ELSEVIER

Information and Software Technology 42 (2000) 429–438

INFORMATION AND SOFTWARE TECHNOLOGY

www.elsevier.nl/locate/infsof

Implementation of an efficient requirements-analysis supporting system using similarity measure techniques

S. Park*, H. Kim, Y. Ko, J. Seo

Department of Computer Science, Sogang University, Seoul 121-742, South Korea

Received 20 May 1999; received in revised form 31 August 1999; accepted 19 November 1999

Abstract

As software becomes more complicated and larger, the software engineer's requirements-analysis becomes an important and uneasy activity. This paper proposes a requirements-analysis supporting system that supports informal requirements-analysis. The proposed system measures the similarity between requirement sentences to identify possible redundancies and inconsistencies, and extracts the possible ambiguous requirements. The similarity measurement method combines a sliding window model and a parser model. Using these methods, the proposed system supports to trace dependency between documents and improve quality of requirement sentences. Efficiency of the proposed system and a process for requirement specification analysis using the system are presented. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Requirements engineering; Natural language processing; Software analysis; Similarity measurements

1. Introduction

Requirements-analysis plays an increasingly important role in software development because most problems are caused by an incorrect analysis of customer's requirements. Some specific reasons why a correct understanding of customer requirements include:

1. errors in system level requirements that are not corrected early in the software development life cycle ultimately end up as maintenance problems in the delivered software. Recent studies show that while we expend 5% of the total cost of a major system on design and development, 70% of the ability to influence the quality comes from this meager amount [1];
2. the total system failure is often caused by the misunderstanding of requirements between a customer and an analyst;
3. the later in the development life cycle that a software error is detected, the more expensive it will be to repair.

In terms of percent of effort expended, requirements-analysis consumes merely 5% of the total cost of the effort, while affording 50% of the leverage to influence improved quality. In terms of pre-production costs, system definition absorbs 5% of these costs and provides 50% of the leverage to influence the improvement of quality [1]. Lastly, many errors remain latent and are not detected until well after the stage at which they are made because a software system has become larger and more complex. If it was possible to understand better how to detect flaws during the systems level requirements phase, there could be extensive savings in development cost and time.

Software requirements often have some additional problems like imprecision, conflict, inconsistency, ambiguity and incompleteness, because they are generally written in a natural language format and are affected by the characteristic of an analyst. If these problems are handed over to the next phase without being resolved, they may generate other critical problems because the next phase of the software development cycle is based on the current one. In the worst case, we may have to totally redesign and reconstruct the system. So, it is very important that we check dependency and consistency between requirements, and check for conflicts and ambiguity. In this paper, we propose an efficient requirements-analysis supporting system in Korean, which is based on similarity measures in information retrieval. The system can alleviate some critical problems such as conflict, inconsistency and ambiguity.

This paper begins with a survey of related works in

* Corresponding author. Tel.: +82-705-8928; fax: +82-2-704-8485.

E-mail addresses: sypark@ccs.sogang.ac.kr (S. Park), hskim@nlpzodiac.sogang.ac.kr (H. Kim), kyj@nlpzodiac.sogang.ac.kr (Y. Ko), seojy@ccs.sogang.ac.kr (J. Seo).

PII: S0950-5849(99)00102-0

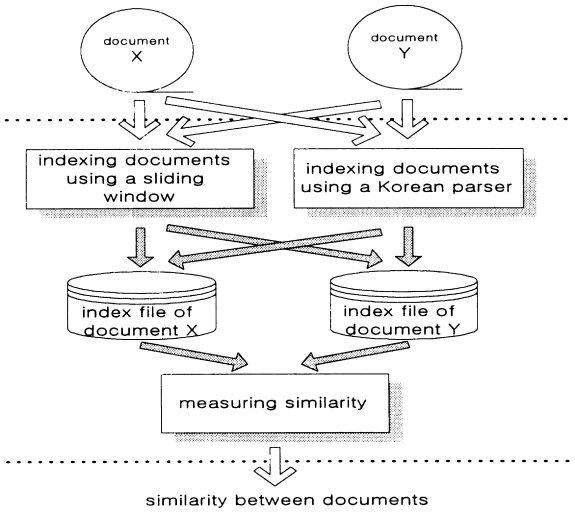

Fig. 1. The module to measure similarity between documents.

Section 2. In Section 3, we propose a requirements-analysis system and explain an indexing method and similarity measuring methods. In Section 4, we report some promising experimental results. After showing the implementation of our system in Section 5, we draw a few conclusions in Section 6.

2. Related works

There has been extensive work in software engineering to build automatic requirements-analysis supporting systems [2–6]. These systems can be roughly classified into two groups according to the approach adopted, the free-text indexing approach as defined in information retrieval (IR), and the knowledge-based approach as defined in artificial intelligence (AI) [2]. The AI approach can be useful in some applications. However, the IR approach is generally preferred for the following reasons.

- Cost: the system is built entirely automatically.
- Transportability: the system can be rebuilt for a domain, since it requires manually provided domain knowledge.
- Scalability: the repository can be easily updated when new components are inserted.

Traditional requirements-analysis supporting systems have used similarity measures and keywords extracted from the specifications. Palmer proposed the Two-Tired Clustering (TTC) algorithm [4] which indexes and clusters requirements-specifications. It, first, identifies the verbs and keywords of each requirement, then it clusters these requirements by functionality. Next, each cluster is subdivided using cosine similarity between clusters. To automatically construct software libraries, Maarek proposed the method to index and cluster UNIX manual pages, which uses a sliding window and a level clustering technique [4]. Many researchers in information retrieval have focused on indexing documents and measuring similarity between documents using Machine Readable Dictionary (MRD) like thesaurus or a linguistic feature like reiteration and collocation of words [7–13]. However, methods using MRD or linguistic features have some problems. They do not give an analyst efficient information like conflict, inconsistency, and ambiguity between requirements in a specification. Although the methods using the thesaurus outperform others in precision, building an MRD is difficult, domain specific and time-consuming. Although there has been lots of work on indexing and clustering in information retrieval, we cannot easily find a system based on them.

3. Indexing and measuring schemes

3.1. An overview of the requirements-analysis supporting system

A requirements-analysis supporting system consists of three modules: one to measure similarity between documents, one to measure similarity between sentences and one to extract ambiguous sentences. Measuring similarity between documents is useful for tracing dependency between mixed specifications in a distributed developing environment. The module consists of two sub-modules as shown in Fig. 1.

The sub-module to index documents extracts word pairs including co-occurrence information by the use of a sliding window and a parser using the syntactic dependency relations. The system gets simple co-occurrence information between words within a span of five words using the sliding window. The parser generates modifier–modified relations between word-phrases, which are called dependency relations. The dependency relation supplements simple co-occurrence information extracted by the sliding window and covers co-occurrence information over a span of five words. The module to measure similarity between sentences is useful for checking consistency between a requirement in a high-level specification and a requirement in a low-level specification or between requirements in a specification. The module is similar to Fig. 1, as shown in Fig. 2. However, algorithms that measure similarity have to be different because the number of indices extracted from a sentence is much less than the number of indices extracted from a document. Therefore, the system uses two different methods.

The module to extract ambiguous sentences finds inadequate sentences in a requirements-specification. To support this function, the system makes use of a word set and a Part-Of-Speech (POS) tagger as shown in Fig. 3. The ambiguous words in the set are manually extracted from a lot of requirements-specifications.

3.2. The indexing scheme

There has been much work in IR dealing with natural

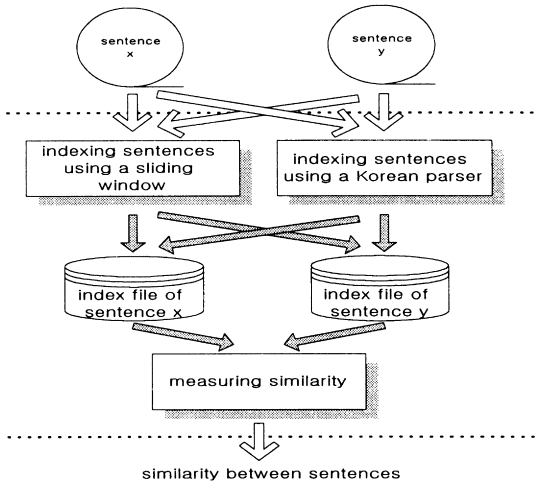

Fig. 2. The module to measure similarity between sentences.

language text: a large variety of techniques have been devised for indexing, classifying and retrieving documents [14,15]. One of the main concerns in IR is the automatic indexing of documents, which consists of producing, for each document, a set of indices that form a *profile* of the document. A *profile* is a short-form description of a document, easier to manipulate than the entire document, which plays the role of a surrogate at the retrieval stage. In other words, it is a keyword file including content words of documents. Another main concern in IR is the methods for generating a *profile* from a given document. The methods are generally divided into two types. One is a single-term index, each of which is a single word without contextual information. The other is a term-phrase index, which uses term phrase as indexing units rather than single terms so as to refine the meaning of constituent words. Although the former method can extract a lot of indices from a small document, it cannot provide a refined analysis including semantics. A possible solution to this problem is to use information such as POS or dependency relation in order to provide further control over phrase formation. In this paper, we propose an efficient term-phrase index scheme that uses a sliding window and a syntactic parser.

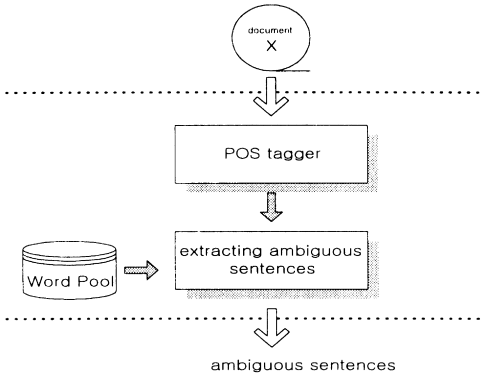

Fig. 3. The module to extract ambiguous sentences.

3.2.1. The sliding window technique

A term-phrase index is derived from the notation of Lexical Affinity (LA). In linguistics a syntagmatic LA, also termed a lexical relation, between two units of language stands for a correlation of their common appearance in the utterances of language [16]. The observation of LAs in large textual corpora has been shown to convey information on both syntactic and semantic levels and provides us with a powerful way of taking context into account [17]. Ideally, LAs are extracted from a text by paring it, since two words share an LA if they are involved in a modifier–modified relation. Unfortunately, automatic syntactic parser of a free-style text is still not very efficient, and the number of LAs extracted by parsing is not enough to measure the similarity between documents or sentences if we do not have a large corpus. So, the system makes use of simple co-occurrence information extracted by a sliding window and supplement its flaws using a syntactic parser. It has been shown by Martin et al. [18] that 98% of lexical relations relate words that are separated by at most five words within a single sentence. Therefore, most of the LAs involving a word w can be extracted by examining the neighborhood of each occurrence of w within a span of five words (-5 words and $+5$ words around w). In this paper, the system just extracts LAs within $+5$ words around w because modified words in Korean usually locate after modifiers. A sliding window technique [2] consists of sliding a window over the text and storing pairs of words involving the head of the window if it is an open-class word[1] and any of the other open-class elements of the window. The window is slid word by word from the first word of the sentence to the last, the size of the window decreasing at the end of the sentence so as not to cross sentence boundaries, since LAs cannot relate words belonging to different sentences. The window size being smaller than a constant, the number of extracted LAs is in linear relationship to the number of words in the document. Fig. 4 shows an example extracting LAs by the use of the technique.

3.2.2. The parser using syntactic dependency relations

Although a sliding window technique extracts a lot of LAs, it can not extract LAs out of the boundaries of a fixed window. Dependency relations extracted from a text by parsing it can supplement the weak point because the distance between a modifier and a modified word is flexible from 1 to the length of a sentence. The system extracts LAs using a syntactic parser that is based on dependency grammar [19,20] and the sliding window. Although the LAs extracted by the parser play an important role in extracting long distant relations, they have some problems as indices. First, a dependency relation shows the relation between word-phrases rather than the relation between words. If a

[1] In general, open-class words include nouns, verbs, adjectives and adverbs, while closed-words are pronouns, prepositions, conjunctions and interjections.

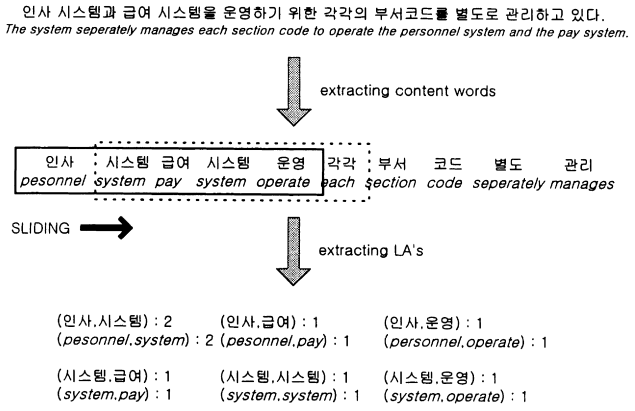

Fig. 4. An example extracting LAs by the use of the sliding window.

word phrase consists of some open-class words, dependency relations within a word-phrase are not extracted. For example, if "*kakkakuy pwusekhotunun pyeltolo kwatolo kwanlitoynta.* (The codes of each department are separately managed.)" is analyzed using the parser, the noun phrase, *kakkakuy pwusekhotu* (*The codes of each department*), depends on the verb phrase, *kwanlitoynta* (*are managed*). However, the relation between *kakkakuy pwuse* (*each department*) and *khotu* (*the codes*) is not extracted. The relation between *khotu* (*the code*) and *kwanlitoynta* (*are managed*) is also not extracted. Second, verbs in Korean are often generated using *a noun + do/become* or *a noun + be*. For example, *retrieve* is generated using *retrieval + do*. In those cases, a modifier usually depends on *do/become* or *be*. For example, if "*sinkyukhotuka iplyektoymyen, kwanlyen khotulul DBeyse kemsaykhanta.* (When new codes are input, the system retrieves the DB associated with the codes.)" is analyzed using the parser, *DB* depends on *do* because *retrieve* is expressed as *retrieval + do*. However, the relation is not suitable for index because *do* carries less information than *retrieval*. The relation that *DB* depends on *retrieval* is more suitable for index. In this paper, we propose some heuristics to overcome these problems as shown in Fig. 5. Fig. 6 shows an example for dependency relations to be changed by the heuristics.

3.2.3. Generating a profile from lexical affinities

When analyzing a document, many potential LAs are identified. Some of these LAs are conceptually important and some are not. Although frequency of appearance is a good indicator of relevance, some noise exists, mainly due to words appearing too often in a given context like *be* and *do/become*. To reduce the influence of such words, it is necessary to select the most representative ones among LAs; i.e. those containing the most information among LAs. The *quantity of information*, as shown in Eq. (1), is defined as a measure evaluating the resolving power of a word within a corpus [14,21].

$$\mathrm{INFO}(w) = -\log_2 (P\{w\}), \quad \text{where } P\{w\} \text{ is the observed probability of occurrence } w \text{ in the corpus.} \tag{1}$$

Eq. (1) means that the more frequent a word is in a domain, the less information it carries. If we consider words within the textual universe as independent variables to simplify the computation, the *quantity of information* of an LA can be approximated [2] as

$$\mathrm{INFO}((w1, w2)) = -\log_2 (P\{w1, w2\}) \approx -\log_2 (P\{w1\} \times P\{w2\}) \tag{2}$$

Then, the resolving power of an LA in a given document is computed using the frequency of the LA in the document and the *quantity of information* of the LA as shown in Eq. (3). The resolving power is called *ρ-score* [2]:

$$\rho((w1, w2, f) = f \times \mathrm{INFO}((w1, w2)), \quad \text{where } f \text{ is the frequency of the LAs.} \tag{3}$$

To be able to compare the relative performances in terms of resolving power of different documents, the system transforms the *ρ-score* into a standardized score as shown in Eq. (4). In Eq. 4, $\bar{\rho}$ is the mean of the *ρ-scores*, and σ is the standard derivation of the *ρ-scores*. The standardized score is called *z-score* [2]:

$$\rho_z = \frac{(\rho - \bar{\rho})}{\sigma} \tag{4}$$

In this paper, the system extracts LAs using the sliding window and the parser and generates *profiles* including *z-scores* of these LAs. Each *profile* represents a requirements-specification.

1. The last content word in a modifying word-phrase depends on the last word in a modified word-phrase. For example, in "정보검색은 자연어처리분야이다. (Information retrieval is a part of natural language processing. ", the last word *검색 (retrieval)* in *정보검색 (Information retrieval)* depends on the last word *분야 (part)* in *자연어처리분야 (a part of natural language processing).*

2. Each content word in a word-phrase depends on the next word in the phrase. For example, in "정보검색기법 (information retrieval scheme)", *정보 (information)* depends on *검색 (retrieval)*, and *검색 (retrieval)* depends on *기법 (scheme).*

3. If a modified word consists of a *noun + do/become* or a *noun + be*, a modifier depends on the noun. For example, in "DB 를 검색한다. (The system retrieves DB.)", *DB* depends on *검색 (retrieval).*

Fig. 5. Indexing heuristics.

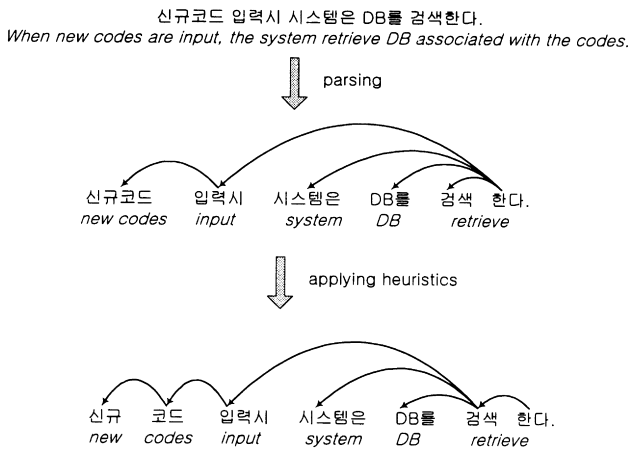

Fig. 6. An example applying the heuristics.

3.3. Measure of similarity between documents

In information retrieval, numerous measures of similarity between documents have been defined [22]. The simplest of all is defined as:

$$|X \cap Y|, \text{ where } X \text{ and } Y \text{ are the index file of two documents.} \tag{5}$$

This measure represents the number of common index units. The other measures like *Dice's coefficient*, *Jaccard's coefficient* and *Salton's cosine coefficient* are almost normalized version of Eq. (5) [22]. Theses measures perform well in IR, but they are not applicable to the system because LAs have more information than just the presence or absence of index units. Therefore, the system uses the measure like Eq. (6) because the equation can reflect the *z-scores* of LAs without information loss [2]:

$$\delta(X, Y) = \sum_{i \in p(X) \cap p(Y)} (\rho X(i) \times \rho Y(i)) \tag{6}$$

In Eq. (6), $p(X)$ means the *profile* of a document X, and $p(Y)$ means the *profile* of a document Y. $\rho X(i)$ means the *z-score* of ith index of $p(X)$, and $\rho Y(i)$ means the *z-score* of ith index of $p(Y)$. In this paper, the system adds 1 point to each *z-score* and consider only LAs with the *z-score* which is greater than 0 in order to alleviate sparse data problems caused by small corpus.

3.4. Measure of similarity between sentences

To measure similarity between sentences, the system uses another measure of similarity, *Salton's cosine coefficient* [14], as shown in Eq. (7), because Eq. (6) is not applicable to measure similarity between sentences according to the following reasons. Firstly, f in Eq. (6) means the frequency of LAs in a document. If the system counts f between sentences, the system cannot get a meaningful value because the f is always around 1. Secondly, the number of LAs for measuring the similarity between sentences is much less than the number of LAs for measuring the similarity between documents. So, $\bar{\rho}$s, means of *ρ-scores*, are similar to each other, and *z-scores* have similar values as a result.

$$\cos(x, y) = \frac{|x \cap y|}{|x| \times |y|} \tag{7}$$

In Eq. (7), $|x|$ is the number of LAs in sentence x, and $|y|$ is the number of LAs in sentence y. $|x \cap y|$ is the number of common LAs in sentence x and sentence y.

3.5. Extraction of ambiguous sentences

Ambiguous words are words that could cause requirement sentences to be interpreted in multiple ways. For example, "For up to 12 aircraft, the small display format needs to be updated frequently, otherwise large display format needs to be updated occasionally". In this requirement sentence 'small display format' and 'large display format' are ambiguous since different engineers might have different ideas on 'small' and 'large' and so are 'frequently' and 'occasionally'. In this case, the words, 'small', 'large', 'frequently', and 'occasionally', causes requirement sentence ambiguous. We call these types of word ambiguous words.

To extract ambiguous sentences from a document, the system makes use of a POS tagger and a word set including ambiguous words which are extracted manually from a lot of requirement specifications. In processing, the system annotates a given document with POS tag. Then, it extracts verbs, adverbs and adjectives in the document and compares them with the words in the word set. If a word in the document is in the word set, the system displays the sentence including that word because the sentence has a high possibility of being ambiguous.

To construct a word set including ambiguous words, we first collect clearly ambiguous words such as 'small', 'large', 'little', 'many', etc. and give each word an initial weight of 1. Then, the weights are updated by analyst's feedback while he/she analyzes documents. In other words, if a candidate sentence is extracted because the sentence includes an ambiguous word in the set, he/she checks whether the sentence is really ambiguous or not. If the sentence is ambiguous, he/she gives the system a feedback that updates the weight of the ambiguous word as shown in Eq. (8).

$$W(w_i) = W(w_i) + 1, \text{ if } w_i \subset S_j \tag{8}$$

In Eq. (8), $W(w_i)$ means the weight of the ith ambiguous word, and S_j means a set of words in the jth sentence that is decided to be an ambiguous sentence by an analyst. In this paper, the system ranks ambiguous words according to the weight and shows candidate ambiguous sentences according to the word rank. This function helps the analyst to give more attention to ambiguous words that are often used in the document. The recall rate and the precision of the

Table 1
The precision (%) of the three methods

Rank	The sliding window method	The syntactic parser method	The proposed method
Top-1	60.6	9.1	72.7
Top-3	81.8	42.4	90.9

function could be improved by adding more words and by using better weight factor.

4. Experimental results

4.1. The experimental data

For the experiment, we collected requirements specifications from real fields. The data consisted of 33 documents with 1090 sentences. We divided them into two test sets: one to evaluate the module to measure similarity between documents, the other to evaluate the module to measure similarity between sentences.

The module to measure similarity between documents can be used to check dependency between requirements-specifications in a distributed developing environment. So, we assumed such a situation and constructed the experimental data as follows. We divided a document into two sub-documents. We constructed *data-A* and *data-B* including each of the 33 sub-documents. The 33 sub-documents in *data-A* include odd lines in the 33 original documents. Those in *data-B* include even lines. In the experiment, the module measures similarities between a sub-document in *data-A* and all sub-documents in *data B*. If a sub-document in *data-B* has the highest rank, and the given sub-document in *data-A* and the sub-document in *data-B* were a same document before being separated, we regard it as the correct answer.

The module to measure similarity between sentences can be used to check inconsistency between a high-level sentence and a low-level sentence. So, we divided the data into two groups by hand. One consisted of high-level documents and the other, low-level documents. In the experiment, the module measures similarities between a sentence in a high-level document and all sentences in a low-level document. If the module finds a correct low-level sentence of the given high-level sentence, we regard it as the correct answer.

To evaluate the module to extract ambiguous sentences, we used the same experimental data used for evaluating the measure of similarity between sentences. The word set, including ambiguous words, was built manually by two graduate students. The set includes 16 ambiguous words like *manhta* (*many*), *cekta* (*little*), *swipta* (*easy*), *khuta* (*big*), *cakta* (*small*), *ppaluta* (*fast*), *nulita* (*slow*), *phyenli-hata* (*convenient*), etc.

4.2. The experimental result

To evaluate the module to measure similarity between documents, we compared three methods to extract LAs: a method using only the sliding window, a method using only the syntactic parser and the proposed integrated method. The precision in Table 1 means how many sub-documents in *data-B* are included in the given top rank with the correct answer as shown in Eq. (9).

$$precision = \frac{\text{the number of correct answers in the given rank}}{\text{the number of sub-documents in } data\text{-}B} \times 100 \quad (9)$$

The table reveals that the proposed method is superior to the other methods. Fig. 7 shows the precision of each method through graph.

The method using only the syntactic parser had much worse precision than the others. We believe that it is caused by the sparse data problem. Fig. 8 shows the ranks of correct documents in *data-B* that are compared to each document in *data-A* using the proposed method. In Fig. 8, the proposed

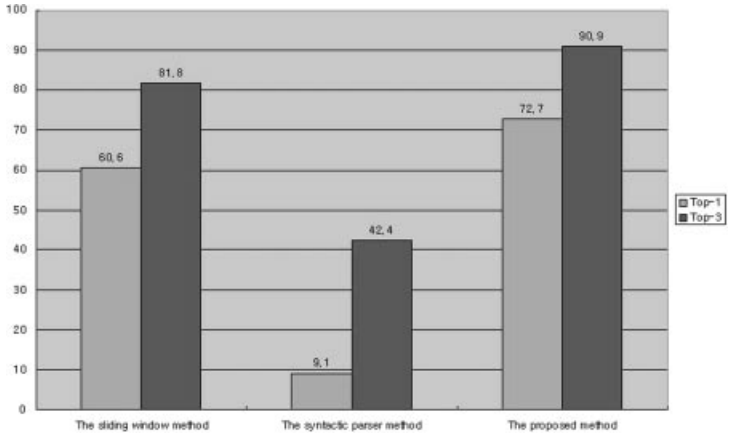

Fig. 7. The precision of the three methods.

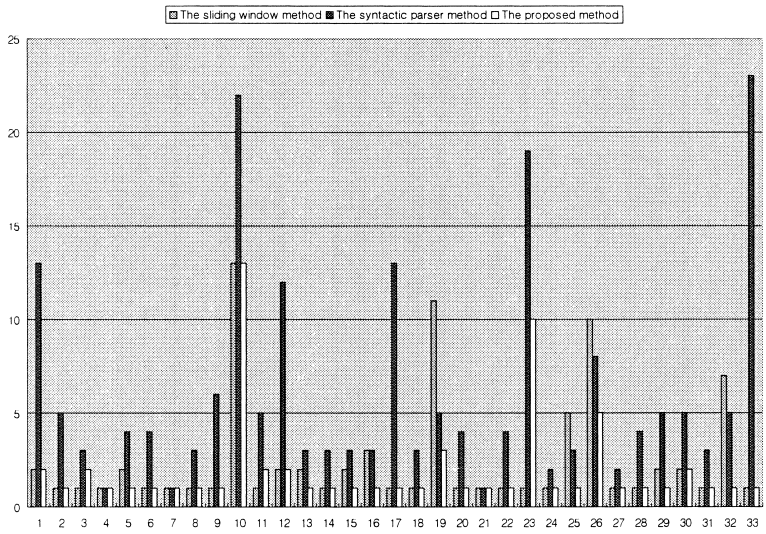

Fig. 8. The ranks of correct documents.

method ranked the correct answer of the 16th document first although the other methods ranked it third. It reveals that the proposed method efficiently puts together co-occurrence information and dependency relation. If an analyst makes use of the proposed module, he/she can find a target document with a precision of 90% after looking up just three documents. It means that he/she can reduce his/her efforts by 9.4%.

To evaluate the module to measure similarity between sentences, we compared two indexing schemes: single-term indexing scheme and term-phrase indexing scheme. The term phrase indexing scheme, is superior to the single-term indexing scheme as shown in Table 2. It reveals that LAs carry more information than single-term indices. If an analyst makes use of the proposed module, he/she can find low-level sentences that are similar to a high-level sentence with 72.7% precision. It means that he/she can efficiently check inconsistency by using similarity between a high-level sentence and low-level sentence because similar sentences have the higher possibility that either they are redundant or conflict.

To evaluate the module to extract ambiguous sentences, we compared the number of ambiguous sentences in whole documents with the number of candidate sentences extracted by the system. The system found all ambiguous sentences and the recall rate was 100%. However, we excluded ambiguous requirements sentences based on the context in our experiments. The precision was 47.1%. Table 3 shows the experimental result of ambiguous sentence extraction.

Table 2
The precision (%) of the two indexing schemes

Rank	Single-term indexing scheme	Term-phrase indexing scheme
Top-3	68.2	72.7

As shown in the table the proposed method is effective, although it adopts a straight forward approach. If an analyst make use of the module, he/she can find all ambiguous sentences after looking up just 34 sentences out of 1090 sentences. It means that he/she can reduce his/her efforts by 96.9% in searching for ambiguous sentences. The recall rate depends on the number of words in the ambiguous word set, and the precision depends on the quality of the words. We believe that the recall rate will not be down if new words are continually added. To keep the recall rate and precision, we designed a system, wherein, the set can be easily updated by a user. We expect that the module can be used to improve the quality of requirements-specifications.

5. Implementation

We implemented the requirements-analysis supporting system in a client–server architecture. The server was implemented on Solaris 2.5.1 using C language. The client was implemented on MS-Windows 98 using Visual Basic. Fig. 9 shows the initial snapshot of the client.

The client consists of a preprocessing part and a function part. The system generates *profiles* of requirements-specifications in the former and gives an analyst some useful tools in the latter. The system supports a function to trace dependency between documents, improve completeness in a document, reduce inconsistency between sentences and improve the quality of a document.

The system measures similarity between documents as shown in Fig. 10. Using this function, an analyst can trace the dependency between documents because he/she can easily find which specifications are associated with a new requirements specification.

Table 3
The recall rate and the precision of the module to extract ambiguous sentences

Number of total sentences	Number of ambiguous sentences/number of extracted candidate sentences	Recall (%)	Precision (%)
1090	16/34	100	47.1

The system measures the similarity between a sentence in a high-level document and a sentence in a low-level document as shown in Fig. 11. Using this function, an analyst can efficiently check consistency between documents because the system automatically displays which sentence in a low-level document is most similar to the given sentence in a high-level document. For example, the first sentence in Fig. 11 does not have any candidate sentences. It means that there are no sentences in the low-level document associated with the given sentence in the high-level document. Looking through those sentences, an analyzer can improve completeness of requirements-specifications.

Fig. 12 shows the function to measure similarity between sentences in a document. Using this function, an analyst can

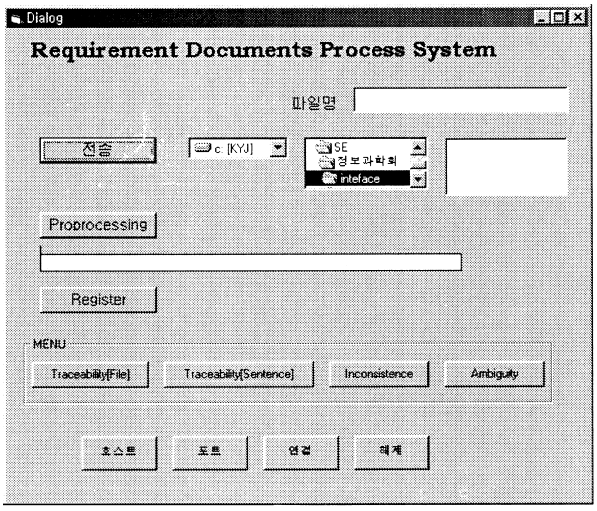

Fig. 9. The initial snapshot of the client.

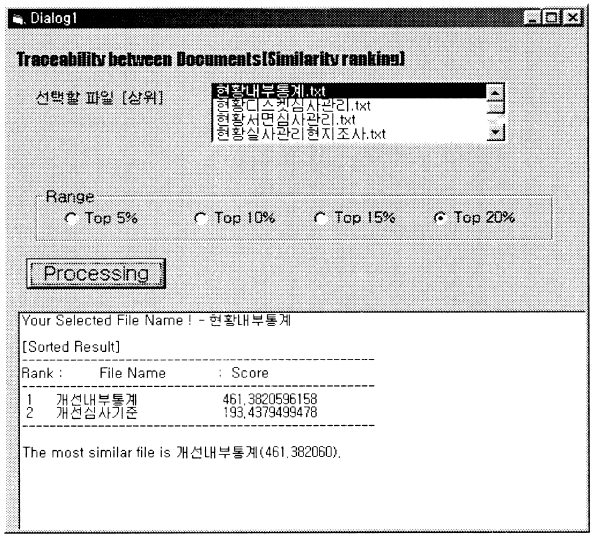

Fig. 10. The snapshot to measure similarity between documents.

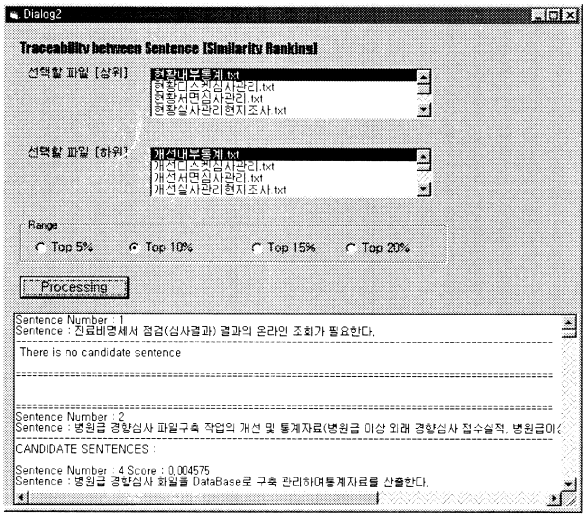

Fig. 11. The snapshot to measure between a high-level sentence and a low-level sentence.

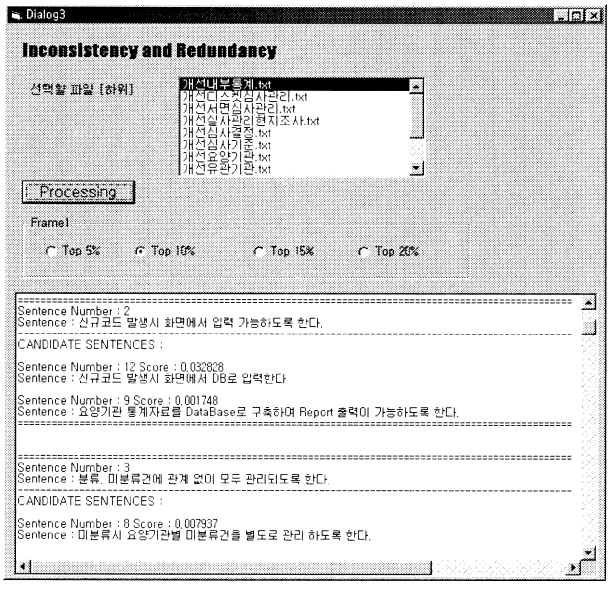

Fig. 12. The snapshot to measure similarity between sentences in a document.

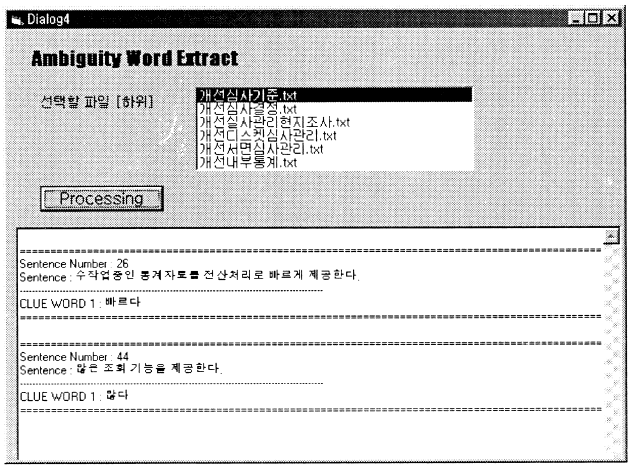

Fig. 13. The snapshot to display ambiguous sentences.

reduce inconsistency between sentences and can improve completeness in a document. For example, he/she can find the facts that the second sentence in Fig. 12, *Sentence Number:3* (*All documents, whether it is a classified document or not, should be maintained*), is inconsistent with its candidate sentence, *Sentence Number:8* (*Unclassified documents should be separately maintained*). And he/she can also find that the first sentence, *Sentence Number:2* (*When a new code is needed, system administrators should be able to input it using a keyboard*), is duplicate of the first candidate sentence, *Sentence Number:12* (*When a new code is needed, system administrators should be able to insert it into the DB using a keyboard*).

The system supports the function to display ambiguous sentences as shown in Fig. 13. This function helps an analyst to improve the quality of a document because non-quantifiable words produce an ambiguity problem.

6. Conclusions

In this paper, we proposed the requirement analysis supporting system that can be used to identify potential errors in requirements. The system supports functions to trace dependency between documents, improve completeness and reduce inconsistency between sentences that improve quality of a requirements-document. To provide these functions, it uses an indexing scheme and the measures of similarity that are used in information retrieval. We combined the sliding window method with the syntactic parser method including some heuristics in order to extract indices containing more information about a document or a sentence. The experimental results reveal that the combined method efficiently complements each method. To measure similarity between documents or sentences, the system uses *z -scores* and *Salton's cosine coefficients*. The system has some advantages such as simple framework, easy implementation because it was done only using a POS tagger, a syntactic parser and some heuristics. It also shows effectiveness in terms of performance. The developed system supports an analyst to identify potential problems of requirements-documents.

Acknowledgements

This research is supported by BK21 research fund.

Appendix A

This table shows the requirements examples used for the experiment. The examples are the requirements for building an integrated system of the medical insurance society. The first column shows sentences in high-level documents,

Sentences in a high-level document	Sentences in a low-level document
Errors of the budget data received from a company medical insurance society must be revised.	Error lists are sent back and they are revised by each company medical insurance society. The new system will have a function to revise errors of the budget data.
In a head office and branch offices, the state of a turning point report must be referred through online service.	2.1 Turning point reports of medical insurance society of each company, which are processed using UNIX system, are referred through online service. 2.2 A head office sends a turning point report file to each branch office.
As a head office send a business achievement statistical data of local society to each branch office, there is a very large amount of output (800pp).	3.1 A head office sends a business achievement statistical data to each branch office. And each branch office print it. 3.2 A business achievement statistical data of local society is made by UNIX system in each branch office. Therefore, its result is referred by online service.

which describe problems of the present system. The second column shows sentences in low-level documents, which describe schemes for improving each problem in the high-level documents.

References

[1] A.S. Shumskas, V.N. Askman, J.F. Cunningham, Conceptual information retrieval using RUBIC, Proceedings of the 10th Annual International ACMSIGIR Conference on Research and Development in Information Retrieval, 1987.

[2] Y. Maarek, D. Berry, G. Kaiser, An information retrieval approach for automatically construction software libraries, IEEE Transaction On Software Engineering 17 (8) (1991) 800–813.

[3] J.D. Palmer, Y. Liang, W. Wang, Proceedings of ASIS'90, Workshop on Classification Research, Toronto, Canada, 4 1990.

[4] J. Palmer, Y. Liang, Indexing and clustering of software requirements specifications, Information and Decision Technologies 18 (1992) 283–299.

[5] S. Park, J.D. Palmer, Automated Support to System Modeling from Informal Software Requirements, Proceedings of the sixth International Conference on Software Engineering and Knowledge Engineering, 1994.

[6] D. Samson, A knowledge-based requirements system, Productivity: Progress, Prospects, and Payoff, 27th Annual Technical Symposium, DC Chapter of the ACM and NBS, Washington, 1991.

[7] M. Hajime, H. Takeo, O. Manabu, Text segmentation with multiple surface linguistic cues, Proceedings of the COLING-ACL'98, 1998, pp. 881–885.

[8] M. Hearst, Multi-paragraph segmentation of expository text, Proceedings of the ACL'94, 1994.

[9] A. Jobbins, L. Evett, Text Segmentation using Reiteration and Collocation, Proceedings of the COLING-ACL'98, 1998, pp. 614–618.

[10] M. Kim, J. Klavans, K. McKeown, Linear Segmentation and Segment Significance, Proceedings of the Sixth International Workshop of Very Large Corpora (WVLC-6)1998, pp. 197–205.

[11] H. Kozima, Text Segmentation Based on Similarity between Words, Proceedings of ACL'93, 1993, pp. 286–288.

[12] D. Litman, R. Passonneau, Combining Multiple Knowledge Sources for Discourse Segmentation, Proceedings of the 33rd ACL, 1995.

[13] Y. Yaari, Segmentation of Expository Texts by Hierarchical Agglomerative Clustering, Proceedings of RANLP'97, 1997, pp. 135–142.

[14] G. Salton, M.J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, New York, 1983 Computer Series.

[15] G. Salton, Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer, Addison-Wesley, Reading, MA, 1989.

[16] F. Saussure, Cours de Linguistique Generale, 4, Librairie Payot, Paris, 1949.

[17] F.A. Smadja, Lexical co-occurrence: the missing link, Literary and Linguistic Computing 4 (3) (1989) 14–25.

[18] W.J.R. Martin, B.P.F. Al, P.J.G. van Sterkenburg, On the processing of a text corpus: from textual data to lexicographic information, in: R.R.K. Hartmann (Ed.), Lexicography: Principles and Practice, Academic Press, London, 1983, Applied Language Studies Series.

[19] P. Hellwig, Dependency unification grammar, Proceedings of Colling'86, 1986, pp. 195–198.

[20] I.A. Mel'cuk, Dependency Syntax: Theory and Practice, State University of New York Press, New York, 1988.

[21] R. Ash, Information Theory, Wiley-Interscience, New York, 1965.

[22] C.J. Rijsbergen, Information Retrieval, 2, Butterworths, Stoneham, MA, 1979.