Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach

Haruhiko Kaiya
Faculty of Engineering,
Shinshu University
4-17-1 Wakasato,
Nagano, 380-8553, Japan
kaiya@cs.shinshu-u.ac.jp

Motoshi Saeki
Graduate School of Information Science and Engineering,
Tokyo Institute of Technology
2-12-1, Ookayama, Meguro-ku,
Tokyo, 152-8552, Japan
saeki@se.cs.titech.ac.jp

Abstract

We propose a software requirements analysis method based on domain ontology technique, where we can establish a mapping between a software requirements specification and the domain ontology that represents semantic components. Our ontology system consists of a thesaurus and inference rules and the thesaurus part comprises domain specific concepts and relationships suitable for semantic processing. It allows requirements engineers to analyze a requirements specification with respect to the semantics of the application domain. More concretely, we demonstrate following three kinds of semantic processing through a case study, (1) detecting incompleteness and inconsistency included in a requirements specification, (2) measuring the quality of a specification with respect to its meaning and (3) predicting requirements changes based on semantic analysis on a change history.

Keywords: Requirements Engineering,
Lightweight semantic processing, Thesaurus.

1. Introduction

One of the goals of requirements analysis is to develop a requirements specification document of high quality. There are several methods to achieve this goal and their supporting tools are going to be used in practice, e.g., goal oriented requirements analysis methods, scenario analysis, use case modeling techniques and so on. One of the most crucial problems to automate requirements analysis is that requirements documents are usually written in natural language, e.g. English or Japanese. Although techniques for natural language processing (NLP) are being advanced nowadays, it is hard to handle such requirements documents sufficiently by computer. However, semantic processing in requirements is indispensable for producing requirements specifications of high quality. To overcome the problem, there are several approaches, but each of them has its inherent problems. In some studies, a semi-formal notation for representing requirements, e.g. restricted natural languages was introduced, but it was difficult for human engineers to write syntactically and semantically correct requirements sufficiently by using this notation. Rigorous formal notations with axioms and inference system seem to be suitable, but its usage is very limited to practitioners because of their difficulty and complexity in the practitioners' learning and training.

We use an ontology system to develop a requirements document of high quality. Ontology technologies are frequently applied to many application domains nowadays. Because concepts, relationships and their categorizations in a real world can be represented in ontology, ontology can be used as resources of domain knowledge, especially in a specific application domain. By using such ontology, several kinds of semantic processing can be achieved in requirements analysis without rigorous NLP techniques.

In this paper, we propose a requirements analysis method by using an ontology technique, where we establish a mapping between a requirements specification and ontological elements. This technique allows us to have the possibility of automating semantic analysis with lightweight processing, not heavyweight NLP techniques. We call the proposed technique as *Lightweight Semantic Processing*. By mapping requirements descriptions in a requirements document onto ontological elements, which represents fragments of meaning in a problem domain, each description can be semantically interpreted. By applying inference rules to the ontological elements, we can achieve semantic processing about the requirements document.

The rest of this paper is organized as follows. In the next section, we explain how to interpret requirements documents by using ontology, and discuss what kinds of processing can be achieved in our ontology technique. In sections 3, 4 and 5, typical semantic processing techniques are introduced respectively by using a case study about "software music player" running on a personal computer (PC). Finally, we conclude our current results and discuss the future directions.

2. Ontology as a Semantic Model for Requirements Documents

2.1 Requirements for Ontology in Requirements Analysis

One of the most famous definitions of the term "ontology" is "formal explicit specification of shared conceptualization" [10]. Another famous definition is "classifications of the existing concepts". In our study, each requirements statement should be interpreted based on the knowledge of atomic constituents of meaning, and ontology is used as such knowledge. Thus an ontology, which is used in requirements analysis, has to have atomic concepts that are interpreted in the same way by any stakeholder in a specific application domain.

By using lexical decomposition technique [14], each requirements statement can be decomposed into several terms that are interpreted in the same way by anyone. Semantic structure among such terms, i.e., a thesaurus is required for our requirements analysis because such terms can be used as pointers to concepts. For example, we can use synonym relationship among terms when we unify different terms for the same concept. Generalization relationship among terms is also useful because we don't have to explain general characteristics of a specific term again. We need not only generic concepts and relationships such as synonym or generalization but also those appearing in software products and processes. For example, concepts such as "function" and "constraint" and relationships such as "apply (a function)" and "require (a constraint)" are required in our thesaurus.

We want to analyze a requirements document in the following ways, first detecting incompleteness and inconsistency, second measuring the quality of the document with respect to its meaning, and last predicting requirements changes. Then, we need a mechanism to process the thesaurus, a requirements document and their relationships. We will use logical inference mechanism because it seems to be suitable for three kinds of requirements analysis above and supporting tools such as prolog are available. To summarize, ontology used in this study consists of a thesaurus and a set of inference rules.

2.2 Ontology System for Requirements Documents

Figure 1 illustrates mappings from requirements items (statements) in a requirements document to elements in an ontology. The ontology is written in the form of class diagrams. The requirements document may be described in advance, or it may be described incrementally through the interaction between a requirements analyst and stakeholders. The requirements document is analyzed by using this kind of mappings. For example, we may suspect a requirements document is incomplete when not all elements in an appropriate ontology are related to items in the document, e.g.,

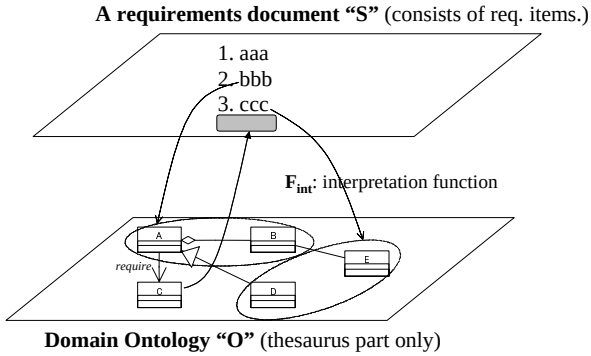

Figure 1. Mapping from Requirements to Ontology

concept C in Figure 1. How to make such mapping and to make or acquire an appropriate ontology is out of scope of this paper. We assume requirements analyst achieves such mapping. We formally define the notations and semantics of our ontology system below.

2.3 Thesaurus in an Ontology System

To process requirements semantically, we have to explicitly describe semantic information in a thesaurus. Thus we represent a thesaurus in a directed graph with typed nodes and arcs like UML class diagram. We use the class diagram notation because we want to simply represent a typed directed graph. We will use another powerful notation when our thesaurus becomes more complex. Nodes in the graph correspond to concepts in ontology, and arcs correspond to semantic relationships among concepts. We have decided types of concepts and relationships for requirements analysis as shown in Figure 2. To detect inconsistency among concepts, we explicitly introduce a relationship "contradict". We also introduce a relationship "require" to capture concepts missing in a requirements document. We introduce software requirements specific concepts and relationships. For example, "function" and "quality" are frequently mentioned in requirements documents, and a relationship "apply" is used to represent e.g., a function is applied to an object. According to the type of each concept or relationship, different inference rules are used, thus we can process a requirements document semantically without natural language processing techniques. To summarize, our ontology system can be represented in the following way.

$\mathrm{Ontology\_System} = (\mathrm{Con}, \mathrm{Rel}, \mathrm{Rules})$

Con: a set of concepts. $\mathrm{Con} = \cup_{\mathrm{t} \in \mathrm{Type}} \mathrm{Con_t}$

$\mathrm{Con_t}$: a set of concepts where its type is t, e.g., $\mathrm{Con_t}$="function"

Rel: a set of relationships. $\mathrm{Rel} = \cup_{\mathrm{u} \in \mathrm{Rel}} \mathrm{Rel_u}$

$\mathrm{Rel_u} = 2^{\mathrm{Con} \times \mathrm{Con}}$: a set of concept pairs that can be related by $\mathrm{Rel_u}$, e.g., by "contradict"

"Rules" represent a set of inference rules and we will explain them in detail below.

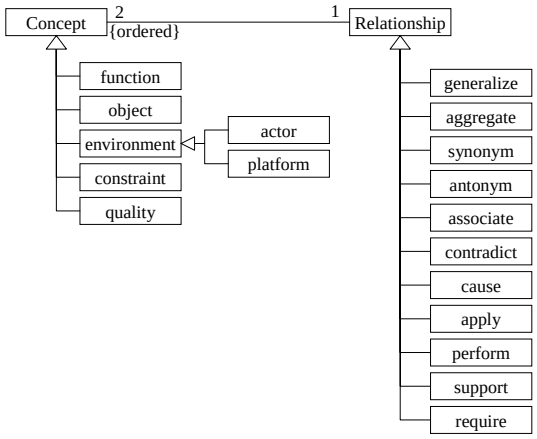

Figure 2. Concepts and Relationships in Our Thesaurus

For convenience, following predicates are used in the rest of this paper.

- P(x): the type of a concept or a relationship "x" is "P". e.g., function(play), object(music).
- R(x,y): the type of relationship between concept "x" and "y" is "R". e.g., aggregate(car, tire), generalize(operation, play), apply(delete, file), require(player, codec).
- InSpec(x,S): requirements document S refers to concept or relationship x. e.g., InSpec(A, S) and InSpec(require(A,C), S) are true but InSpec(C, S) is false in Figure 1.

2.4 Mapping and Inference Rules

Mapping from requirements to thesaurus in Figure 1 can be written formally bellow.

$$\mathrm{F_{int}} : \mathrm{ReqItem} \rightarrow 2^{\mathrm{Con} \cup \mathrm{Rel}}$$

"ReqItem" is a set of requirements items (statements) in a requirements document, and we call a function $\mathrm{F_{int}}$ as "an interpretation function". In the case of Figure 1, $\mathrm{F_{int}}(\mathrm{bbb}) = \{\mathrm{A}, \mathrm{B}, \mathrm{aggregate}(\mathrm{A}, \mathrm{B})\}$ and $\mathrm{F_{int}}(\mathrm{ccc}) = \{\mathrm{D}, \mathrm{E}\}$. A predicate InSpec can be written as follows by using $\mathrm{F_{int}}$.

$$\mathrm{InSpec}(\mathrm{x}, \mathrm{S}) \equiv \exists \mathrm{r} \in \mathrm{S} \cdot (\mathrm{x} \in \mathrm{F_{int}}(\mathrm{r}))$$

Note that r shows requirements items in a document S.

By using the interpretation function and inference rules, we can analyze a requirements document semantically. In this study, we use first order predicate logic for the inference mechanism. Logical formulas including inference rules basically come from two kinds of resources, one is a thesaurus in an ontology system and another is explicitly described rule.

Concepts in a thesaurus are used as constant values in rules, and relationships are used as predicates. For example in Figure 1, we can obtain the following four formulas from the thesaurus.

require(A, C). aggregate(A,B). generalize(A,D). associate(B,E).

According to the meaning of each relationship and concept, we can define inference rules in advance. For example, a relationship "generalize" is used to show that characteristics of a super class are inherited to its sub classes. This fact can be written in the following logical schema.

$$\mathrm{generalize}(\mathrm{x}, \mathrm{y}) \wedge \mathrm{P}(\mathrm{x}) \rightarrow \mathrm{P}(\mathrm{y}).$$

In the case of Figure 1, we can obtain the following rule

$$\mathrm{generalize}(\mathrm{A}, \mathrm{D}) \wedge \mathrm{require}(\mathrm{A}, \mathrm{C}) \rightarrow \mathrm{require}(\mathrm{D}, \mathrm{C}).$$

by applying assignments x=A, y=D and P(x)=require(x,C). By using this rule, we can systematically know that requirements items corresponding to a concept C seem to be missed in a requirements document in Figure 1.

Inherent characteristics of each relationship are also represented as inference rules. For example, as a relationship "generalize" is reflective and transitive, we use the following inference rules.

$$\forall \mathrm{x} \cdot \mathrm{generalize}(\mathrm{x}, \mathrm{x}).$$

$$\mathrm{generalize}(\mathrm{x}, \mathrm{y}) \wedge \mathrm{generalize}(\mathrm{y}, \mathrm{z}) \rightarrow \mathrm{generalize}(\mathrm{x}, \mathrm{z}).$$

On the other hand, as a relationship "antonym" is symmetrical, we use the following rule.

$$\mathrm{antonym}(\mathrm{x}, \mathrm{y}) \rightarrow \mathrm{antonym}(\mathrm{y}, \mathrm{x}).$$

2.5 An Example of an Ontology System

1. Play a music, pause. Go to next or previous music.
2. Forward and rewind.
3. Adjust volume and mute.
4. Repeat play list.
5. Random play list.

Figure 3. Requirements Document A (for music player)

Here we show a simple example to demonstrate the use of our ontology system. Figure 3 shows a requirements document that should be analyzed, and Figure 4 shows an ontology (thesaurus part only). Because the ontology in Figure 4 is generated based on the document in Figure 3, each requirements item is successfully mapped to the part of the ontology. Round rectangles written in dashed line in Figure 4 show each return value of the interpretation function $\mathrm{F_{int}}$. For example, $\mathrm{F_{int}}$(item 3 in Figure 3) corresponds to the following set.

{ volume op, volume, adjust, mute,
generalize(volume op, adjust),
generalize(volume op, mute),
apply(volume op, volume) }

Several ambiguous, incomplete or inadequate points can be found in the requirements document. For example, "play

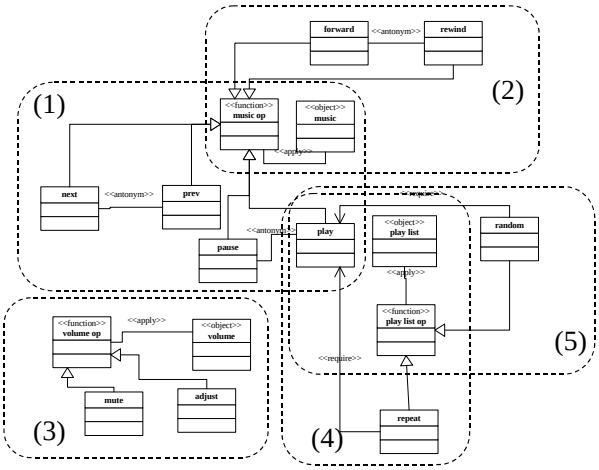

Figure 4. Ontology A (generated from a Document A)

list" in items 4 and 5 in Figure 3 is not clearly specified. The ontology system should provide a relationship such as aggregate(play list, music) to specify the term clearly. In addition, editing operations for play list such as create, delete or update should be also provided for the completeness of the requirements document. Section 3 provides additional means for detecting such issues.

3. Detecting Inconsistency and Incompleteness

We check whether a requirements document is consistent or complete by using an ontology system. As mentioned in last section, each requirements item (statement) is mapped onto a set of elements (concepts and relationships) in the thesaurus of the ontology system. To detect inconsistency of a requirements document, we try to find mutually contradicting elements where requirements items are mapped. For example, we decide the document is inconsistent if there is a relationship "contradict" between two concepts where the document is mapped. To detect incompleteness of a requirements document, we follow specific relationships from concepts where the document is already mapped. For example, we follow "require" relationship and find a concept that does not appear in the current document. Then, we add new requirements items (statements) corresponding to the concept.

These detections are systematically achieved by inference rules. By using a thesaurus in Figure 5 and inference rules below, we will improve a requirements document A in Figure 3 with respect to the consistency and completeness.

First, we suppose the following inference rule for improving the completeness of a requirements document.

$$\forall s \forall x \exists y \cdot ((\mathrm{object}(x) \wedge \mathrm{inSpec}(x, s)) \rightarrow (\mathrm{function}(y) \wedge \mathrm{apply}(y, x) \wedge \mathrm{inSpec}(y, s)))$$

The intuitive meaning of this rule is that "if an object is mentioned in a document, functions that can be applied to the object are also mentioned in the document". As an object(play list) is mentioned in the document A, the rule above can be applied for improving the completeness of the document A as follows.

$$\exists y \cdot ((\mathrm{object}(\text{play list}) \wedge \mathrm{inSpec}(\text{play list}, \text{document A})) \rightarrow (\mathrm{function}(y) \wedge \mathrm{apply}(y, \text{play list}) \wedge \mathrm{inSpec}(y, \text{document A})))$$

As a result, we can investigate the possibility about inSpec(create, document A), inSpec(delete, document A), inSpec(import, documnet A) and so on. Of course, the requirements analyst finally decide whether a function is added to the requirements or not. Suppose a function(import) is added and the following requirements item (statement) 6 is added to the document A. We call the extended document as the document B.

> 6. Import play list.

Second, we also suppose the following inference rule for improving completeness of a requirements document.

$$\forall s \forall x \forall y \cdot (\mathrm{InSpec}(x, s) \wedge \mathrm{require}(x, y) \rightarrow \mathrm{InSpec}(y, s))$$

The intuitive meaning of this rule is "if a concept is mentioned in a requirement document, concepts required by the concept should be also mentioned in the document". As function(import) is mentioned in the document B and the function requires another function(convert) as shown in Figure 5, the rule above can be applied to the document B as follows.

$$\mathrm{InSpec}(\mathrm{import}, \mathrm{documentB}) \wedge \mathrm{require}(\mathrm{import}, \mathrm{convert}) \rightarrow \mathrm{InSpec}(\mathrm{convert}, \mathrm{documentB})$$

As a result, a new requirements item (statement) 7 related to function(convert) should be added to the requirements document B in the following way.

> 7. Convert play list.

We call such extended document as requirements document C. A relationship "antonym" plays the same role as "require". For example, if function(create) is mentioned in a requirements document and antonym(create,delete) exists in an ontology, function(delete) should be also mentioned in the document.

Finally, we suppose the following formula for detecting inconsistency.

$$\forall s \forall x \cdot (\mathrm{InSpec}(x, s) \rightarrow \exists y \cdot (\mathrm{InSpec}(y, s) \wedge \mathrm{contradict}(x, y))$$

The intuitive meaning of this formula is "if a concept x is mentioned in a requirements document, another concept y is also mentioned in the document and x contradicts y". When this formula becomes true, we may assume the requirements document is inconsistent. Suppose the following two items 8 and 9 are added to the requirements docu-

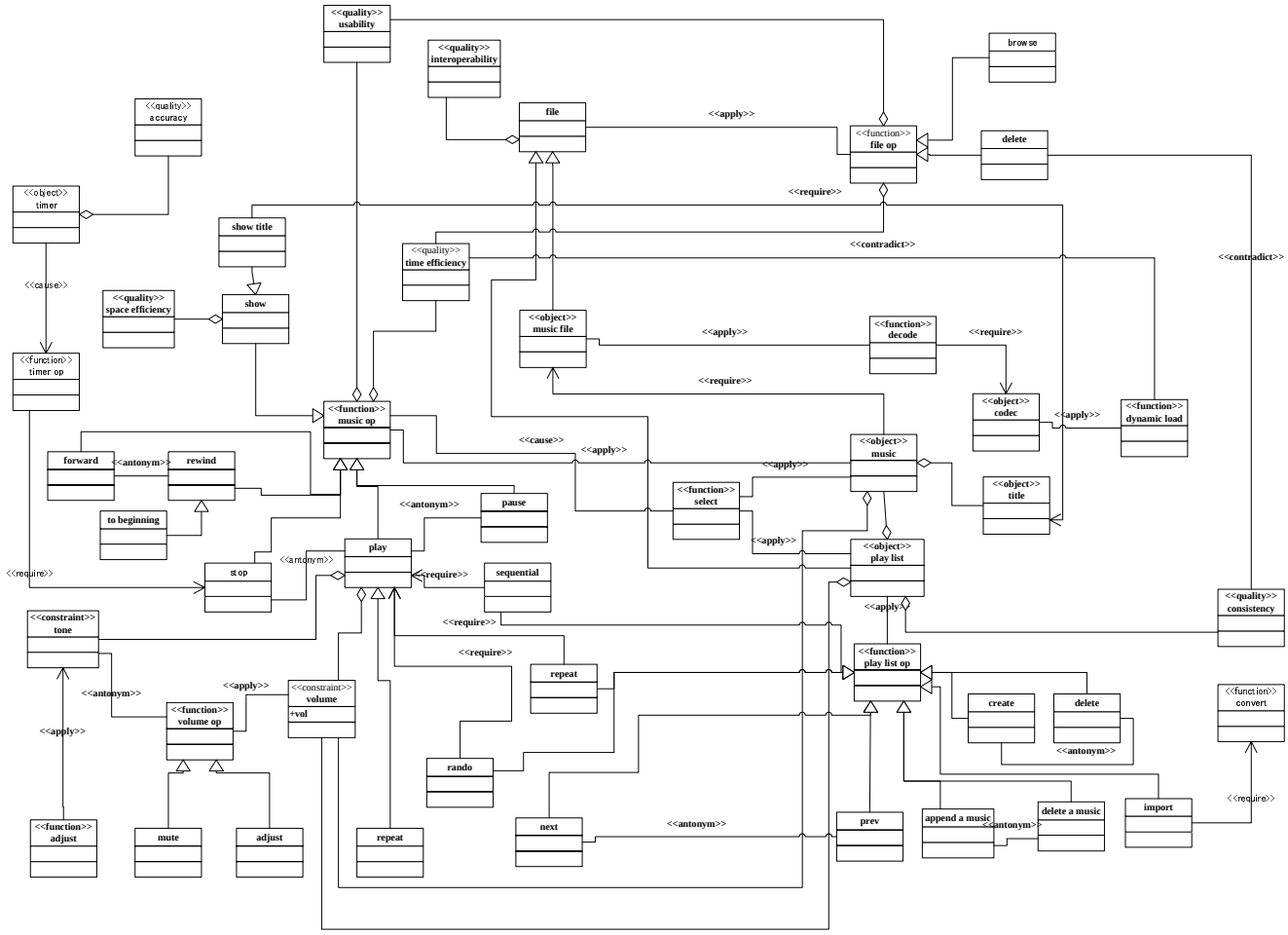

Figure 5. Ontology B (more complete version)

ment C, and we call the extended documents as document D.

> 8. Operations are quickly. (time efficiency)
> 9. Dynamic and on-demand loading of codec modules for unknown formats (dynamic load)

Two concepts quality(time efficiency) and function(dynamic load) are found in the ontology in Figure 5, and they correspond to the requirements items 8 and 9 respectively. In addition, the following relationship is also found in the ontology.

contradict(quality(time efficiency),
function(dynamic load))

The formula above is systematically interpreted as true when x=quality(time efficiency), s=document D and y=function(dynamic load). In other words, we can know function(dynamic load) is harmful with respect to the quality(time efficiency) by interpreting the formula above under the ontology in Figure 5. In this way, we can find inconsistency in a requirements document. How to resolve the inconsistency is out of scope of this paper, but it depends on the analysts and/or stakeholders in general.

We have introduced three inference rules or logical formula for detecting incompleteness and inconsistency about a requirements document. As we illustrated, mapping requirements items to the elements in an ontology seems to be useful and lightweight for semantic processing in requirements analysis.

4. Metrics for Requirements Documents

By using our ontology system, we can calculate metrics for the characteristics of a good software requirements specification. In IEEE 830 standard [1], there are eight characteristics and the following four characteristics are related to the semantics of a requirements specification. Note that “ReqItem” means the set of requirements items (statements) in a requirements document, “Con” means the set of concepts in an ontology, “Rel” means the set of relationships in the ontology, and “Clo” means a closure of $\mathrm{Con} \cup \mathrm{Rel} \backslash \{\mathrm{contradict}, \mathrm{antonym}\}$.

- Correctness = $\frac{|\{\mathrm{x} \mid \mathrm{x} \in \mathrm{ReqItem} \wedge \mathrm{F_{int}(x)} \neq \phi\}|}{|\mathrm{ReqItem}|}$

  We regard ontology as a semantic basis for a specific domain, thus all requirements items should correspond to elements in the ontology.
- Completeness = $\frac{\left|\begin{array}{l}\{\mathrm{x} \mid \mathrm{x} \in \mathrm{Con} \cup \mathrm{Rel} \wedge \\ \quad \exists \mathrm{y} : \mathrm{ReqItem} \cdot \mathrm{x} \in \mathrm{F_{int}(y)}\}\end{array}\right|}{|Con \cup Rel|}$

  Ideally, all elements in ontology should be mentioned in the requirements document for its completeness.
- Consistency = $\frac{|\{\mathrm{x} \mid \mathrm{x} \in \mathrm{RCC} \wedge \neg \mathrm{contradict(x)}\}|}{|\mathrm{RCC}|}$

  Relationships between concepts mentioned in the requirements document are focused in this metric. Such relationships can be represented in the following way.

  $$\mathrm{RCC} = \begin{array}{l}\{\mathrm{r} \mid \exists \mathrm{a} \exists \mathrm{b} : \mathrm{Con} \cdot \exists \mathrm{x} \exists \mathrm{y} : \mathrm{ReqItem} \cdot \\ \quad a \in F_{int}(x) \wedge b \in F_{int}(y) \wedge r(a,b)\}\end{array}$$

  If there are "contradict" relationships in such relationships, we regard the document is inconsistent.
- Unambiguity = $\frac{|\{\mathrm{x} \mid \mathrm{x} \in \mathrm{ReqItem} \wedge \mathrm{F_{int}(x)} \subseteq \mathrm{Clo}\}|}{|\mathrm{ReqItem}|}$

  When a requirements item is mapped onto several elements that are not semantically related, the item is regarded as ambiguous one. The set Clo is the maximal set of elements that are semantically related, thus $\mathrm{F_{int}(an\ item)}$ should be a subset of Clo for unambiguity.

According to the definitions above, we calculated metrics for a requirements document in Figure 6 against the ontology in Figure 5. Note that $|\mathrm{Con}| = 48$, $|\mathrm{Rel}| = 67$ and $|\mathrm{ReqItem}| = 15$.

- Correctness = $13/15 = 87\%$
  ReqItem 8 and 9 cannot be mapped onto the ontology.
- Completeness = $51/(48 + 67) = 44\%$
- Consistency = $35/36 = 97\%$
  A concept for ReqItem 4 and another concept for ReqItem 14 have a contradict relationship in the ontology.
- Unambiguity = $(15 - 2)/15 = 87\%$
  ReqItem 11 and 13 are ambiguous.

From this small experiment, the metrics seems to be valid, but there are some problems. Our ontology inherently contains contradict relationships. Therefore a requirements document is inconsistent if its completeness metric is 100%. Because this fact seems to be a little bit strange against our intuition, we have to reconsider the definition of completeness metrics. One of the ideas is to use a transitive closure of relations except "contradict" relation as the denominator of the definition. Other problem is about consistency about play list. If ReqItem 15 is accepted, a music file that is listed in a play list can be freely deleted by other applications, e.g., by file browser and the play list cannot be handled consistently by the music play. Current definition of consistency cannot take such case into account. Another problem is about priority about functionalities. Both in ReqItem 7 and 12, the rate of volume can be set in different ways. Because priority about these settings is not mentioned in this requirements document, the document is intuitively ambiguous. However, current definition of unambiguous cannot take such case into account. All these cases are against our intuition, thus we will improve the definitions of the metrics.

1. Play a music.
2. Pause a music.
3. Go to next or previous music.
4. Operations are quickly.
5. rewind and forward a music.
6. Adjust volume and mute.
7. Set volume for each music.
8. Play a music in any speed.
9. Play a music in any tone.
10. Random play list.
11. Delete play list.
12. Set volume for each play list.
13. Repeat play.
14. Dynamic and on-demand loading of codec modules for unknown formats.
15. Music files can be operated by other applications.

Figure 6. Requirements document X

5. Predict Requirements Changes

5.1 Change Patterns on the Ontology

If requirements changes in the next version can be predicted in advance, there are following kinds of advantages.

- We can decide which part in a requirements document is stable and which part is unstable. Note that stable parts are rarely changed in the future, but unstable parts will be frequently changed.
- Predicted changes enable us to improve the completeness of requirements specification incrementally.

We try to collect patterns of requirements changes frequently occur to predict requirements changes in the future. We assume effective and useful patterns can be characterized not only by syntactic features but also by semantic features of requirements. We examine this assumption by exploring change history of music player software.

We found the following two patterns of requirements changes in the history.

1. Append new functionality: Lacking functionality is found and it is appended.
2. Improve the quality of existing functionality: Insufficient quality, e.g., performance, usability and/or interoperability, of a function are found and the quality

is improved. This kind of insufficiency is usually detected after the system worked.

In both patterns, our ontology system contributes to predict the changes. The relationship "require" suggests us functions to be added in the former pattern in the same way in Section 3. The relationship aggregate(function(x), quality(y)) suggests us qualities to be improved in the latter pattern.

We represent the patterns above in a logical formula below so that requirements analyst can predict expected changes by applying the formula to the current version of software. If the formula is not satisfied in a version, the version should be changed so that the formula becomes true.

$$(\neg \mathrm{InSpec}(\mathrm{x}, \mathrm{S_i}) \wedge \mathrm{InSpec}(\mathrm{x}, \mathrm{S_{i+1}})) \rightarrow \exists \mathrm{j} \exists \mathrm{y} \exists \mathrm{r_{i+1}} \exists \mathrm{r_j} \exists \mathrm{S_j} \cdot ((\mathrm{j} > \mathrm{i}+1) \wedge ((\mathrm{InSpec}(\mathrm{y}, \mathrm{S_j}) \wedge \mathrm{quality}(\mathrm{y}) \wedge \mathrm{Rel}(\mathrm{x}, \mathrm{y}) \wedge \mathrm{r_j} \in \mathrm{S_j} \wedge \mathrm{r_{i+1}} \in \mathrm{S_{i+1}} \wedge ((\mathrm{y} \in \mathrm{F}(\mathrm{r_j}) \wedge ((\mathrm{y} \notin \mathrm{F}(\mathrm{r_{i+1}})) \vee ((\mathrm{y} \in \mathrm{F}(\mathrm{r_{i+1}}) \wedge \mathrm{r_{i+1}} \neq \mathrm{r_j})))$$

Note that $\mathrm{S_i}$ means a requirements document S version i, and the more the version number is, the newer the document is. ReqItem r_{i+1} and r_j are related to quality(y) in S_{i+1} and S_j respectively. Rel(x,y) shows whether function(x) is related to quality(y). The quality(y) could be a part of x or a part of an object applied by x. We can represent this predicate as follows.

$$\mathrm{Rel}(\mathrm{x}, \mathrm{y}) = \exists \mathrm{a} \cdot (\mathrm{generalize}(\mathrm{a}, \mathrm{x}) \wedge \mathrm{aggregate}(\mathrm{a}, \mathrm{y})) \vee \exists \mathrm{b} \exists \mathrm{c} \cdot (\mathrm{generalize}(\mathrm{b}, \mathrm{x}) \wedge \mathrm{apply}(\mathrm{c}, \mathrm{b}) \wedge \mathrm{aggregate}(\mathrm{c}, \mathrm{y}))$$

The intuitive meaning of the formula is as follows. When a new function x is added to a requirement document version i+1, requirements items (statements) about quality(y) that has a relationship (Rel) with x in the later version j of the document represent the one of the following characteristics.

1. $\mathrm{r_j}$ is added. ($\mathrm{y} \in \mathrm{F}(\mathrm{r_j}) \wedge \mathrm{y} \notin \mathrm{F}(\mathrm{r_{i+1}}) \cdots$)
2. items are updated from $\mathrm{r_{i+1}}$ to $\mathrm{r_j}$. ($((\mathrm{y} \in \mathrm{F}(\mathrm{r_j}) \wedge (\ldots \vee ((\mathrm{y} \in \mathrm{F}(\mathrm{r_{i+1}}) \wedge \mathrm{r_{i+1}} \neq \mathrm{r_j})))$)

5.2 Evaluation

To evaluate the patterns above, we analyzed a change history of software music player by using the ontology in Figure 5. The history is written in natural language (Japanese), 35 versions, including minor updates, of the software exist in the history. The version number of oldest software is 1.00, and the lasted version number is 2.02. Except the records of bug and defects fixes, 50 changes are found in the history. We categorize the changes as shown in Table. Note that one change is sometimes categorized into several types, thus the total number of categorized changes is more than 50.

An example of changes categorized into "Interoperability" is "increasing file formats that can be handled". An example of "Time Efficiency" is "improving response time of play operations". Examples of "Usability" are "introducing short-cut operations" and "improving the behavior of pop-up menus".

Table 1. Types of Changes and Their Frequencies

Type name of changes	#
Add Function	12
Improve Reliability (Error handling)	3
Improve Time Efficiency	3
Improve Interoperability	6
Improve Usability	30
Improve Accuracy	1

As shown in Table 1, 12 changes about adding functions are found in the history. According to the history, quality features were added to functions occurred after two out of the 12 functions' changes occurred. Quality features of functions were improved after other 8 out of 12 changes occurred. In the rest changes (two changes), new functionalities were simply added. From this fact, changes related quality features seems to occur frequently after new functions are added. This supports the patterns in section 5.1, thus we may predict requirements changes by using such patterns. We will refine such patterns to fit more specific situations specified by each domain ontology.

6. Conclusions and Future Works

In this paper, we propose a requirements analysis method by using domain ontology. Even though the method does not support rigorous natural language processing techniques (NLP), the method enables us to detect incompleteness and inconsistency about a requirements document, to measure the quality of the document, and to predict requirements changes in the future versions of the document. We examine the method about software music player.

We did not have/use specific supporting tools in our current study, but we use generic tools like a spreadsheet and a diagram editor with macro processing. After defining the process to use our ontology approach, we will design and implement its supporting tools.

There are many studies using NLP for requirements engineering. For example, inconsistencies in natural language requirements are discovered [17], conceptual models are semi-automatically generated by linguistic analysis [13], or formal method and lightweight natural language processing are used together [9]. However, it seems to be unclear how to handle domain knowledge and quality of requirements document itself in such studies. Studies to handle ambiguity in use case descriptions written in natural language exist [6], [7], but they also unclearly handled domain knowledge. There is a CASE tool to generate class diagrams from requirements [8], and a description called "world model" is used in the tool. The world model corresponds to the ontology in our study, but the world model is not used to interpret a requirements document but to make up for a deficiency of such a document.

Domain ontology in high quality is indispensable for our study, thus how to develop or acquire such ontology should

be also studied. We will introduce data mining techniques on several kinds of natural language descriptions such as users manuals, change histories, use case descriptions, scenarios and so on. Such kinds of documents can be requirements specification [2]. However, most methods for building ontology are ambiguous, thus the quality and efficiency of building ontology depend on the skills of each engineer [4]. Therefore, we have to explore systematical procedure to build ontology. Normally, we focus on the frequency of the occurrences of words or phrases in the documents when we build ontology. In addition, comparison among several documents also helps to build ontology in high quality [12].

COTS (Commercial Off-the-Shelf Software) and OISR (Off-the-Shelf Information Systems) will be easily selected by ontology generated from documents about COTS and OISR. Ontology is already used in OISR selection [15], and techniques in such study will be able to be used in requirements analysis.

There is a study to predict source code changes by mining change history [16], but there is no such studies in requirements analysis. In contrast to source codes, there are no unified and formal languages in requirements documents and change history thus it is hard to analyze them in requirements analysis. In our study, ontology plays a role to relate different versions of documents and their change histories with each other, thus we can predict changes in requirements documents.

In our study, quality characteristics are also represented as concepts in ontology. However, such characteristics are represented in a goal model and such goal model and ontology are combined in a study [5]. We also have our own goal oriented requirements model [11], thus we try to explore the possibility to combine a goal model and ontology. With respect to extending a model for semantic processing, we have to take implementation issues into account. To add knowledge about implementation into ontology, tasks in design and implementation phases could be supported by the ontology. Knowledge representation in WinWin [3] can be regarded as ontology with knowledge about implementation, and we will be able to gain an insight from the representation.

Acknowledgement

This study is supported by Grant-in-Aid Scientific Research #16016230 on Priority Area “Informatics” (Area #006).

References

[1] IEEE Recommended Practice for Software Requirements Specifications, 1998. IEEE Std. 830-1998.

[2] D. M. Berry, K. Daudjee, J. Dong, I. Fainchtein, M. A. Nelson, T. Nelson, and L. Ou. Users manual as a requirements specification: case studies. *Requirements Engineering*, 9(1):67 – 82, Feb. 2004.

[3] B. Boehm and H. In. Identifying Quality-Requirement Conflict. *Software*, 13(2):25–35, Mar. 1996. IEEE.

[4] K. K. Breitman and J. C. S. do Prado Leite. Ontology as a Requirements Engineering Product. In *11th IEEE International Requirements Engineering Conference (RE'03)*, pages 309–319, Sep. 2003.

[5] L. M. Cysneiros, J. C. S. do Prado Leite, and J. de Melo Sabat Neto. A Framework for Integrating Non-Functional Requirements into Conceptual Models. *Requirements Engineering*, 6(2):97 – 115, Jun. 2001.

[6] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Application of Linguistic Techniques for Use Case Analysis. In *IEEE Joint International Conference on Requirements Engineering (RE'02)*, pages 157–164, Sep. 2002.

[7] A. Fantechi1, S. Gnesi, G. Lami, and A. Maccari. Applications of linguistic techniques for use case analysis. *Requirements Engineering*, 8(3):161 – 170, Aug. 2003.

[8] R. Gaizauskas and H. Harmain. CM-Builder: An Automated NL-Based CASE Tool. In *The Fifteenth IEEE International Conference on Automated Software Engineering (ASE'00)*, pages 45–53, Grenoble, France, Sep. 2000.

[9] V. Gervasi and B. Nuseibeh. Lightweight Validation of Natural Language Requirements: A Case Study. In *4th International Conference on Requirements Engineering (ICRE'00)*, pages 140–148, Schaumburg, Illinois, Jun. 2000.

[10] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[11] H. Kaiya, H. Horai, and M. Saeki. AGORA: Attributed Goal-Oriented Requirements Analysis Method. In *IEEE Joint International Requirements Engineering Conference, RE'02*, pages 13–22, Sep. 2002.

[12] R. Lecceuche. Finding Comparatively Important Concepts between Texts. In *The Fifteenth IEEE International Conference on Automated Software Engineering (ASE'00)*, pages 55–60, Grenoble, France, Sep. 2000.

[13] S. P. Overmyer, B. Lavoie, and O. Rambow. Conceptual Modeling through Linguistic Analysis Using LIDA. In *23rd International Conference on Software Engineering (ICSE'01)*, pages 401–410, Toronto, Canada, May 2001.

[14] M. Saeki, H. Horai, and H. Enomoto. Software Development Process from Natural Language Specification. In *Proc. of 11th International Conference on Software Engineering*, pages 64–73, 1989.

[15] P. Soffer, B. Golany, D. Dori, and Y. Wand. Modelling Off-the-Shelf Information Systems Requirements: An Ontological Approach. *Requirements Engineering*, 6(3):183 – 199, Oct. 2001.

[16] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting Source Code Changes by Mining Change History . *IEEE Trans. on Software Engineering*, 30(9):574–586, Sep. 2004.

[17] D. Zowghi, V. Gervasi, and A. McRae. Using Default Reasoning to Discover Inconsistencies in Natural Language Requirements. In *Eighth Asia-Pacific Software Engineering Conference (APSEC'01)*, pages 113–120, Dec. 2001.