Toward an Automatic Reusable Software Using Textual Entailment

Yasser Kotb* and Takuya Katayama**

Japan Advanced Institute of Science and Technology
School of Information Science
Asahidai 1-1, Nomi, 923-1292, Ishikawa, Japan

IS-RR-2006-016
November 24, 2006

* Corresponding author: kotb@jaist.ac.jp.
** Corresponding author: katayama@jaist.ac.jp

Toward an Automatic Reusable Software Using Textual Entailment

Yasser Kotb and Takuya Katayama
Japan Advanced Institute of Science and Technology
School of Information Science
Asahidai 1-1, Nomi, 923-1292, Ishikawa, Japan
+81 761 51 1261
{kotb, katayama} @ jaist.ac.jp

ABSTRACT

Software reuse is the use of software resources from all stages of the software development process in new applications. Rather the *opportunistically* reuse (e.g., by cutting and pasting code snippets from existing programs into new programs), we consider here the *systematic* reuse. This paper introduces a novel framework that helps identify automatically the segments of software that can be reused in building new software. This framework based on a UML use case textual specification and a recent natural language processing technique called *Textual Entailment*. We argue that such approach will reduce software development cycle time and cost and improve software quality.

Categories and Subject Descriptors

D.1.0 **[Programming Techniques]**: General; D.2.1 **[Software Engineering]**: Requirements/Specification – languages, methodologies, tools; D.2.2 **[Software Engineering]**: Design Tools and Techniques-Programmer workbench, Software libraries D.2.7 **[Distribution, Maintenance, and Enhancement]** Documentation, Portability; D.2.9 **[Management Software]** Cost estimation, Software configuration management, Quality assurance (SQA); D.2.13 **[Reusable Software]**: Domain engineering, Reusable libraries, Reuse models; D.2.m **[Software Engineering]** Miscellaneous-Reusable software; D.3.4 **[Programming Languages]** Processors. I.7.1 **[Document and Text Processing]** Document management, Languages; I.2.7 **[Artificial Intelligent]**: Natural Language Processing-Language parsing and understanding, Text analysis.

General Terms

Design, Documentation, Economics, Languages, Management, Performance, Reliably, Standardization.

Keywords

UML, Textual Entailment, Use Case, Software Reuse, Quality.

1. INTRODUCTION

Software reuse is the process of creating software systems from existing software rather than building software systems from scratch. [1]. Reuse has been a popular topic of discussion for over three decades in the software engineering community and still on. There are two main categories for reuse: *opportunistically software reuse* and *Systematic software reuse*. Opportunistically reuse means to choice the appropriate source code parts/components that can be used in the new software system from old ones. Many developers have successfully applied reuse opportunistically, e.g., by cutting and pasting code snippets from existing programs into new programs. Opportunistic reuse works fine in a limited way for individual programmers or small groups. However, it does not scale up across business units or enterprises to provide systematic software reuse.

Systematic software reuse means constructing and applying multi-use assets like architectures, patterns, components, and frameworks as assets base. This assets base uses to derive the appropriate parts that are useful to build the new software systems. Systematic reuse is a promising means to reduce development cycle time and cost, improve software quality, and leverage existing effort.

It has long been recognized that reuse in the early development phases of an application can reduce the effort of producing new software and improve their quality. In this paper, we present a novel framework that helps identify automatically the segments of software that can be reused in building new software. This framework based on a UML use case textual specification and a recent natural language processing technique called *Textual Entailment*.

The concept *textual entailment* is a new approach that is applied in natural language processing field [2] [3]. It used to indicate the state in which the semantics of natural language written text can be inferred from the semantics of another one. Especially, if the truth of a text segment entails the truth of another text segment. This paper addresses using textual entailment approach in the software reuse field. This new approach based on the use case stage inside the software development cycle. This stage is one of the earliest stages in the software development life cycle. Textual use case describes the necessary steps required to illustrate the behavior of actors outside the system with the use cases that lies inside the system.

The rest of the paper is organized in the following way. Section 2 gives a brief introduction to text entailment approach. In Section 3, introduces briefly textual use case. Section 4 illustrates our motivation example that will play role to clarify our ideas in next sections. Section 5 proposes our textual entailment reuse software framework that automatically checks whether the new system/subsystem can be inferred from software asset database and then generates the new system/subsystem from the old use case scenarios. Section 5 addresses related research topics to our research. Lastly, Section 6, we draw attention for some conclusions and some idea about future works.

2. TEXT ENTAILMENT APPROACH

Textual entailment defines the task that requires to recognize, given two text fragments, whether the meaning of one text can be inferred (entailed) from another text. More concretely, *textual entailment* (TE) is defined as a directional relationship

between pairs of text expressions, denoted by ***T*** - the entailing "Text", and ***H*** - the entailed "Hypothesis". We say that ***T*** ***entails*** ***H*** if the meaning of *H* can be inferred from the meaning of *T*, as would typically be interpreted by people. For example, given the texts:

1. A Union Pacific freight train hit five people.
2. A Union Pacific freight train struck five people

it is clear that the meaning of the second one can be inferred from the meaning of the first one. Therefore, it is said that textual entailment exists between both texts. This somewhat informal definition is based on (and assumes) common human understanding of language as well as common background knowledge. It is similar in spirit to evaluation of applied tasks such as *Question Answering* (QA) and *Information Extraction* (IE), in which humans need to judge whether the target answer or relation can indeed be inferred from a given candidate text.

In fact, one of the fundamental characteristic of natural language is the variability of text semantic expression, where the same meaning can be expressed by, or inferred from, different texts. This natural language characteristic may be considered as the language ambiguity twin problem. Both can be combined to form the many-to-many mapping between language expressions and meanings during the language processing approaches. Many natural language processing applications, such as question answering, information extraction, multi-document summarization, and machine translation evaluation, need a model for this variability characteristic in order to recognize that a particular target meaning can be inferred from different text variants. Although there are many different applications that are need similar models for text semantic variability. This problem has been addressed many times in a different application-oriented manners and method views that are evaluated by their impact on final application performance. For example, one of the earliest applications of text similarity is perhaps the vectorial model in *information retrieval* (IR), where the document most relevant to an input query is determined by ranking documents in a collection in reversed order of their similarity to the given query [4].

Overall, these various approaches become difficult to compare such various practical methods that were developed within different applications under the same framework conditions. Furthermore, researchers within one application area might not be aware of relevant methods that were developed in the context of another application. This leads to big challenge to build a clear framework that has clear generic task definitions and evaluations. Recently, there are two consecutive attempts to investigate such challenge: the *first Recognizing Textual Entailment Challenge* (15 June 2004 - 10 April 2005) [2] and the *second Recognizing Textual Entailment Challenge* (1 October 2005 - 10 April 2006) [3]. Both try to promote an abstract generic task that captures major semantic inference needs across applications. However, as in other evaluation tasks, these challenges give a new definition of textual entailment from operational view and corresponds to the judgment criteria given to the annotators who decide whether this relationship holds between a given pair of texts or not.

Recently there have been a few suggestions in the literature to regard entailment recognition for texts as an applied, empirically evaluated, task (cf. [5]). Textual entailment is also related, of course, to formal literature about logical entailment and semantic inference. It may be noted that from an applied empirical perspective, much of the effort is directed at recognizing meaning-entailing variability at the lexical and syntactic levels, rather than addressing relatively delicate logical issues. It seems that major inferences, as needed by multiple applications, can indeed be cast in terms of textual entailment.

For example, a QA system has to identify texts that entail a hypothesized answer. Given the question "What does Peugeot manufacture?", the text "Chrétien visited Peugeot's newly renovated car factory" entails the hypothesized answer form "Peugeot manufactures cars".

Similarly, for certain information retrieval queries the combination of semantic concepts and relations denoted by the query should be entailed from relevant retrieved documents. In IE entailment holds between different text-variants that express the same target relation. In multi-document summarization a redundant sentence, to be omitted from the summary, should be entailed from other sentences in the summary. And in machine translation evaluation a correct translation should be semantically equivalent to the gold standard translation, and thus both translations should entail each other. Therefore, it is expected that the textual entailment recognition is a suitable generic task for evaluating and comparing applied semantic inference models.

Eventually, such efforts promote the development of entailment recognition "engines" which provide useful generic modules across applications (see [6] [2] [3]).

3. TEXTUAL USE CASES

A use case is a technique for modeling the functional requirements of systems and systems-of-systems as perceived by outside users, called *actors*. A use case is a coherent unit of externally visible functionality provided by a system unit and expressed by sequences of messages exchanged by the system unit and one or more actors of the system unit. Each use case provides one or more scenarios that convey how the system should interact with actors to achieve a specific business goal or function. Use case actors may be end users or other systems. The purpose of the use case model is to list the actors and use cases and show which actors participate in each use case without revealing the internal structure of the system [7]. In the fact, the main motivation for using use cases is that the use case model represents the conceptual center of the software development cycle and able to drive everything that follows in the software life cycle.

In the traditional approach, natural languages, like plain English, is used for writing the use cases. A strong benefit of this approach is that use cases are comprehendible to a wide audience. A textual use case describes the possible scenarios of the future system, by showing how a particular goal is achieved by the *System under Design* (SuD) [8] and its surrounding actors in a sequence of steps. A step of a textual use case describes an action that is either a request or information passed between an actor and SuD actors, or an action internally processed by SuD (Figure 1 shows a fragment of a textual use case). These steps are forming the main success scenario specification, with extensions and variations specifying additional scenarios. Although there are slightly different methodologies to represent the use cases, the basic idea is same that a use case is usually specified as a sequence of steps forming the main success scenario specification, with extensions and variations specifying additional scenarios. In general, the structure of the textual descriptions is defined as template, providing generic guidelines as to how to specify the generated scenarios by means of main success scenario, its extensions, and variants, plus which fields should be included in the header of a use case.

In the practical approach, graphical representation of such process is easier to produce and to understand. Use case diagrams are graphical depictions of the relationships between actors and use cases and between a use case and another use case. In both cases, use cases and use case diagrams should be documents that users can interpret easily.

4. MOTIVATING EXAMPLE

We introduce the textual entailment approach to reuse software through a simple example. The example presents the use case representation for sell apartment system. This example shows a part of the dynamic phase of real estate system [9] that manages buying and/or selling the apartments. The textual use case for the proposed example is listed as follows:

Use Case Name: *Sell Apartment*

System Textual Use Case Summary:

The seller lists the Apartment, a buyer purchases the Apartment, and the agent guides them through the process and offers advice, caution, and recommendation.

Basic Course of Events:

1. The seller selects an agent.
2. The system responds by assigning an agent and notifying the seller's agent.
3. The seller lists the Apartment to sell.
4. The system responds by displaying this Apartment in the Apartment listing and linking it for searches.
5. The buyer selects an agent.
6. The buyer reviews the Apartment listing by entering search criteria.
7. The system responds by displaying properties that match the buyer's search criteria.
8. The buyer finds an Apartment and makes an offer on it.
9. The system responds by notifying the seller and the seller's agent.
10. The seller responds to the offer with a counteroffer.
11. The system responds by notifying the buyer and the buyer's agent.
12. The buyer and the seller agree to terms.
13. The system responds by recording the agreement.
14. The buyer indicates that a loan is required.
15. The system responds by locating an appropriate loan provider.
16. The buyer and the loan provider agree to loan terms.

17. The system responds by recording the terms of the loan.

18. The buyer and seller close on the Apartment.

19. The system responds by recording the details of the close.

Pre-conditions: *N/A*

Post-conditions: *N/A*

Figure 1 is the corresponding use case diagram for sell apartment example above. The use case diagram shows actors interacting with use cases. In this example, we have three actors; *buyer*, *seller*, and *advisor*. These actors interact with each others to achieve the objective of the *sell apartment* use case, which to sell an apartment. However, in our approach we will concentrate ourselves to textual use case model only.

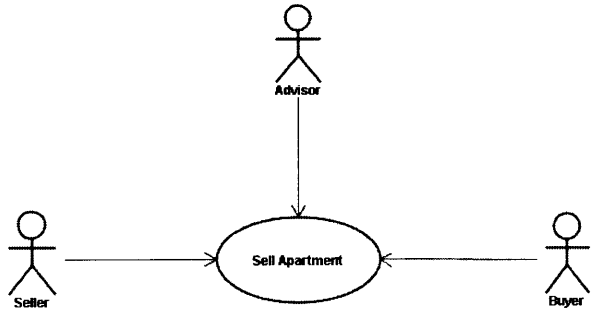

Figure 1 Sell Apartment Use Case Diagram

Suppose there is another software group wishes to model a new system for buying/selling a house. Perhaps the context use case description will be like these cases:

S1: *From available house selling list, a shopper acquires the house, and the agent helps them.*

and

S2: *The buyer looking for house in the selling list, the owner sales the house and the real state guides them.*

In both cases, we can see that they have the same semantic meaning of the previous example summery (let us gives this summery by S0). However, in the first case the meaning of S1 can be entailed from the meaning of S0. The second case is semantically equivalent to the given example S0 from the natural textual view. Here we will restrict ourself to study the first case in which the meaning of some sentence can be entailed from another sentence. However, we argue that same framework proposed here (next section) can be extending to cover such cases.

Suppose we consider T = S0 as the text and H = S1 as the Hypothesis. It clear that T entails H, this can done automatically by extracting the similar lexical words, for example, we use here the WordNet [10]; the lexical database for English language. In

our example, *buyer* and *shopper* have the same lexical mean. Also the verbs *purchases* and *acquires* and the nouns *apartment* and *house* have the same semantics (as shown in figure 2).

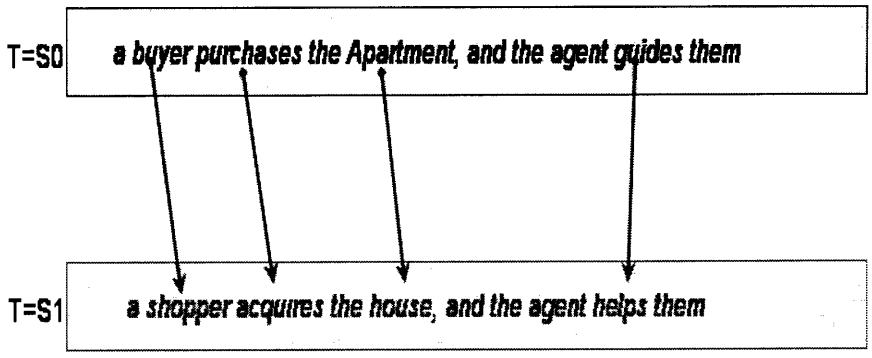

Figure 2 Textual Entailment between Two Use Case Summaries

As a result from this semantic similarity, it clears that the basic course of events for both systems will be same. Only we need to change the similar lexical entities to its equivalent in the new system. This is a simple task, which can be done automatically by using the results of extracting the similar lexical words from WordNet. However, here the example is simple for sake of simplicity; we may have that the structure of both sentences is different although the meaning is same. In that case, we have to build the parser trees [6] for both sentences, then matching the corresponding nodes in both trees to check the lexical similarity between words. Only in such approach we must pay attention to negation that may be found in one of these sentences.

This motivate us to use textual entailment to automatically checks whether the new system/subsystem can be inferred from software asset database and then generates the new system/subsystem from the old use case scenarios. As it is well known that the use case stage one of the earliest stages for software development cycle, we argue that using this approach will save a lot of efforts and will make the cost low.

5. TEXTUAL ENTAILMENT SOFTWARE REUSE FRAMEWORK

Figure 3 sketches our proposal framework to employ textual entailment approach to automatically reusable the software systems/subsystems. The framework can be separated into main two stages:

5.1 New Use Case Request Stage

New textual use case summery is submitted to the frame work. Almost any text editing tool can be used in this stage.

5.2 Lexical Textual Entailment Processor

This stage considers the kernel for our framework. The new context use case entity is checked whether it possible to entailed from the software Asset database. Depends on the result of this processor that either it is true or false, we can go further into one of these two sub-stages:

5.2.1 Scenarios Generator:

As the lexical textual entailment processor success to find that the new use case entity entails from the software asset database, it start automatically to generate the new scenarios that corresponds to the new entity.

5.2.2 Design New Use Case:

The new use case request did not find inside the software database and it is need to design manually the new scenario for it. After this done, we add it to our software asset database to enhance our asset base by new use case scenario.

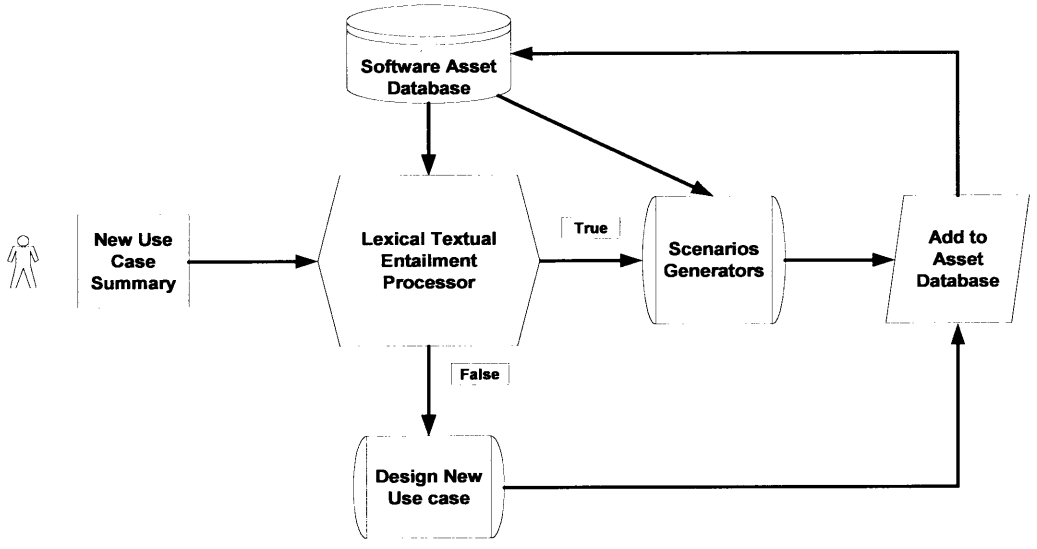

Figure 3 Textual Entailment Software Reuse Framework

6. RELATED WORK

There are some works that combine the natural language processing techniques and software development methodology. None of them has been used this new approach mentioned here. For example, Mencel [8] have proposed a way to derive a behavior specification from a textual use case by employing linguistic tools. By identifying the communication actions he able to describe how the principle attributes of the action described by a use case step can be acquired from the parser tree of the sentence specifying the step. This parse tree constructed using the proposed linguistic tool.

Osborne and MacNish [11] use natural language processing techniques to aid the development of formal descriptions from requirements expressed in controlled natural language (CL). Also, a rule-based parser is used to analyze natural language

requirements with the goal to assign Logical Form (LF) to natural language requirement specifications; focus is put on resolving parsing errors and ambiguities. In the controlled language selection, it is pointed out that a too restrictive controlled language may be irritating to use (and read), as well as hard to learn and to follow.

Rolland and Proix [12] argues that the requirements engineering process should be supported by a CASE tool based on a linguistic approach. They present a requirement engineering support environment that generates the conceptual specification from a description of the problem space provided through natural language statements. Rule-based parsing is employed to generate a conceptual schema; later, the requirements are validates by paraphrasing the original requirements – generating natural language sentences from the conceptual schema.

7. CONCLUTIONS AND FUTURE WORK

This article presented the application of the textual entailment method in a new application domain, the area of software development. More precisely the software reuse field. We consider the *systematic* software reuse. This paper introduced a novel framework that helps identify automatically the segments of software that can be reused in building new software. This framework based on a use case textual specification stage that employs in the earliest software development life cycle. Therefore we argue that using this approach will save a lot of efforts and will make the software cost low.

As future work, we intend to integrate the use of lexical textual entailment and the syntactical textual entailment to check not only the sentences that have same syntactic structure but also the sentences with different syntactic structure but have the same semantic means.

8. ACKNOWLEDGMENTS

This research is supported by JSPS (Japan Society for the Promotion of Science) and the Grant-in-Aid for JSPS Fellows.

9. REFERENCES

[1] Charles W. Krueger, *Software reuse*, ACM Computing Surveys (CSUR), v.24 n.2, p.131-183, June 1992.

[2] Dagan, I. Glickman, O. and Magnini, B. 2005. *The PASCAL Recognizing Textual Entailment Challenge*. Lecture Notes in Computer Science, Volume 3944, Jan 2006, Pages 177 - 190.

[3] Bar-Haim, R., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B. and Szpektor, I. 2006. *The Second PASCAL Recognising Textual Entailment Challenge*. In Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment.

[4] Lesk, M. E. *Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone.* In Proceedings of the SIGDOC Conference 1986, Toronto, June 1986. pp. 24-26.

[5] Corley, C., Csomai, A. and Mihalcea, R. *Text Semantic Similarity, with Applications.* In Proceedings of International Conference Recent Advances in Natural Language Processing (RANLP 2005), Borovets, Bulgaria, September 2005.

[6] J. Herrera, A. Peñas, F. Verdejo, 2006. *Textual Entailment Recognition Based on Dependency Analysis and WordNet.* In: J. Quinonero-Candela, I. Dagan, B. Magnini, F. dAlché-Buc (editors). MLCW 2005. LNAI. Springer. 3944. 231-239 (see also inside [2]).

[7] James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley Object Technology Series, Addison-Wesley Professional; 1999, ISBN: 020130998X.

[8] Mencl, V.: *Deriving Behavior Specifications from Textual Use Cases.* in Proceedings of Workshop on Intelligent Technologies for Software Engineering (WITSE04, Sep 21, 2004, part of ASE 2004), Linz, Austria, ISBN 3-85403-180-7, pp. 331-341, Oesterreichische Computer Gesellschaft, Sep 2004.

[9] Daryl Kulak, Eamonn Guiney, Erin Lavkulich. *Use Cases: Requirements in Context.* Addison-Wesley Professional; 1st edition 2000, ISBN: 0201657678.

[10] WordNet: *a lexical database for the English language.* Website: http://wordnet.princeton.edu/

[11] Osborne, M., MacNish, C. K.: *Processing Natural Language Software Requirement Specifications.* In Proceedings of ICRE 1996, pp. 229-237, April 15 - 18, 1996, Colorado Springs, Colorado, USA. IEEE Computer Society.

[12] Rolland, C., Proix, C.: *A Natural Language Approach for Requirements Engineering.* In Proceedings of CAiSE'92, Manchester, UK, May 12-15, 1992, LNCS 593, Springer 1992.