A Verification Method of Elicited Software Requirements using Requirements Ontology

Dang Viet Dzung
Graduate School of Science and Engineering
Ritsumeikan University
1-1-1 Noji-Higashi Kusatsu, Shiga, 525-8577 Japan
Email: dungdv@selab.is.ritsumei.ac.jp

Atsushi Ohnishi
Department of Computer Science
Ritsumeikan University
1-1-1 Noji-Higashi Kusatsu, Shiga, 525-8577 Japan
Email: ohnishi@cs.ritsumei.ac.jp

***Abstract*—This paper briefly presents a checking method of elicited software requirements using requirements ontology, and introduces two extensions of the method. The first is capability of checking attributes of functional requirements (who, when, where, why, how), and the second is permission of requirements verifiers to specify new rules. The method is illustrated with examples, supported by a checking tool, and evaluated with experiments.**

***Keywords*-software requirements verification; requirements ontology; checking elicited requirements**

I. Introduction

An important task but difficult and lacking of tool in requirements engineering is requirements elicitation, in which the major result is a list of all features that the system will provide. The difficulty is due to complexity of natural language in human to human communication, and the tactic to extract knowledge from stakeholders. Knowledge-based approach is a solution to this obstacle, where understanding about certain problem domain is stored in advance, and the knowledge-base is used to guide the elicitation process.

To use domain knowledge in requirements elicitation, we need a knowledge structure to store them, and a method to guide the elicitation process using the knowledge. We have proposed requirements ontology models and a method of reasoning new requirements using requirements ontology[1]. The method works by extracting key words from elicited requirements and mapping them to requirements ontology. Then pre-defined rules are used for checking to find errors and revise existing requirements or add potential requirements. The method can check the lacking or redundance of functions of developing software, but it still cannot check the detail descriptions of functions (agent, place, time, reason, method), and the limited number of pre-defined rules also restricts types of errors that the method can detect. This paper will extend the checking method with two topics: checking attributes of functional requirements, and capability for requirements verifiers to specify new rules.

The paper is organized as follows. The next section will shortly introduce the requirements checking method using ontology. After that, Sect. III shows how to check attributes of functional requirements, and Sect. IV introduces how to allow requirements verifiers to specify new rules. Sect. V presents two experiments to evaluate the checking method. Sect. VI discusses related works, and finally, the paper arrives a conclusion in Sect. VII.

II. Checking Elicited Requirements using Requirements Ontology

A. Requirements Elicitation using Ontology

In requirements elicitation, stakeholders focus on functions or features of the developing software system. A domain knowledge about functions of applications in the corresponding domain can help stakeholders during elicitation activities. An ontology—which normally consists of objects, attributes, and relations—can be used to store the domain knowledge of software requirements. As we want to represent functional requirements in ontology, the objects become functions of software systems (expressed by a verb and several nouns); attributes are the scenarios of using the functions: agent (who), location (where), reason (why), time (when), method (how); relations among functions are: complement, supplement, aggregate, inherit, contradict, redundant[1]. The meta-model of functional requirements ontology is illustrated in Fig. 1.

Fig. 2 shows an example of functional requirements ontology for an education management system (EMS). The ontology is a hierarchy of functions in the application domain, and each function has attributes (who, when, where, why, how) and relations to others functions. For example, the EMS system has functions such as: "manage students, register courses, organize exams, manage scores," and so on. The function "manage scores" have several sub-functions: "input

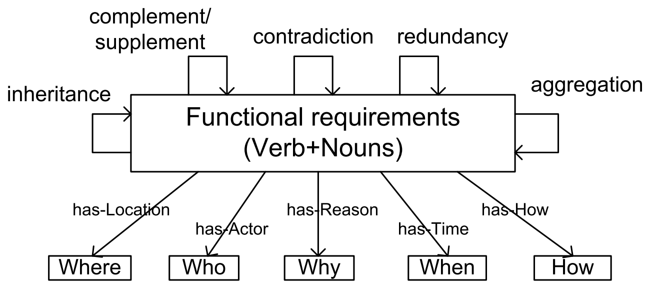


Fig. 1. Meta-model of functional requirements ontology[1].

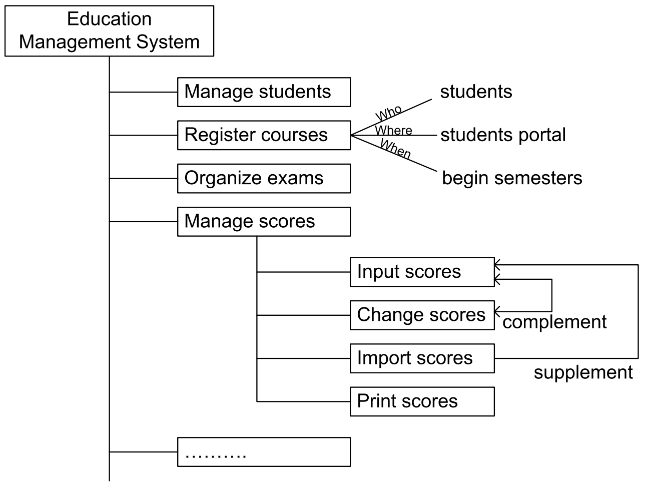


Fig. 2. A part of the requirements ontology for an education management system.

(1)	complement(X,Y) $\wedge$ inSRS(X) $\rightarrow$ shouldInSRS(Y)
(2)	supplement(X,Y) $\wedge$ inSRS(Y) $\rightarrow$ mayInSRS(X)
(3)	aggregate(X,Y) $\wedge$ inSRS(Y) $\rightarrow$ mayInSRS(X)
(4)	inherit(X,Y) $\wedge$ inSRS(X) $\rightarrow$ shouldInSRS(Y)
(5)	contradict(X,Y) $\rightarrow \neg$inSRS(X) $\vee \neg$inSRS(Y)
(6)	redundant(X,Y) $\rightarrow \neg$inSRS(X) $\vee \neg$inSRS(Y)

Fig. 3. General rules for checking elicited requirements.

inSRS(X): function X is already in requirements specification.
shouldInSRS(X): function X is mandatory.
mayInSRS(X): function X is optional.
relation(X,Y): there is a relation between X and Y.

Fig. 4. Predicates used in rules.

scores, change scores, import scores, print scores", in which the function "change scores" is complement to the function "input scores", and the function "import scores" is supplement to the function "input scores". The Fig. also shows examples of attributes (who, when, where) for the function "register courses".

How to use the requirements ontology to help elicitation? Elicited requirements are compared with ontology to reason new requirements and to revise requirements having vague or lacking information. This method is called requirements elicitation using ontology (REO). However, the method of reason new requirements is unclear, and mostly based on efforts and tactics of analysts. Ontology-based checking method is a solution to the question: How to reason new requirements using ontology?

B. Ontology-based Checking Method in Requirements Elicitation

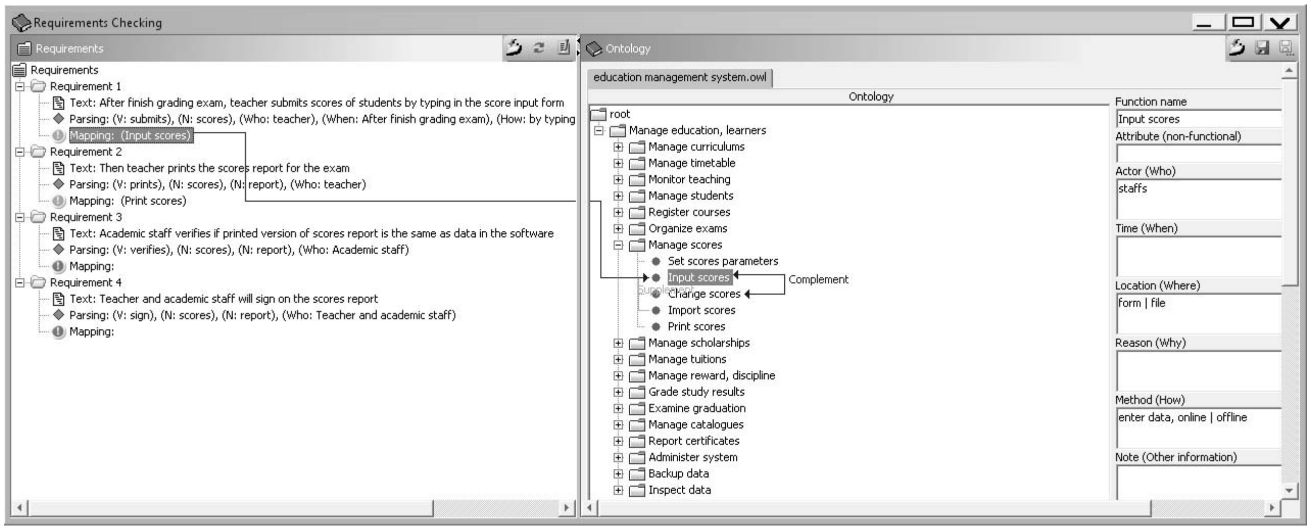
The checking method is described as follows. Elicited requirements are described as a list of sentences. Each sentence is parsed to extract verbs and nouns, and then mapped to nodes in requirements ontology. After that, the ontology and rules are used for checking to find errors in elicited requirements. Based on checking results, elicited requirements are revised to improve quality and increase quantity. The ontology-based checking method of software requirements has four steps:

1) Parse elicited requirements to extract key words (verb, nouns);
2) Map requirements to ontology using key words;
3) Check requirements using ontology and rules;
4) Revise requirements based on checking results.

The checking method uses pre-defined rules (or general rules) to detect errors in elicited requirements. Six general rules are listed in Fig. 3; and the predicates used in rules are explained in Fig. 4. For example, the complement rule (1) states that two functions having relationship of complementary should exist together in software requirements specification (SRS), while the supplement rule (2) specifies that the supplement function is optionally added to SRS. If an education management software has the function "input scores," it should have the complement function "change scores," but it may have the supplement function "import scores."

We have developed a tool for checking elicited requirements. The REO checking tool support four steps in requirements checking method: parsing requirements, mapping requirements to ontology, checking requirements using rules, and generating questions for revising requirements. The tool uses OpenNLP (Open source Natural Language Processing) library[2] to parse elicited requirements. It uses a thesaurus to help mapping requirements to ontology[1]. The tool allows management of requirements ontology and stores them in OWL files, but the reasoning power of OWL language is limited, so initial checking conditions such as mappings, functions relationships, and pre-defined rules are transformed to Prolog—a general purpose logic languages—for checking execution. The checking results will be displayed and the tool will generate questions for stakeholders by using questions templates.

Fig. 5 shows a screenshot of the REO checking tool. In the figure, elicited requirements for a scores input module for teachers are parsed and mapped to requirements ontology of education domain. For instance, the requirements 1 "After grading exam, teacher submits scores of students by typing in the score input form" is extracted with verbs and nouns such as "(V: submits), (N: scores)," then it is mapped to the function node "input scores" in requirements ontology. Using pre-defined rules and relations in requirements ontology, we can find incomplete requirements, inconsistent requirements, and redundant requirements. In the example in Fig. 5, the function "change scores" is complement to the function "input scores," so the checking tool will detect the lacking of function "change scores." With these errors, analysts can suggest stakeholders to add, remove, or revise requirements.



Requirements Checking

Requirements

- Requirements
  - Requirement 1
    - Text: After finish grading exam, teacher submits scores of students by typing in the score input form
    - Parsing: (V: submits), (N: scores), (Who: teacher), (When: After finish grading exam), (How: by typing
    - Mapping: (Input scores)
  - Requirement 2
    - Text: Then teacher prints the scores report for the exam
    - Parsing: (V: prints), (N: scores), (N: report), (Who: teacher)
    - Mapping: (Print scores)
  - Requirement 3
    - Text: Academic staff verifies if printed version of scores report is the same as data in the software
    - Parsing: (V: verifies), (N: scores), (N: report), (Who: Academic staff)
    - Mapping:
  - Requirement 4
    - Text: Teacher and academic staff will sign on the scores report
    - Parsing: (V: sign), (N: scores), (N: report), (Who: Teacher and academic staff)
    - Mapping:

Ontology

education management system.owl

Ontology

- root
  - Manage education, learners
    - Manage curriculums
    - Manage timetable
    - Monitor teaching
    - Manage students
    - Register courses
    - Organize exams
    - Manage scores
      - Set scores parameters
      - Input scores
      - Change scores
      - Import scores
      - Print scores
    - Manage scholarships
    - Manage tuitions
    - Manage reward, discipline
    - Grade study results
    - Examine graduation
    - Manage catalogues
    - Report certificates
    - Administer system
    - Backup data
    - Inspect data

Function name: Input scores
Attribute (non-functional):
Actor (Who): staffs
Time (When):
Location (Where): form | file
Reason (Why):
Method (How): enter data, online | offline
Note (Other information):

Fig. 5. Screenshot of REO checking tool.

The checking method of elicited requirements can detect the functional requirements having problems: incomplete, inconsistent, or redundant, but how to check the correctness of 4W1H attributes (who, when, where, why, how) of functional requirements? The following section will examine the extension of our checking method with capability of checking 4W1H.

III. Checking 4W1H attributes

In order to check the correctness of 4W1H attributes, firstly we need to extract 4W1H phrases from requirements. Secondly, we compare 4W1H phrases in requirements with 4W1H phrases in ontology.

A. Extraction and classification of 4W1H attributes

With OpenNLP library[2], we can parse each requirements sentence to a parsing tree and extract subject, main verb, nouns, and preposition phrases of the main verb. The term <verb nouns> becomes function name to represent the requirements. The subject and preposition phrases would become 4W1H attributes of requirements, but how to classify them into five types of 4W1H attributes: who, when, where, why, how? For example, a requirements "After grading exams, teacher submits scores of students by typing in the score input form." is parsed as "V: submits, N: scores, 4W1H: after grading exams, 4W1H: teacher, 4W1H: by typing, 4W1H: in the score input form". We can classify preposition phrases by types of words in 4W1H phrases: agent or subject (who), location (where), time (when), verb phrase (why or how). However, classification of preposition phrases is a difficult problem [3], and we do not focus on natural language processing, so we let requirements verifiers to select the types of 4W1H manually. In the example above, after verifiers classify 4W1H phrases, the requirements sentence is represented by the structure: submit-score (who: teacher, when: after grading exams, where: score input form, how: typing). This structure will be compared with 4W1H attributes in ontology to detect errors.

$\mathrm{MO}(R \mapsto X) \wedge \mathrm{hasWho}(R,WR) \wedge \mathrm{hasWho}(X,WX) \rightarrow \mathrm{shouldSimilar}(WR,WX)$
$\mathrm{MO}(R \mapsto X) \wedge \neg\mathrm{hasWho}(R,WR) \wedge \mathrm{hasWho}(X,WX) \rightarrow \mathrm{lackWho}(R)$
$\mathrm{MO}(R \mapsto X) \wedge \mathrm{hasWhen}(R,WR) \wedge \mathrm{hasWhen}(X,WX) \rightarrow \mathrm{shouldSimilar}(WR,WX)$
$\mathrm{MO}(R \mapsto X) \wedge \neg\mathrm{hasWhen}(R,WR) \wedge \mathrm{hasWhen}(X,WX) \rightarrow \mathrm{lackWhen}(R)$
$\mathrm{MO}(R \mapsto X) \wedge \mathrm{hasWhere}(R,WR) \wedge \mathrm{hasWhere}(X,WX) \rightarrow \mathrm{shouldSimilar}(WR,WX)$
$\mathrm{MO}(R \mapsto X) \wedge \neg\mathrm{hasWhere}(R,WR) \wedge \mathrm{hasWhere}(X,WX) \rightarrow \mathrm{lackWhere}(R)$
$\mathrm{MO}(R \mapsto X) \wedge \mathrm{hasWhy}(R,WR) \wedge \mathrm{hasWhy}(X,WX) \rightarrow \mathrm{shouldSimilar}(WR,WX)$
$\mathrm{MO}(R \mapsto X) \wedge \neg\mathrm{hasWhy}(R,WR) \wedge \mathrm{hasWhy}(X,WX) \rightarrow \mathrm{lackWhy}(R)$
$\mathrm{MO}(R \mapsto X) \wedge \mathrm{hasHow}(R,HR) \wedge \mathrm{hasHow}(X,HX) \rightarrow \mathrm{shouldSimilar}(HR,HX)$
$\mathrm{MO}(R \mapsto X) \wedge \neg\mathrm{hasHow}(R,HR) \wedge \mathrm{hasHow}(X,HX) \rightarrow \mathrm{lackHow}(R)$

Fig. 6. Rules for checking 4W1H attributes.

MO(R $\mapsto$ X) : requirements R is mapped to ontology node X.
hasWho(X,WX) : function X has agents of the action (who) as WX.
hasWhere(), hasWhen(), hasWhy(), hasHow(): similar to hasWho().
shouldSimilar(WR,WX): the two terms WR and WX should be similarity.
lackWho(R): requirements R is lacking information about the agents.
lackWhen(), lackWhere(), lackWhy(), lackHow(): similar to lackWho().

Fig. 7. Predicates used in 4W1H rules.

B. Checking 4W1H attributes

For checking 4W1H, we use rules to compare 4W1H phrases in requirements with corresponding phrases in ontology nodes. Fig. 6 list rules for checking 4W1H phrases. The first rule states that if a requirement R is mapped to an ontology node X, the agents of the functions (who) of R and X should be similar. The second rule specifies conditions when a requirement is lacking descriptions of the agents, but the mapped ontology node has specified them. Other rules for checking 'when, where, why, how' are similar to the rules for checking 'who'.

For example, after parsing requirements as in Fig. 8, we use 4W1H rules for checking 4W1H attributes. The rules require that the 4W1H attributes in requirements and ontology should

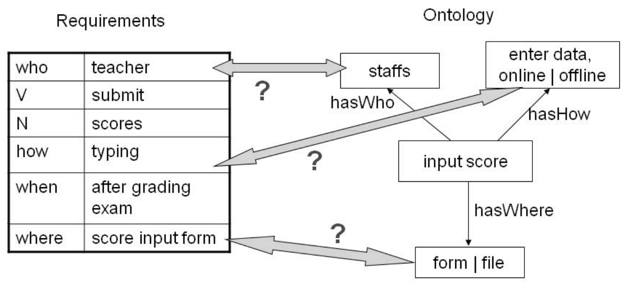


Fig. 8. Example of checking 4W1H attributes.

be similar. In order to check if two attributes are similar, the tool use a function to calculate the similarity between two strings with the support of a thesaurus[1]. For example, in Fig. 8, the attribute ‘how’ of the functional requirements is ‘typing,’ but the attribute ‘how’ of the ontology node is ‘enter data, online|offline,’ so the checking tool will show this checking result and generate question about the difference between two attributes. Analysts can use these suggestions to discuss with customers, e.g., whether the software should allow teachers to input scores offline.

IV. Rules Extension

To allow requirements verifiers to specify new rules for checking requirements, we use a context-free grammar and a rules definition tool.

A. Rules Description

Software requirements specification in a certain domain need to conform to regulations and rules which are set for that domain subjectively or objectively. These regulations are often called as domain rules, domain specific rules, or business rules. Can domain rules be expressed formally and help requirements elicitation? We propose a rule grammar to specify domain rules as follows (called grammar DREO).

DREO-RULE ::= <INTEGRITY-RULE>|<DERIVATION-RULE>
<INTEGRITY-RULE>::= <term> MUST <constraint>
<DERIVATION-RULE>::= IF {<predicate>} THEN <predicate>
<term>::= <V>|<N>|<V N>|<V N>.<attributes>| <variable>|<user-defined-term>
<constraint>::= <relation><term>
<relation> ::= not | equal | similar | in | out | greater | less
<predicate>::= <functor>({<term>})
<functor>::= <pre-defined-functor>|<user-defined-funtor>
<V>::= {list of verbs in problem domains}
<N>::= {list of nouns in problem domains}
<attributes>::= who|when|where|why|how
<variable>::= X|Y|WX|WY|<user-defined-variable>

With the grammar above, we can specify domain rules with capability of expressing objects in requirements ontology, and defining new predicates. We will show some examples to illustrate the DREO grammar.

For example, requirements verifiers can specify a domain rule which states that only administrators can manage users of a software, and manage implies add, edit, or delete:

(DR1) Vmanage ::= ‘add’ | ‘edit’ | ‘delete’
(Vmanage ‘users’).who MUST ‘administrator’

A rule that students cannot access scores can be written such as:

(DR2) Vaccess ::= ‘input’ | ‘change’ | ‘retrieve’
(Vaccess ‘score’).who MUST not(‘student’)

Verifiers can define a group of functions to insert item and a group of functions to delete item, and write a rule that the function insertion requires the function deletion as follows:

(DR3) Vins ::= ‘insert’ | ‘add’ | ‘create’
Vdel ::= ‘delete’ | ‘drop’ | ‘remove’
IF inSRS(Vins <N>) THEN shouldInSRS(Vdel <N>)

Domain specific rules and application specific rules need to be specific, because they are used to check specific requirements. On the contrary, general rules can be applied to all domains. Grammar DREO also has capability of expressing general rules. For example, verifiers want to assure that each requirement sentence should map to one node in ontology; if a sentence maps to two nodes, it may be ambiguous:

(DR4) IF MO(R $\mapsto$ X), MO(R $\mapsto$ Y) THEN mayAmbiguous(R)

B. A Tool for Defining Rules

A screenshot of the tool for requirements verifiers to specify rules is displayed in Fig. 9. Rules can be described gradually by the grammar DREO: each non-terminal symbol can be replaced by a selection from a list of expressions, according to the production rules defined in grammar DREO. For example, in Fig. 9, in the rule which is defining, the symbol <variable> is going to be replaced by the symbol <Y>.

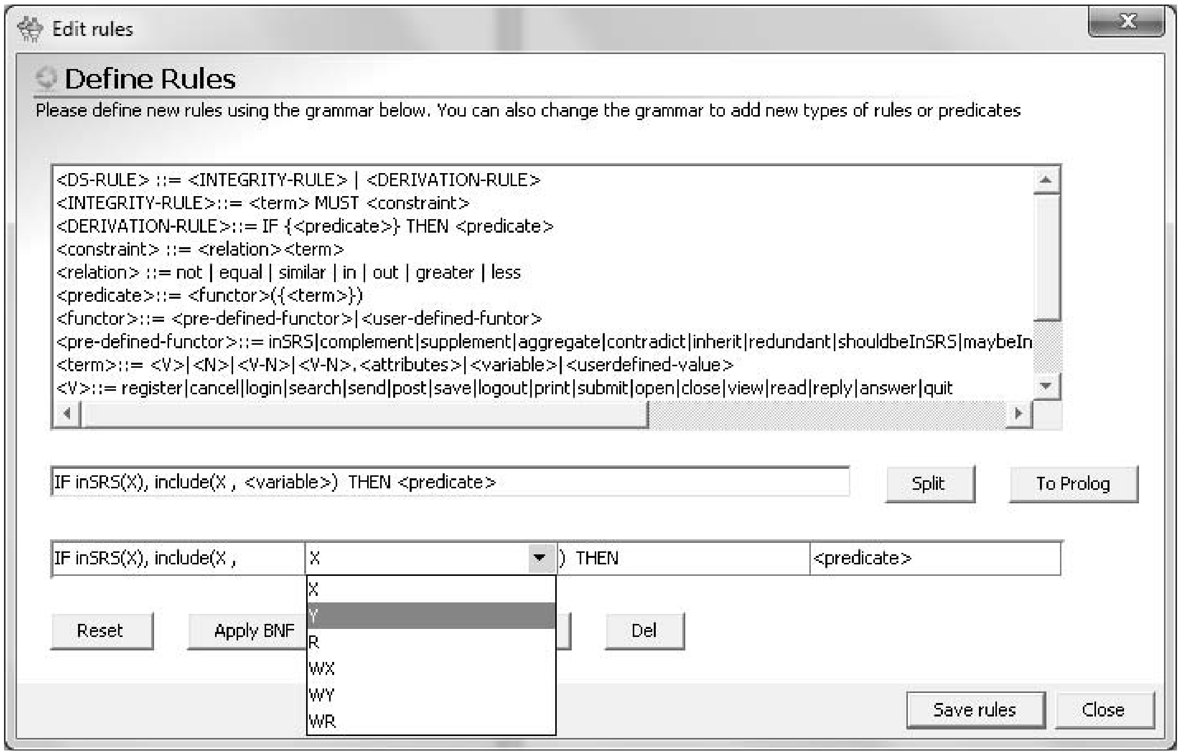
User-defined rules can be transformed from grammar DREO to Prolog in order to execute in the checking environment. Because we defined domain rules using a context-free grammar, it is easy to add new types of rules. For example, to describe expressions such as: possibility(may, can), cardinality, scope, coverage (all, every, only if), time sequence, we can add new rules templates having keywords such as MAY, CAN, ALL, EVERY to the grammar DREO.

C. Example

Suppose that an university want to develop a students portal site as a new module of the EMS system. Initial requirements are described as follows.

1) We want to create a new module as students portal.
2) On the portal, teachers can create accounts for new students.
3) Students can view their transcripts by their accounts.

Requirements verifiers find that the requirement 2) is inconsistent with domain rule (DR1), since rule (DR1) states that only administrators can create new accounts. They can also use



Please define new rules using the grammar below. You can also change the grammar to add new types of rules or predicates

```
<DS-RULE> ::= <INTEGRITY-RULE> | <DERIVATION-RULE>
<INTEGRITY-RULE>::= <term> MUST <constraint>
<DERIVATION-RULE>::= IF {<predicate>} THEN <predicate>
<constraint> ::= <relation><term>
<relation> ::= not | equal | similar | in | out | greater | less
<predicate>::= <functor>({<term>})
<functor>::= <pre-defined-functor>|<user-defined-funtor>
<pre-defined-functor>::= inSRS|complement|supplement|aggregate|contradict|inherit|redundant|shouldbeInSRS|maybeIn
<term>::= <V>|<N>|<V-N>|<V-N>.<attributes>|<variable>|<userdefined-value>
<V>::= register|cancel|login|search|send|post|save|logout|print|submit|open|close|view|read|reply|answer|quit
```

IF inSRS(X), include(X , <variable>) THEN <predicate>

IF inSRS(X), include(X ,	X	) THEN	<predicate>

X
Y
R
WX
WY
WR

Fig. 9. Screenshot of rules definition tool.

domain rule (DR2) to detect a problem with the requirement 3), because rule (DR2) specifies that students cannot access scores. In addition, by applying domain rule (DR3) to the requirement 2), since the software has the function “create accounts,” verifiers will find the need of the function “delete accounts.” Totally, using domain rules, verifiers can find three errors in the set of requirements above. These errors are potential problems that the checking method detects, but the revision of requirements depends on decision of stakeholders. For instance, after discussion with customers, the requirements team might keep the requirement 3) which allows students to access scores on portal site.

V. Evaluation

To evaluate the effect of ontology-based requirements verification method, we conducted two experiments. In each experiment, two objects checked a same SRS, but one object used requirements ontology and checking tool, and the other object worked freely. The working methods of two objects interchanged in the two experiments: the object who used checking method in the first experiment worked freely in the second experiment, and vice versa. Two objects in the experiments were software engineers; both had five years’ experience in software development.

In the first experiment, two objects were asked to check a SRS of 40 requirements of an accounting software for a high school. The first object worked freely and the second object was provided with ontology and tool. We built requirements ontology of accounting software from manuals of available commercial products, and provided the ontology to the second object. When receiving checking results from two objects, a requirements specialist reviewed their results to examine whether each reported problem is adequate to be considered an error. We use 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.

Table I summarizes the checking results from two objects. Each error type is listed in the table with the metrics of recall and precision. We see that all of the recall metrics of the results from the object using checking method are higher than those from the object not using the method. However, in the precision metrics, results from the object with method are only higher than results from the object without method in the incomplete errors. The precision metrics of off-topic errors in the results from the object with method is only 6/9. This is because the checking method use an assumption about off-topic errors: a requirement which does not map to any function in requirements ontology might be out of scope of the problem domain. The checking tool only detects potential problems which might be errors, and requirements verifiers have to determine if those problems are really errors.

In the second experiment, two objects checked a SRS of

TABLE I
NUMBER OF ERRORS FOUND BY TWO OBJECTS IN THE FIRST EXPERIMENT

	Recall		Precision	
	w/o. method	w. method	w/o. method	w. method
Incomplete	2/6	4/6	2/4	4/4
Inconsistent	0/1	1/1	0/0	1/1
Redundant	3/7	6/7	3/3	6/6
Ambiguous	2/7	5/7	2/2	5/5
Off-topic	4/7	6/7	4/6	6/9
4W1H errors	1/6	5/6	1/1	5/5

TABLE II
NUMBER OF ERRORS FOUND BY TWO OBJECTS IN THE SECOND EXPERIMENT

	Recall		Precision	
	w/o. method	w. method	w/o. method	w. method
Incomplete	3/9	7/9	3/4	7/7
Inconsistent	0/1	0/1	0/0	0/0
Redundant	2/3	2/3	2/3	2/4
Ambiguous	2/6	4/6	2/2	4/5
Off-topic	0/1	1/1	0/2	1/2
4W1H errors	2/10	8/10	2/2	8/8

33 requirements of a hotel management system. As saying above, the working methods of two objects interchanged in this experiment: the first object used ontology and checking tool while the second object did not. Table II summarizes the metrics of recall and precision of results from two objects. In major types of errors, the results from object with method have more correctly detected errors than the results from the object without method. However, in the precision metrics of redundant errors and ambiguous errors, the results from the object with method have lower values than those from the object without method. The reasons are that the method uses assumptions about redundant and ambiguous errors: two requirements mapping to a same function in ontology might have a similar meaning so they might be redundant; a requirement mapping to two different nodes in ontology might have two meanings so it might be ambiguous. The improvement of these assumptions is left as topics in future research.

In summary, two experiments' results suggest that our checking method has positive effect on detection of errors in elicited requirements or requirements specification.

VI. Related Works

There are several related works in requirements engineering using ontology. In [4], Kaiya and Saeki propose ontology-based requirements elicitation method; their method measures the quality of elicited requirements. In [5], Xiang et al. propose an automated service requirements elicitation mechanism (SREM) based on a goal-oriented requirements language. Zong-yong and colleagues [6] represented requirements model as UML diagrams and checked the model using domain ontology and rules, but their method does not support eliciting new requirements. The above three researches focus on requirements elicitation or improvement of the quality of requirements using ontology, but they do not focus on verification of software requirements with user defined rules.

Kluge et al. [7] described business requirements and software characteristics in terms of ontology, somewhat similar to our method of functional requirements representation by ontology. Their method helps business people to compare and match their functional requirements to functions of available commercial software products, and choose a suitable one. Nevertheless, our method is used by analysts during development of new products. Zhang, Mei, and Zhao [8] and Sun, Zhang, and Wang [9] provided feature diagrams to represent domain knowledge and proposed method to check feature models. Their frameworks are similar to our checking method, but their method uses feature modeling while our method uses requirements ontology.

VII. Conclusion

The extensions in this paper have improved the capability of the ontology-based checking method of software requirements. The two experiments have evaluated the effect our checking method including checking 4W1H, but they were still on a small scale and did not evaluate rules extension. Our method can be extended in some directions. Using the parsing technique as in our tool to parse software manual documents, we can obtain information to build ontology. The checking method can be more formally, such as translating SRS from natural language to controlled language and then to first-order logic, and applying model checking. How to use data ontology to support elicitation, or to support the requirements checking method? Can the ontology and rules be used to check diagrams in requirements specification (such as UML diagrams or data-flow diagrams)?

References

[1] D. V. Dzung and A. Ohnishi, "Ontology-based reasoning in requirements elicitation," in *Proc. 7th IEEE Int. Conf. on Software Engineering and Formal Methods (SEFM'09)*, Nov. 2009, pp. 263–272.

[2] Apache Software Foundation. Opennlp library. [Online]. Available: http://opennlp.sourceforge.net/

[3] D. Gildea and D. Jurafsky, "Automatic labeling of semantic roles," *Comput. Linguist.*, vol. 28, no. 3, pp. 245–288, Sep. 2002.

[4] H. Kaiya and M. Saeki, "Using domain ontology as domain knowledge for requirements elicitation," in *Proc. 14th IEEE Int. Requirements Engineering Conference (RE '06)*, 2006, pp. 186–195.

[5] J. Xiang, L. Liu, W. Qiao, and J. Yang, "Srem: A service requirements elicitation mechanism based on ontology," in *Proc. 31st IEEE Int. Conf. on Computer Software and Applications (COMPSAC '07)*, vol. 1, July 2007, pp. 196–203.

[6] L. Zong-Yong, W. Zhi-Xu, Z. Ai-Hui, and X. Yong, "The domain ontology and domain rules based requirements model checking," *International Journal of Software Engineering and Its Applications*, vol. 1, no. 1, pp. 89–100, 2007.

[7] 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. 32nd IEEE Int. Conf. on Computer Software and Applications (COMPSAC '08)*, Aug. 2008, pp. 1017–1022.

[8] W. Zhang, H. Zhao, and H. Mei, "A propositional logic-based method for verification of feature models," in *Formal Methods and Software Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin/ Heidelberg, 2004, vol. 3308, pp. 115–130.

[9] J. Sun, H. Zhang, and H. Wang, "Formal semantics and verification for feature modeling," in *Proc. 10th IEEE Int. Conf. on Engineering of Complex Computer Systems (ICECCS '05)*, 2005, pp. 303–312.