, because in the ontology, the function "Change scores" has a wrong Who attribute such as "student". The verification tool also suggests to delete that wrong information from ontology.

5. Evaluation

To evaluate the effect of using verification tool to support rule-based verification method of requirements ontol-

ogy (VRRO method), we conducted two comparative experiments. The subjects in the experiments were four students who studied information science. All subjects have been trained with errors types in requirements ontology and introduced with VRRO method. In each experiment, subjects were asked to check a same requirements ontology and submit results. Four subjects were divided into two groups, each group had two people; one group checked by tool (with tool) and the other group checked by hand (without tool). The group checking by hand was provided with ontology editor which allows to view functions structure, search for functions, and display attributes of functions. The group checking by tool was provided with ontology editor, and verification tool which allows to define checking rules and to verify automatically using rules. Both groups were provided with samples of checking rules for corresponding domains (pre-defined rules). The working method of two groups exchanged in the second experiment: the group using tool in the first experiment checked by hand in the second experiment and vice versa.

The major difference between using tool and not using tool is the capability to detect similar errors with a known error. All of the subjects know the correctness and the incorrectness of requirements ontology. For example, indispensable functional requirements should exist in ontology, wrong attributes of a functional requirement should not be specified, and wrong relationships should not be specified, and so on. Subjects with tool check requirements ontology and when they found an error, they can write a rule to detect similar errors. For example, if user is wrongly specified as an actor of a function of management of rental object, they specify a rule that actor of functions of managing something should be staff. Then the tool can detect similar errors if exist. By contrast, subjects without tool check ontology and when they found an error, they correct the error, but they cannot easily detect similar errors without the tool. This is the major difference between subjects with/without tool. In other words, subjects with tool can easily detect similar errors using the tool, but subjects without tool detect similar errors with effort.

We used two metrics to compare the results:

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

In the first experiment, two groups of subjects checked a requirements ontology for "hotel reception desk software". The requirements ontology included 80 function nodes; subjects were allowed 30 minutes to check and find errors in the requirements ontology. Table 2 summarizes the recall and precision metrics of the results by two groups. We denote subjects as A, B, C, and D. Table 2 shows that all of the recall metrics and precision metrics of the results by the group with tool are higher than those by the group without tool.

In the second experiment, subjects checked a requirements ontology for "asset management software". The requirements ontology included 51 function nodes; subjects were also allowed 30 minutes to check the requirements ontology. Table 3 summarizes the recall and precision metrics of the results by two groups. The table shows that all of the recall metrics and precision metrics of the results by the group with tool are higher than those by the group without tool, though two groups have exchanged working method.

To illustrate the recall and precision metrics more visually, we show these values in 100% scale in Fig. 9. The figure displays that in all cases in the two experiments, the precision metrics and recall metrics of the results by group with tool were higher than metrics of the results by group without tool. The averages of the recall metrics and precision metrics of the results by using tool, as shown in Table 4, were 41.46% and 80.16%, respectively, higher than those average metrics by not using tool which were 25.96% and 51.90%, respectively.

Table 2 Precision and recall metrics in the first experiment.

Group	Subject	Precision		Recall	
		Metrics	%	Metrics	%
with tool	A	11 / 15	73.33%	11 / 26	42.31%
	B	13 / 16	81.25%	13 / 26	50.00%
without tool	C	6 / 13	46.15%	6 / 26	23.08%
	D	8 / 20	40.00%	8 / 26	30.77%

Table 3 Precision and recall metrics in the second experiment.

Group	Subject	Precision		Recall	
		Metrics	%	Metrics	%
without tool	A	9 / 14	64.29%	9 / 34	26.47%
	B	8 / 14	57.14%	8 / 34	23.53%
with tool	C	11 / 14	78.57%	11 / 34	32.35%
	D	14 / 16	87.50%	14 / 34	41.18%

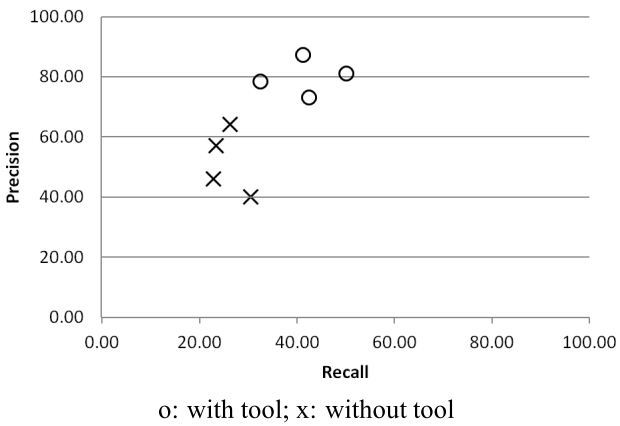

Fig. 9 Distribution of precision metrics and recall metrics in two experiments in 100% scale.

Table 4 Averages of metrics in two experiments (higher is better).

	with tool	without tool
Recall	41.46%	25.96%
Precision	80.16%	51.90%

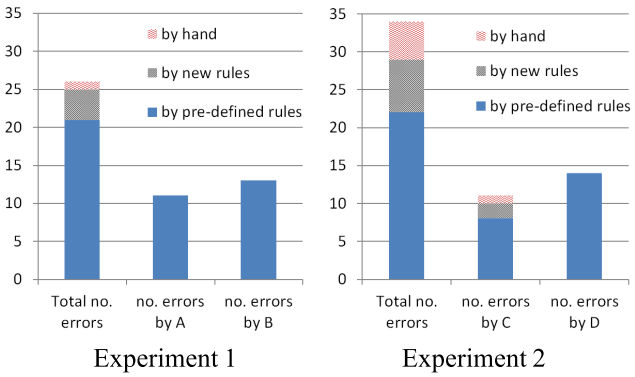

Fig. 10 No. errors found by subjects with tool.

The recall metrics of results by subjects using tool were not high (the average was 41.46%). To find the reason, we classify the correctly detected errors into three sources: (1) errors detected by pre-defined rules, (2) errors detected by definition of new rules, (3) errors detected by hand. Figure 10 shows the portions of sources of errors found by subjects with method. Majority of errors were detected by using pre-defined rules, and only minority of errors were detected by definition of new rules and by hand (subject C in the experiment 2). On the errors detected by pre-defined rules, the subjects with tool also did not utilize all the errors that the tool could detect. It was because of their unfamiliarity with VRRO method and tool, and because of limited time. The verification tool only suggested potential issues, and subjects evaluated whether these issues were errors. In the future, if subjects could master VRRO method and have more time to define new rules, the recall metrics would be higher.

There were errors detected by subjects with tool but not detected by subjects without tool. For example, in the experiment 1 about "hotel reception desk software", there was an error of lacking of supplement relation between two functions: "split group receipt" and "process guest check out." The group using tool detected this error by a rule of the third type: "If complement(x, y) and supplement(y, z) then ADD supplement(x, z)" (The reasoning was as follows: Because there existed a relation complement (split group receipt, merge group receipt) and a relation supplement (merge group receipt, process guest check out), so we should add a relation supplement (split group receipt, process guest check out)). However, not having support by reasoning engine of the verification tool, subjects without tool did not detect this error. Another example was that in the experiment 2 about "asset management software," subject C (using tool) defined a new rule of the first type: "System should be who information of * session parameters." Using this rule, subject C detected an error of wrong who attribute of function "set session parameters" in the requirements ontology, which was previously described as "staff." Nevertheless, subjects without tool did not detect this error.

On the other hand, there were errors which were not detected by tool, but detected by hand. For example, in the experiment 2, there existed an error of lacking of supplement relation between functions "view unused assets" and "view list of assets." This error was detected by the group without tool, but was not detected by the group with tool. Though this error can be detected with VRRO method by definition of a rule of the second type, subjects with tool did not find this error by mistake. In addition, detection of this error required knowledge of the domain "asset management software." One solution is to provide a customization mechanism or rules to automatically apply the customized rules to ontologies in future work.

Table 5 The no. of pre-defined rules and the no. of new rules defined by subjects with tool.

	Rules sources	no. of rules
Exp. 1	pre-defined rules	17
	new rules by A	0
	new rules by B	0
Exp. 2	pre-defined rules	18
	new rules by C	3
	new rules by D	0

Table 6 New rules defined by subject C.

R#	Rule	Rule type
#RC1	System should be who information of * session parameters	first type
#RC2	There exists a relation of supplement between Print X and Create X	second type
#RC3	There exists a relation of complement between Allocate * and Update *	second type

Table 5 summaries the number of pre-defined rules and new rules which were defined by subjects using tool. Mostly pre-defined rules were used, but three new rules were described by subject C in the experiment 2 (The three new rules are shown in Table 6). One reason that new rules were not defined so much was that subjects were allowed a short time in each experiment (30 minutes). All the three new rules by subject C were syntactically and semantically correct. In definition of new rules, subjects were guided with examples of rules, and they used these examples to construct new rules.

Using precision metrics and recall metrics separately is not always appropriate to show performance of an errors retrieval method. When recall tends to increases, precision would be decreased, and vice versa [20]. Harmonic mean is another metrics that combined both recall and precision [21]. Harmonic mean is computed as in Eq. (1). It aslo has value from 0 to 1; the bigger it is, the better the method is.

$$HarmonicMean = \frac{2 * Recall * Precision}{Recall + Precision} \quad (1)$$

Figure 11 displays the distribution of harmonic mean metrics in scale of 100%. It shows that in results of all subjects, the harmonic mean metrics by using tool were higher than the harmonic mean metrics by not using tool.

In summary, the recall, precision, and harmonic mean metrics of results by using tool were higher than those of re-

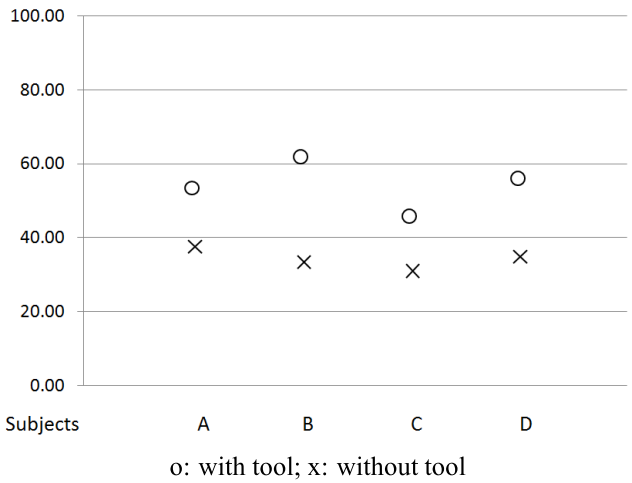

Fig. 11 Harmonic mean metrics in two experiments in 100% scale.

sults by not using tool. It suggests that using the verification tool with checking rules has positive effect in verification of requirements ontology, and helps improve the percentage and total of correctly detected errors in verification of requirements ontology. Based on errors found, requirements engineers will revise and improve the quality requirements ontology.

6. Conclusion

We proposed a rule-based verification method of the correctness of requirements ontology. We have developed a supporting tool for ontology verification on the basis of the method and illustrated the behaviors of the tool with examples. The experiments show that our method enables to detect incorrect components of requirements ontologies and improve the quality of requirements ontologies. Using a requirements ontology of good quality, we can elicit requirements with good quality [18], [19]. In the future, we plan to provide a customization mechanism of rules to apply the rules to ontologies. The rule-based verification method of requirements ontology has potential to support verification of other ontologies.

References

[1] K. Breitman and J. do Prado Leite, "Ontology as a requirements engineering product," Proc. 11th IEEE Int. Conf. on Requirements Engineering (RE'03), pp.309–319, Sept. 2003.
[2] W. Zhang, H. Mei, and H. Zhao, "A feature-oriented approach to modeling requirements dependencies," Proc. 13th IEEE Int. Conf. on Requirements Engineering (RE'05), pp.273–282, Sept. 2005.
[3] A. Bao, L. Yao, W. Zhang, and J. Yuan, "Approach to the formal representation of owl-s ontology maintenance requirements," Proc. 9th Int. Conf. on Web-Age Information Management (WAIM '08), pp.56–61, July 2008.
[4] H. Kaiya and M. Saeki, "Using domain ontology as domain knowledge for requirements elicitation," Proc. 14th IEEE Int. Requirements Engineering Conference, RE '06, pp.186–195, 2006.
[5] R. Kluge, T. Hering, R. Belter, and B. Franczyk, "An approach for matching functional business requirements to standard application software packages via ontology," Proc. 32nd IEEE Int. Conf. on Computer Software and Applications (COMPSAC '08), pp.1017–1022, Aug. 2008.
[6] L. Zong-yong, W. Zhi-xue, Y. Ying-ying, W. Yue, and L. Ying, "Towards a multiple ontology framework for requirements elicitation and reuse," Proc. 31st IEEE Int. Conf. on Computer Software and Applications (COMPSAC '07), pp.189–195, July 2007.
[7] G. Dobson, S. Hall, and G. Kotonya, "A domain-independent ontology for non-functional requirements," Proc. IEEE Int. Conf. on e-Business Engineering (ICEBE'07), pp.563–566, Oct. 2007.
[8] J. Xiang, L. Liu, W. Qiao, and J. Yang, "Srem: A service requirements elicitation mechanism based on ontology," Proc. 31st IEEE Int. Conf. on Computer Software and Applications (COMPSAC '07), pp.196–203, July 2007.
[9] L. Liu, Q. Liu, C. hung Chi, Z. Jin, and E. Yu, "Towards a service requirements ontology on knowledge and intention," Proc. 6th Int. Conf. on Quality Software (QSIC'06), pp.452–462, Oct. 2006.
[10] A. Kalyanpur, B. Parsia, E. Sirin, and J.A. Hendler, "Debugging unsatisfiable classes in owl ontologies," J. Web Semantics: Science, Services and Agents on the World Wide Web, vol.3, no.4, pp.268–293, 2005.
[11] H. Wang, M. Horridge, A.L. Rector, N. Drummond, and J. Seidenberg, "Debugging owl-dl ontologies: A heuristic approach," International Semantic Web Conference, pp.745–757, 2005.
[12] Protege: http://protege.stanford.edu/
[13] Pellet: http://clarkparsia.com/pellet/
[14] OWL Web Ontology Language Overview: http://www.w3.org/TR/-owl-features/
[15] SWRL: A Semantic Web Rule Language: http://www.w3.org/Subm-ission/SWRL/
[16] B.Q. Huy and A. Ohnishi, "A verification method of the correctness of requirements ontology," Proc. of the 10th Joint Conference on Knowledge-Based Software Engineering (JCKBSE 2012), pp.1–10, 2012.
[17] M. Satonaka, Y. Iyoda, and A. Ohnishi, "A supporting method of abstraction of software documents," IEICE Technical Report, SS2012-53, Jan. 2013 (in Japanese).
[18] D.V. Dzung and A. Ohnishi, "Ontology-based reasoning in requirements elicitation," Proc. 7th IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM'09), pp.263–272, Nov. 2009.
[19] D.V. Dzung and A. Ohnishi, "A verification method of elicited software requirements using requirements ontology," Proc. 19th Asia Pacific Software Engineering Conference (APSEC 2012), pp.553–558, Dec. 2012.
[20] R.A. Baeza-Yates and B. Ribeiro-Neto, "Retrieval evaluation," in Modern Information Retrieval, p.81, Addison-Wesley, 1999.
[21] W.S. Jr, R. Burgin and P. Howell, "Performance standards and evaluations in ir test collections: Cluster-based retrieval models," Information Processing and Management, vol.33, no.1, pp.1–14, 1997.

Dang Viet Dzung received B. of Engineering degree from Hanoi University of Science and Technology in 2003, and M. of Engineering degree from Ritsumeikan University in 2009. He was a Researcher at Vietnam National University from 2003 to 2007, and from 2009 to 2011. Currently he is a doctor course student at Ritsumeikan University. His current research interests include requirements engineering, and ontology-based verification. Dzung is a student member of IEEE.

Bui Quang Huy received B. of Engineering from Ritsumeikan University in 2011. Currently, he is one of co-founders of Rikkeisoft Co., Ltd, Hanoi, Vietnam.

Atsushi Ohnishi received B. of Engineering, M. of Engineering, and Dr. of Engineering degrees from Kyoto University in 1979, 1981, and 1988, respectively. He was a Research Associate of Kyoto University from 1983 to 1989 and an Associate Professor of Kyoto University from 1989 to 1994. Since 1994 he has been a Professor at Department of Computer Science, Ritsumeikan University. He was a visiting scientist at UC Santa Barbara, California, U.S.A. from 1990 to 1991 and also a visiting scientist at Georgia Institute of Technology, Georgia, U.S.A. in 2000. His current research interests include requirements engineering, object oriented analysis, and software design techniques. Dr. Ohnishi is a member of IEEE Computer Society, ACM, IEICE, Information Processing Society (IPS) Japan, and Japan Society for Software Science and Technology (JSSST).