Modeling Software Specifications with Case Based Reasoning

Nuno Seco[1], Paulo Gomes[2], and Francisco C. Pereira[2]

[1] Department of Computer Science, University College Dublin,
Dublin 4, Ireland

[2] CISUC - Centro de Informática e Sistemas da Universidade de Coimbra,
Departamento de Engenharia Informática, Universidade de Coimbra,
3030 Coimbra, Portugal

Abstract. Helping software designers in their task implies the development of tools with intelligent reasoning capabilities. One such capability is the integration of Natural Language Processing (NLP) in Computer Aided Software Engineering (CASE) tools, thus improving the designer/tool interface. In this paper, we present a Case Based Reasoning (CBR) approach that enables the generation of Unified Modeling Language (UML) class diagrams from natural language text. We describe the natural language translation module and provide an overview of the tool in which it is integrated. Experimental results evaluating the retrieval and adaptation mechanisms are also presented.

1 Introduction

Design is a complex task involving several types of reasoning mechanisms and knowledge types [1]. The field of software design is no exception, especially if we consider software design as a way of computationally modeling part of the real world. CASE tools were developed some decades ago to help designers model software systems. Many of these tools are only graphical editors that graphically assist the user designing the system. Given this fact, these tools are bereft of capabilities other than simple editing skills, one can suggest that the inclusion of intelligent reasoning capabilities is a goal that must be attained in order to relieve the designer from tedious and time consuming tasks.

We are developing a CASE tool with several functions that extend beyond these mundane editing skills. One such function is the ability to translate textual descriptions of a proposed system into the graphical formalism used by the CASE tool. This process implies the three following steps: morphological analysis, syntactic analysis, and semantic analysis. The morphological analysis identifies the lexical category of each word. The syntactic analysis creates the parse tree of the text being analyzed. Finally, semantic analysis associates meaning to text elements and converts them into the formalism used in our CASE tool. For the initial steps, there are well established methods to perform these analysis stemming from the NLP community [2, 3]. The semantic analysis is domain dependent and requires thoroughly developed approaches to cope with

Seco N., Gomes P. and C. Pereira F. (2004).
Modeling Software Specifications with Case Based Reasoning.
In *Proceedings of the 1st International Workshop on Natural Language Understanding and Cognitive Science*, pages 135-144
DOI: 10.5220/0002670701350144

the complexity of natural language. In this paper, we propose a new approach to semantic analysis of texts and analyze means of converting them into a formal software specification language. The proposed approach for semantic analysis is based on the CBR paradigm [4, 5] and converts textual software descriptions into UML class diagrams[3].

The immense number of syntactical and semantic relations which are possible in natural language jeopardizes the development of a theoretical framework for translation. The lack of an aligned corpus containing software requirements and their respective software designs, precludes the possibility of using corpus based strategies, such as Example Based Machine Translation (EBMT). The above reasons lead us to believe that CBR is a suitable approach. Another advantage of a CBR approach, is that it allows the system to evolve in time by learning new cases and by adapting itself to the linguistic modeling preferences of its users. We have dubbed our NLP module NOESIS, which is an acronym for **N**atural Language **O**riented **E**ngineering **S**ystem for **I**nteractive **S**pecifications.

The next section describes the CASE tool where NOESIS is integrated. The NOESIS module is detailed in section 3 where we focus on the NLP steps of the translation process, the case representation, retrieval and finally the adaptation mechanisms. Section 4 presents experimental work. Finally, we bring our analysis to a conclusion and offer some ideas for future work.

2 REBUILDER

The main goals of REBUILDER are to create and manage a repository of software designs and to provide the software designer with a set of functions which are necessary in promoting the reuse of previous design experiences.

Figure 1 illustrates the architecture of REBUILDER. It comprises of four main modules: the UML editor, the knowledge base manager, the knowledge base (KB), and the CBR engine. It also depicts the two different user types: software designers and KB administrators. Software designers use REBUILDER as a CASE tool and subsequently reuse the software design knowledge it has previously stored. The KB administrator keeps the KB updated and consistent. The UML editor serves as the intermediary between REBUILDER and the software designer while the KB manager is the interface between the KB administrator and the system.

3 NOESIS

The aim of NOESIS is to assist the designer in the creation of the initial UML class diagram. Consequently, the diagram may be passed along to the other reasoning modules capable of completing it. At the present stage, NOESIS focuses on the structural requirements of the desired system salient in the software specification documents. A possible passage could be:

[3] UML is the design language used by the CASE tool that we are developing (see [6])

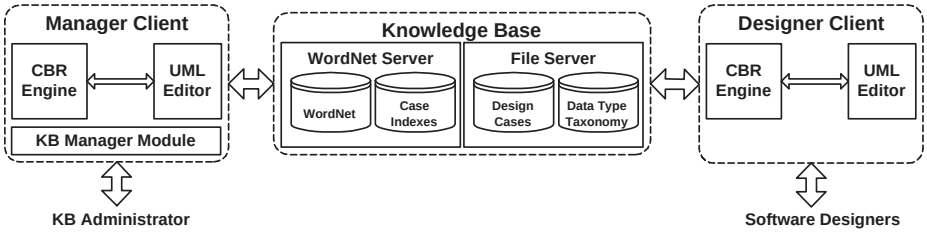

Fig. 1. REBUILDER's Architecture.

A BankingCompany has many Customers. These Customers may have several BankAccounts. A Customer can request a BankCard for each BankAccount.

These texts are subject to some peculiarities and simplifications [7], which somewhat eases the difficulty in understanding them. Of these peculiarities, **explicitness** is probably one of the most significant. It is useful in reducing to a minimum, the amount of unexpressed information in the specifications, thus avoiding potential disputes over what the software should or should not do.

NOESIS, similar to any other NLP system, is aware of the knowledge categories identified in [2] which include Phonetics, Phonology, Morphology, Syntax, Semantics, Pragmatics and Discourse. However, due to the specific nature of the problem, phonetic and phonological knowledge is absent. We assume, for the moment, that the requirements are fed to NOESIS in the form of digitalized texts. The other omitted knowledge category is pragmatic knowledge. As stated above, we assume that these texts are explicit. Therefore, implicit information is avoided, thus relieving the system of dealing with pragmatic issues.

Figure 2 illustrates the main modules that are elements of the NLP engine of NOESIS and the processes that each sentence undergoes. Sentences from the original text are queued and posted individually to the NLP engine. Each word in the sentence is tagged with a label that identifies the grammatical class of that word. The tagged sentence is then sent to a syntactic parser which derives the possible parse trees of the sentence. One of these trees is then selected as the best derivation and is passed on to the *case-based semantic analyzer* which outputs a meaningful representation (e.g. UML Class Diagram). This representation is a diagram that corresponds to the sentence, yet it is only a partial solution to the initial text. As a result, it is necessary to merge these partial diagrams together before being presented as a candidate solution to the user. The use of sentences as the basic unit of processing is arguable. Obviously, by breaking the text into sentences we loose implicit paragraph-level or text-level relations, but on the other hand, we are also able to make more detailed similarity judgments. Nevertheless, as will be stated in section 3.2, during semantic analysis we capture these coarser grained relations (text-level) by including a context and discourse evaluation component in our similarity metric.

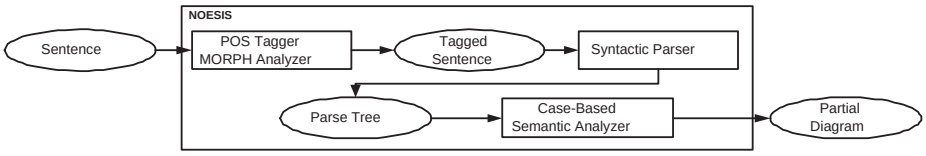

Fig. 2. NOESIS's Architecture.

3.1 Morphological and Syntactic Analyzer

The first phase of processing is Part-Of-Speech tagging. The tagger used in NOESIS is a purely stochastic tagger called QTAG[4]. The tagset used by NOESIS is a simple and reduced form of the original tagset. If the sentence is successfully tagged and all nouns and verbs exist in the WordNet lexicon, then the tagged sentence is passed on to the next analyzer. Otherwise, the sentence will undergo morphological processing in order to identify inflected words. These inflections are removed from the sentence and the tagger is asked to process the sentence again. If the sentence is still not correctly tagged then the user may tag the problematic word(s) manually or rephrase the specification. It should be noted that the reduced tagset that we are using is not fine grained, so distinction between plural and singular forms, which would be an obvious indicator of cardinality of the relations, is not possible at this point.

The objective of the syntactic analyzer is to discover all valid syntactic parse trees for a given sentence. A very simple context free grammar for the English language is used. The parsing algorithm used is known as the Earley algorithm [8]. After parsing, the resulting derivation is passed on to the semantic analyzer which produces the corresponding meaning representation.

3.2 Semantic Analyzer

The semantic analyzer receives a syntactic parse tree as its input and produces an UML class diagram that partially models the requirement text. Our analyzer uses a CBR mechanism to produce a plausible diagram.

The Case Base and Indexes Previous requirements are stored and indexed as cases in the case base. Cases in NOESIS are composed of a single parse tree (the leaves are the words of the sentence), a corresponding class diagram and a mapping tuple that establishes the coupling between nouns in the sentence and the entities in the diagram. These cases are then indexed according to the verbs present in the sentence. Each verb is attributed a synset[5] from WordNet, then an index representing the case is created and attached to the corresponding synset node in WordNet. The use of verbs for indexing is twofold; firstly verbs provide a relational and semantic framework for sentences [9], therefore, the verb occupies a core position and no valid sentence may exist without a

[4] More information can be obtained at http://www.clg.bham.ac.uk/tagger.html
[5] A synset identifies the intended meaning for a word.

verb. The second reason is concerned with performance reasons. There are many more nouns than verbs in the English language [9], which means that searching in WordNet's noun graph is much more demanding than searching in it's verb graph. The most relevant types of relations used to connect noun synsets are hypernym and meronym relations. Regarding verb synsets, meronym relations do not exist. These were replaced by entailment relations (specific verb clustering relations were also added) [9].

Attaching indexes to WordNet helps evaluate semantic similarity, but portrays nothing regarding syntax which is also a very important facet of sentence interpretation. In order to close this gap we have created a second structure that is also used for indexation. This means that every index is simultaneously attached to these two different structures. Figure 3illustrates how this syntactic indexation tree is organized. The root of the tree has no syntactic meaning. It simply exists for the sake of simplifying the algorithms that manipulate these data structures.

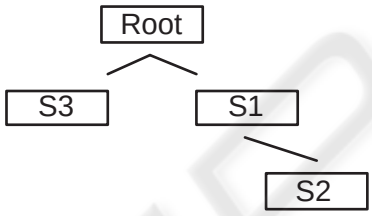

Fig. 3. Syntactic index tree.

Imagine that the nodes *s1* and *s2* correspond to the parse trees in figure 4, these trees could represent the syntactic derivation of the sentences *"Book that flight."* and *"John ate the pizza."*, respectively. As can be seen, each node of the tree in figure 3 syntactically subsumes every other descending node. This kind of structure facilitates the assessment of syntactic similarity. We assert that *s1* subsumes *s2* when every branch of *s1* is contained in *s2*. This indexation tree is dynamically rearranged when new cases are added to the case base.

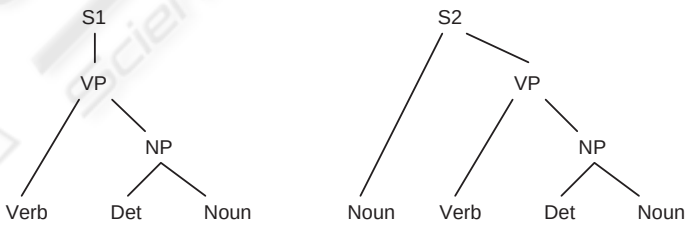

Fig. 4. Example parse trees.

Retrieval Our retrieval algorithm comprises of two steps. Firstly, we look for relevant candidates through the use of indexes and then select the most promising candidate that maximizes our similarity metric.

The first substep takes advantage of the implemented indexing scheme where both WordNet and the syntactic index tree can be used. This flexibility enabled the implementation of several algorithms that use both structures simultaneously or individually (giving preference either to semantics or syntax).

Four retrieval algorithms were implemented which combined the use of these structures. The algorithms are all very similar to the one encountered in [6], except that all relations between verb synsets are used during the search. The algorithm is based on spreading activation, which expands the search to neighbor nodes until some condition is attained and the search terminates. All algorithms commence by finding a list of entry points into to the relevant structure(s). The verbs' synset is used as the entry point into WordNet. Regarding the syntactic index tree, the entry point is the node representing the same syntactic derivation as our target sentence. If no such node exists then a list of entry points is created which contains the nodes that directly subsume and are subsumed by our target sentence. The four algorithms used by NOESIS are:

1. **Semantic Retrieval** — Only uses the WordNet verb graph.
2. **Syntactic Retrieval** — Only uses the syntactic index tree.
3. **Semantic Filtering Retrieval** — Uses the syntactic index tree but eliminates indexes that have a semantic distance above some user defined threshold.
4. **Conjunctive Retrieval** — Uses both structures simultaneously. Only intersecting indexes that are reached during the search on both structures are considered valid and may be returned.

Similarity Assessment The second substep ranks each of the indexes returned by the retrieval algorithm. These indexes contain the parse tree for the case they represent which is then used for similarity assessment. We use four different measures for ranking indexes:

– Syntactic Similarity is given by:

$$\frac{1}{2} \times \left(\frac{\#intersect(target, source)}{\#nodes(target)} + \frac{\#intersect(target, source)}{\#nodes(source)}\right) \quad (1)$$

where *intersect* computes a list of nodes that are syntactically common in both parse trees and *nodes* returns a list of nodes contained in the tree. Since there is no obvious reason to normalize the intersection in relation to either the $source$ or the $target$, we consider both with equal importance.

– Semantic Similarity is given by:

$$\frac{dist(n(target), n(source)) + dist(v(target), v(source))}{2 \times MSL} \quad (2)$$

where *dist* computes the average semantic distance between noun synsets or verb synsets. *n* and *v* return a list of the nouns or verbs contained in the parse tree. MSL (Maximum Search Length) defines the maximum number of arcs that the search may transverse. If the distance can not be computed the value of the MSL is returned.

– Contextual Similarity is given by:

$$\begin{cases} 1 & \text{if } source \text{ belongs to a text from which a sentence was previously used} \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

– Discourse Similarity is given by:

$$\frac{1}{(snum(target) - snum(source))^2 + 1} \quad (4)$$

where *snum* returns the absolute position of the sentence in the original text. Hence, we try to match sentences that occur in similar positions in both texts, allowing us to capture text-level relations.

These four formulas are normalized which will produce values between $[0..1]$. After the computation of each of these components, a weighted sum is used to evaluate the overall similarity between the target and the source. The indexes are then ranked and the index that maximizes expression (6) is considered for reuse.

$$sim(target, source) = \omega_1 \cdot SyntacticSimilarity + \omega_2 \cdot SemanticSimilarity + \omega_3 \cdot ContextSimilarity + \omega_4 \cdot DiscourseSimilarity \quad (5)$$

Adaptation After all sentences of the target text have been mapped with a similar source sentence the adaptation phase begins. Adaptation starts by loading the diagrams specified by the selected indexes into memory. Then the names of the entities from the source diagrams are replaced with the nouns encountered in the target sentence. This replacement of names is possible because each case holds the mapping (see section 3.2) between the nouns in the sentence and the entities of the diagram. This mapping is denoted by (6).

$$\Upsilon : Source_{sentence} \longrightarrow Source_{diagram} \quad (6)$$

Having defined the correspondence function between nouns and entities, we now need a function that can map nouns from the target sentence with nouns apparent in the source sentence. A naive mechanism is used to accomplish this task; basically the isomorphism between nouns is established on the basis of their relative positions in each sentence. This means that the first noun in one sentence will be isomorphic with the first noun of the second sentence and so on, so forth. The correspondence is denoted by (7).

$$\Gamma : Target_{sentence} \longrightarrow Source_{sentence} \quad (7)$$

Obtaining a diagram for the target sentence is now a matter of discovering the mappings between $Target_{sentence}$ and $Source_{diagram}$. This is easily accomplished by applying (8) and replacing the names of the entities in the diagram with the nouns in the target sentence.

$$\Upsilon \circ \Gamma : Target_{sentence} \longrightarrow Source_{diagram} \quad (8)$$

Applying the above process will yield a several diagrams; there will be as many diagrams as target sentences. It frequently occurs that each diagram has entities with the same name (some simple inflections may exist). These entities are merged resulting in a single diagram that is presented to the user.

Retain The proposed diagram may be adapted by the user in the way he or she thinks adequate. If many modifications are made on the suggested diagram, it may be due to a lack of coverage of the case base. In other words, the case base fails to hold the necessary knowledge to satisfactorily solve the problem. Confronted with such a situation, it is reasonable for the user to submit the altered case to the case base. Then, it is up to the KB administrator to decide if the submitted case should be indexed and made accessible to the retrieval component.

4 Preliminary Experimental Studies

In this section, we present some results of preliminary experiments. The aim of these experiments is to compare the output of NOESIS's solution with that of a software designer. We are vividly aware of the subjectivity involved in these experiments and hold the view that the only reliable experiment to assess the proposed strategy, is one where the tool is used in a real software production environment and is evaluated by its users.

The results presented here are based on a case base comprising of 62 cases, which corresponds to 12 different texts (an average of 4 sentences per text). A set of 22 problems (an average of 3 sentence per problem) were used for testing the overall accuracy of the system.

For each problem text a corresponding UML class diagram was designed by the authors. The same problems were given to NOESIS and the two solutions were compared using a class diagram comparison metric yielding a number ranging from 0 to 1, where 1 represents an identical match and 0 represents two completely different diagrams. This metric is actually part of a larger evaluation component which computes the similarity between two packages (each package may contain a diagram and several other packages. A detailed explanation can be found in [10].

The four retrieval algorithms presented in section 3.2 along with different weighting schemes were combined resulting in 60 distinct configurations. The weight combinations used are evident in table 1. The Semantic Filtering Retrieval algorithm was used with a predefined threshold of 5 (see section 3.2). Each problem was solved using each of the configurations presented in table 1.

Figure 5 purveys the average similarity of NOESIS's diagrams in relation to the human generated solutions for every problem description in each of the 15 configurations. In figure 5, SemR, SynR, SemFR and ConR correspond to Semantic Retrieval, Syntactic Retrieval, Semantic Filtering Retrieval and Conjunctive Retrieval, respectively.

ω	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11	C12	C13	C14	C15
ω_1	1	0	0	0	0.5	0.5	0.5	0	0	0	0.33	0.33	0.33	0	0.25
ω_2	0	1	0	0	0.5	0	0	0.5	0.5	0	0.33	0.33	0	0.33	0.25
ω_3	0	0	1	0	0	0.5	0	0.5	0	0.5	0.33	0	0.33	0.33	0.25
ω_4	0	0	0	1	0	0	0.5	0	0.5	0.5	0	0.33	0.33	0.33	0.25

Table 1. The fifteen configurations used for experimental evaluation. The ω's represent the different weights given to the overall similarity metric presented in section 3.2.

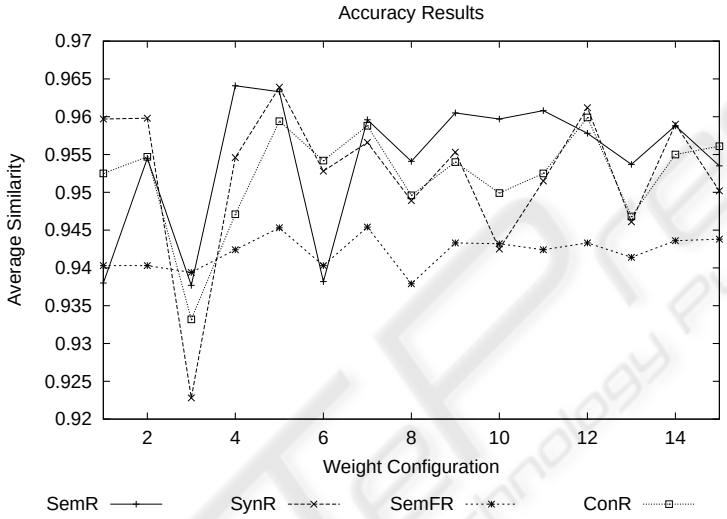

Fig. 5. Average similarity between NOESIS's solutions and human generated solutions.

These results show us that configuration C03 (where only context is used for similarity assessment) has inferior similarity for all four retrieval algorithms. The Semantic Filtering Retrieval algorithm performs worse than all other algorithms. This may be a result of the low threshold value used. This plot also indicates that the adaptation mechanism, independent of the retrieval algorithm, is able to adapt the retrieved cases to the target situation.

5 Conclusions and Future Work

This paper presents a case-based reasoning system capable of translating textual software descriptions into UML class diagrams. One advantage of using CBR to perform semantic analysis of sentences is that CBR does not need a domain theory in which to work. In software design, it is very difficult to come up with a domain theory because the designer is modeling an aspect (or a view) of the real world. It is impractical to cope with all possible views. Another advantage of CBR is that it retains new knowledge in

the form of cases. This not only allows the system to evolve but also makes it possible for the system to adapt itself to the designers' modeling preferences.

The experimental results shown in section 4 are not conclusive but they encourage further research using CBR as the main reasoning mechanism for translation of software descriptions into UML class diagrams. This work has raised several issues that require further research and improvement. One of these issues is the extension of the system to deal with attributes and methods. The linguistic processing mechanism would also improve if issues like, selecting the correct syntactic derivation, the resolution of anaphora and the detection of compound nouns were solved internally by NOESIS. The use of a complete tagset should also allow the system to make better similarity judgments.

References

1. Tong, C., Sriram, D.: Artificial Intelligence in Engineering Design. Volume I. Academic Press (1992)
2. Daniel Jurafsky, J.H.M.: Speech and Language Processing. Prentice Hall (2000)
3. Manning, C., Schütze, H. In: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, US (1999)
4. Aamodt, A., Plaza, E.: Case–based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications **7** (1994) 39–59
5. Kolodner, J.: Case-Based Reasoning. Morgan Kaufmann, San Mateo, California (1993)
6. Gomes, P., Pereira, F.C., Paiva, P., Seco, N., Carreiro, P., Ferreira, J.L., Bento, C.: Case retrieval of software designs using wordnet. In Harmelen, F.v., ed.: European Conference on Artificial Intelligence (ECAI'02), Lyon, France, IOS Press, Amsterdam (2002)
7. Ambriola, V., Gervasi, V.: Processing natural language requirements. In: Proc. of the 12th International Conference on Automated Software Engineering, Los Alamitos, IEEE Computer Society Press (1997) 36–45
8. Earley, J.: An efficient context-free parsing algorithm. Communications of the ACM **13** (1970) 94–102
9. Miller, G., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.J.: Introduction to wordnet: an on-line lexical database. International Journal of Lexicography **3** (1990) 235 – 244
10. Gomes, P., Pereira, F.C., Paiva, P., Seco, N., Carreiro, P., Ferreira, J.L., Bento, C.: Experiments on case-based retrieval of software designs. In Craw, S., Preece, A.D., eds.: 6th European Conference on Case-Based Reasoning (ECCBR'02). Volume 2416., Aberdeen, Scotland, UK, Springer (2002)