On the Identification of Goals in Stakeholders Dialogs

Leonid Kof

Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748 Garching bei München, Germany
kof@informatik.tu-muenchen.de

Abstract. Contradictions in requirements are inevitable in early project stages. To resolve these contradictions, it is necessary to know the rationale (goals) that lead to the particular requirements. In early project stages one stakeholder rarely knows the goals of the others. Sometimes the stakeholders cannot explicitly state even their own goals. Thus, the goals have to be elaborated in the process of requirements elicitation and negotiation.
This paper shows how the goals can be derived by systematic analysis of stakeholders dialogs. The derived goals have to be presented to the stakeholders for validation. Then, when the goals are explicitly stated and validated, it becomes easier to resolve requirements contradictions.

1 Introduction

When a complex system is developed, there exist many different stakeholders or stakeholder groups whose interests should be taken into account. For example, if we build a drive-by-wire system, the obvious stakeholders would be the car manufacturer itself (OEM, original equipment manufacturer), prospective drivers, service staff, and the legislator. Every stakeholder has his own goals. These goals can be conflicting. For example, one of the OEM's goals may be cost reduction. Cost reduction can be achieved, for example, by reducing the brake system to the rear wheels only. This would conflict with the legislator's goal to provide road safety.

The above example of goal conflict should be obviously resolved before the goals are refined. In the case that the goal conflicts are less apparent, the goals could be refined to finer subgoals, before the conflict becomes apparent. For example, the legislator's goal to provide road safety does not conflict with the road maintainer's goal to minimize wearing down of the road surface by the cars. However, reasonable refinements of these goals can become conflicting: "provide road safety" can be refined to "use caterpillars instead of wheels", whereas "minimize wearing down of the road surface" can be refined to "prohibit caterpillars". This conflict cannot be resolved, unless we retreat to the original goals and look for alternative goal refinements.

The problem of conflicting goal refinements is not really a problem, as long as every stakeholder can explicitly state his top-level goals. The normal project situation is, however, that stakeholders themselves have rather vague ideas about their own goals. In this case they can intuitively identify requirements that are problematic (i.e., conflicting with their goals). Conflict resolution, however, results in looking for a requirements set

satisfying every stakeholder. This can be a rather tedious business, particularly when goals are not explicitly specified.

To facilitate the whole requirements engineering process, it is important to identify the stakeholder goals as early as possible. This paper discusses possibilities of goal identification on the basis of stakeholders' dialogues transcripts.

The remainder of the paper has 6 sections. Section 2 introduces the case study used to illustrate the approach. Section 3 gives an overview of goal-oriented requirements engineering, including rules of thumb to identify goals. Section 4 shows how the goals can be manually identified in the case study. Section 5 gives an overview of available approaches to natural language processing (NLP) and their applications to requirements engineering. Here, the idea is to use NLP for goal identification. Then, Section 6 shows how goal identification could be automated. Finally, Section 7 summarizes the whole paper.

2 Case Study

The procedure for goal identification, presented in this paper, is evaluated on a small case study on an airport screening system. The case study is just a two-page document, representing an online stakeholder discussion [1]. This document does not contain any explicitly stated requirements. To give a flavor of the document, Table 1 presents the first three paragraphs of the document.

There are three stakeholders participating in the discussion: a representative of the Transportation Security Administration (TSA), a representative of the Federal Aviation Administration (FAA), and a representative of the airport screening and security staff. The case study represents a rather intense discussion, where none of the stakeholders explicitly states his goals. They all agree on the goal that air traffic security should be improved, but they see different problems and propose different solutions to the common goal. Altogether, each writes just 4-5 paragraphs, which is surely not enough to identify all requirements. However, the statements of every stakeholder are motivated by his goals, which makes the case study a good example to demonstrate goal extraction.

Table 1. Stakeholders' dialogue, excerpt

Federal Aviation Administration: We have to ban on airplane passengers taking liquids on board in order to increase security following the recent foiled United Kingdom terrorist plot. We are also working on technologies to screen for chemicals in liquids, backscatter, you know...
Airport Screening and Security: Technologies that could help might work well in a lab, but when you use it dozens of times daily screening everything from squeeze cheese to Channel No. 5 [sic] you get False Alarms... so it is not quite ready for deployment!
Federal Aviation Administration: Come on! Generating false positives helped us stay alive; maybe that wasn't a lion that your ancestor saw, but it was better to be safe than sorry. Anyway, I want you to be more alert - airport screeners routinely miss guns and knives packed in carry-on luggage.

3 Goal-Oriented Requirements Engineering

Software development involves different stakeholders, and conflicts among stakeholders are common. To resolve the conflicts, it is vital to know not only the position of every stakeholder, but also the rationale for the position, originating from some goal. This idea is the basis for the Win-Win negotiation approach [2].

A *goal* in requirements engineering is "an objective the system under consideration should achieve" [3]. In order to satisfy some goals, cooperation of several active components, or *agents* can be required. For example, to achieve the goal "safe air transportation" it is necessary that the administrative authorities and the airport screening staff cooperate.

Goals can be refined to subgoals in two ways. There exist AND and OR refinements. If some goal is AND-refined to a set of subgoals, it is necessary to satisfy all the subgoals to satisfy the original goal. For example, the goal "safe air transportation" can be AND-refined to the goals "proper aircraft maintenance" and "no terrorists on board", which have both to be satisfied in oder to achieve "safe air transportation". If some goal is OR-refined to a set of subgoals, it is sufficient to satisfy one of the subgoals to satisfy the original goal. For example, the goal "no explosives in carry-on luggage" can be OR-refined to "do not allow any carry-on luggage" and "screen carry-on luggage".

To identify goals, two key questions can be applied: "WHY" and "HOW". An answer to a "HOW"-question for a goal gives a possible refinement of the goal. An answer to a "WHY"-question for a goal identifies its superior goals. For example, if we ask "WHY" the goal "screen carry-on luggage", we get that "there be no explosives in carry-on luggage" and, perhaps, that "there be no sharp items in carry-on luggage" and that "there be no liquids in carry-on luggage".

Apart form asking "WHY"- and "HOW" questions, there are two further way to identify goals:

- List the problems of the existing system. The negation of every problem becomes a goal of the system to be built.
- Look for goal-indicating expressions in the requirements document, like "purpose", "objective", "concern", "intent", "in order to", etc.

Van Lamsweerde provides a much more thorough introduction to goal-oriented requirements engineering [3].

4 Case Study: Manual Goal Identification

In the ideal world, every stakeholder could explicitly state his goals and identify contradictions to other stakeholders' goals. The small case study, treated in this paper, shows that this is not the case in the real world. In the stakeholders' dialog, the goals are mostly implicit, they manifest themselves in proposals that a stakeholder makes and in objections to the proposals made by others. For example, in the case study the FAA officer opens the discussion with the statement that "We have to ban on airplane passengers taking liquids on board *in order to increase security following the recent foiled United Kingdom terrorist plot*." In this sentence, a goal is explicitly stated, introduced by the

phrase “in order to”. The reaction to this statement shows the goal of the airport screening staff, rather indirectly: “Technologies that could help might work well in a lab, but when you use it dozens of times daily screening everything from squeeze cheese to Channel No. 5 [sic] you get False Alarms... *so it is not quite ready for deployment*!” The actual goal is the application of screening techniques in day-to-day operation, not distinguishing squeeze cheese from explosives.

In the case study, we can identify the goals by asking the question for each statement, why the statement was made by its uttering stakeholder. In this way, we can identify the following goals of the stakeholders:

- Goals of the Federal Aviation Administration:
  improvement of security: “We have to ban on airplane passengers taking liquids on board in order to increase security following the recent foiled United Kingdom terrorist plot”
  effectiveness: “We are trying to federalize checkpoints and to bring in more manpower and technology”
- Goals of the Transportation Security Administration:
  improvement of security:
    pro-active thinking: “We have yet to take a significant pro-active step in preventing another attack everything to this point has been reactive”
    consistency in regulations: “I think that enforcing consistency in our regulations and especially in their application will be a good thing to do”
- Goals of the airport screening and security staff:
  application of the rules in everyday operation: “Technologies that could help might work well in a lab, . . . , so it is not quite ready for deployment”, “It’s not easy to move 2 million passengers through U.S. airports daily”
  cost effectiveness for the airlines: “I mean an economic threat is also a threat”
  consistency in rules: “There are constant changes in screening rules - liquids/no liquids/3-1-1 rule”

These goals are not contradiction-free. By analyzing the document, it is possible to identify following contradictions:

- proactive thinking, which is a TSA goal, vs. cost effectiveness, which is an FAA goal. Actually, this is not necessarily a contradiction, but it sounds like a contradiction in the dialog.
- responsibility for the security checks: airlines become responsible, which is an FAA goal, vs. the authority currently performing the checks remains responsible.
- acceptability of false positives: acceptable for FAA, not acceptable for the screening staff

Probably due to the fact that each stakeholder considers his own goals as obvious, no one ever explicitly states them. Instead, each stakeholder presents solutions that seem adequate to him and explains why he thinks the solutions proposed by others are problematic. This observation about indirect goal statements will be used in Section 6 in order to systematize and potentially automate the identification of goals.

5 Natural Language Processing in Requirements Engineering

Traditionally, natural language processing is considered as taking place at four layers: lexical, syntactic, semantic, and pragmatic. Analysis tasks and result types for every kind of analysis are sketched in Table 2.

Table 2. Classification of text analysis techniques

Approach type	**Analysis tasks**	**Analysis results**
lexical	identify and validate the terms	set of terms used in the text
syntactic	identify and classify terms, build and validate a domain model	set of terms used in the text and a model of the system described in the text
semantic	build a semantic representation of every sentence	logical representation of every sentence, formulae
pragmatic	build a representation of the text, including links between sentences	logical representation of the whole text, formulae

For all layers except pragmatic there exist analysis techniques, either potentially automatable or already automated. Lexical techniques are the simplest. They consider each sentence as a character or word sequence, without taking further sentence structure into account. Due to this simplicity lexical techniques are extremely robust. The flip side of this robustness is that lexical methods are limited to pure term extraction. Syntactic approaches, as opposed to lexical ones, take also sentence structure into consideration. Based on this sentence structure, they extract not only the terminology, but also some domain model. Semantic approaches achieve more than the previous two classes: they produce a formal representation of the text. It is mostly a kind of first order predicate logic, but the concrete representation may differ. This task is surely very demanding, which poses severe limitations on the text for the approaches to work. As for pragmatics analysis, there is no automated procedure yet. There exist, however, a logic capable of capturing pragmatics-motivated relations between sentences.

The remainder of this section describes different kinds of text analysis approaches in more detail: Section 5.1 introduces the lexical approaches, Section 5.2 introduces the syntactic approaches and Section 5.3 introduces the semantic approaches. Section 5.4 presents the logic to capture pragmatic relations between sentences. Finally, Section 5.5 discusses the applicability of different kinds of analysis to goal identification.

5.1 Lexical Approaches: Analyzing the Document Vocabulary

The goal of lexical approaches is to identify concepts used in the requirements document. They do not classify the identified terms or build a domain model. The common feature of these techniques is that they analyze the document as only a sequence of characters or words. Berry [4] lists several approaches applying lexical techniques to requirements engineering. To give the flavor of lexical approaches, the following will

be considered here: AbstFinder by Goldin and Berry [5], lexical affinities by Maarek and Berry [6] and documents comparison by Lecoeuche [7].

AbstFinder [5] works in the following way: it considers each sentence simply as a character sequence. Such character sequences are compared pairwise to find common subsequences. These subsequences are assumed to be potential domain concepts to be approved by the user. For example, consider two sentences taken from the steam boiler case study [8]:

> The steam-boiler is characterized by the following elements:

and

> Above m2 the steam-boiler would be in danger after five seconds, if the pumps continued to supply the steam-boiler with water without possibility to evacuate the steam.

The first sentence is shorter and it is augmented with spaces before the start of the search for common character subsequences. Then one of the sentences is rotated character-wise and for each rotated position AbstFinder controls whether there are aligned common subsequences. Rotation of the sentences is necessary to identify character chunks placed differently, like "flight" and "book" from "The flights are booked" and "He is booking a flight". (This example is taken from the AbstFinder article [5].) Such analysis is performed for all sentence pairs.

For the steam boiler example introduced above, the aligned position would look like

```
         The steam-boiler is characterized by...
Above m2 the steam-boiler would be in danger...
```

In this case AbstFinder would identify "the steam-boiler" as a concept contained in the document.

However, when considering two other sentences from the steam boiler specification, like

> Below m1 the steam-boiler would be in danger after five seconds, if the steam continued to come out at its maximum quantity without supply of water from the pumps

and

> Above m2 the steam-boiler would be in danger after five seconds, if the pumps continued to supply the steam-boiler with water without possibility to evacuate the steam

AbstFinder would identify "the steam-boiler would be in danger after five seconds, if the" as a common concept. This is not a concept that can be used to model the application domain. To decide which extracted sequences of characters really represent application-specific concepts, human analyst has to approve the extracted concepts.

The approach by Maarek [6] identifies concepts as word pairs where the appearances of these two words in the same sentence correlate. For example, "steam" and

"boiler" often co-occur in the steam boiler specification [8], so this approach would identify "steam boiler" as an application concept.

Both Goldin and Berry and Maarek assume that the most important terms can be identified as the most frequent ones. Thus, they would miss an important term that is used only once, e.g. in the title or in the once-mentioned explanation of an acronym.

The approach by Lecoeuche [7] is more selective, in the sense that it not only extracts concepts, but also measures their importance and neglects concepts whose importance does not reach the manually set threshold. The approach compares the frequency of the concept in the analyzed document with the frequency of the same concept in some baseline document. Let F_a be the number of occurrences of some term in the analyzed document and F_b the number of occurrences of the same concept in the baseline document. Then, the importance measure of a concept is defined as $imp = \frac{F_a}{F_a+F_b}$. High importance measure can imply that the concept is mentioned just few times in the baseline document (for example in the definitions), but is mentioned many times in the analyzed document. Concepts with a high importance measure are identified as application domain concepts.

Sawyer et al. [9] apply a similar idea to identify application domain concepts. The difference lies in the definition of the baseline documents: For the Lecoeuche's approach, the baseline document has to be provided by the user, whereas Sawyer et al. compare term frequency in the analyzed document with the term frequency in everyday usage. A term whose frequency in the analyzed document significantly differs from the frequency in everyday usage is considered as an important application domain concept.

It is easy to use lexical analysis to identify many potential goals. Van Lamsweerde suggests in [3], for example, to identify potential goals in requirements documents by means of certain key phrases, like "purpose", "objective", "concern", "intent", "in order to", etc. This technique can be used in our case study as well (cf. Section 6).

5.2 Syntactic Approaches: Identifying Terms and Relations

Syntactic approaches, presented in this section, promise more than pure vocabulary analysis. These approaches became widely known in the field of object-oriented analysis, as they allow for easy mapping of extracted concepts to classes, objects, attributes and methods. Some of these approaches do not offer any automation in their original versions, but they could be partially automated using linguistic techniques available now. Complete automation is still not possible, both due to low precision of the available tools and due to necessity to adapt the tools to every concrete document to analyze.

One of the first approaches aiming at analysis of specification texts is the one by Abbott [10]. The goal of Abbot's approach is to

> "... identify the data types, objects, operators and control structures by looking at the English words and phrases in the informal strategy"

Abbott takes the following types of words and phrases into consideration during model building:

- common nouns

– proper nouns and other forms of direct reference
– verbs and attributes

These word types are used in the following way during model building:[1]

1. A common noun in the informal strategy suggests a data type.
2. A proper noun or a direct reference suggests an object.
3. A verb, predicate or descriptive expression suggests an operator.
4. The control structures are implied in a straightforward way by the English.

This strategy works in the following way: given the specification text like

> If the two given DATEs are in the same MONTH, the NUMBER_OF_DAYS between them is the difference between their DAYs of MONTH,

Abbott identifies the common nouns (capitalized in the above example) as data types. A similar strategy is applicable to objects: in a phrase like

> Determine the number of days between THE_EARLIER_DATE to the end of its month. Keep track of this THAT_NUMBER in the variable called "DAY_COUNTER"

there are direct references "THE_EARLIER_DATE" and "THAT_NUMBER", marked by "the"/"that" and a proper noun "DAY_COUNTER". They are identified as program objects.

The third kind of concepts translated from text to program, the operators, are identified either as verbs or as attributes or descriptive expressions. For example, in the sentence

> If the two given dates ARE_IN_THE_SAME_MONTH, THE_NUMBER_OF_DAYS between them is the DIFFERENCE_BETWEEN their DAYS_OF_MONTH,

there is a predicate "ARE_IN_THE_SAME_MONTH" and descriptive expressions "THE_NUMBER_OF_DAYS", "DIFFERENCE_BETWEEN" and "DAYS_OF_MONTH", which become program operators.

Abbott's procedure gives some guidelines for translating the specification text into a program, but these guidelines are not completely automatable. Given a part-of-speech (POS) tagger, attaching a POS-tag to every word, it would be possible to identify nouns, verbs, etc. Such taggers were not available at the time Abbott wrote the paper but are available now. The precision of currently available taggers lies at about 97% [11, 12]. Even the most precise tagger does not achieve a 100% precision and can become an error source.

A POS tagger would allow to identify common and proper nouns: We could say that a common noun is any word assigned the noun tag. Identification of proper nouns is a bit more complex. There exist approaches to recognize standard classes of proper names, i.e. names of people, places and organizations [13]. These approaches can also

[1] This list and the examples are taken from Abbott's paper [10]

be transferred to other classes of proper names, as for example shown by Witte et al. [14] for programming concepts, i.e. variables, classes, and objects. However, to apply these techniques, it is necessary to manually define the set of domain-specific keywords. For example, Witte et al. introduce the keyword "variable" for variables and then recognize names like "variable X" as variable names. To apply Abbott's rules to the above example, we would have to manually define "counter" as a keyword. Then, we could identify "DAY_COUNTER" as well as other counters as program objects.

Abbott's third rule is really difficult to automate: Abbott himself gives examples of operators expressed by a verb, a noun phrase, or a prepositional phrase. However, he does not provide guidelines how to distinguish phrases representing an operator from non-operator phrases.

Chen's method of building entity-relationship (ER) diagrams [15] is similar to Abbott's approach in that each maps natural language texts to application domain models. Chen defines a set of rules for translating English text to ER diagrams. The first two rules coincide with Abbott's ones:

1. A common noun corresponds to an entity type.
2. A transitive verb corresponds to a relationship type.

Other rules are specific to the ER-representation:

3. An adjective in English corresponds to an attribute of an entity in the ER-diagram.
4. An adverb in English corresponds to an attribute of a relationship in an ER-diagram.
8. The objects of algebraic or numeric operations can be considered as attributes.
9. A gerund in English corresponds to a relationship-converted entity type in ER-diagrams.

The remaining rules address firm expression patterns:

5. If the sentence has the form: "There are ... X in Y", we can convert it into the equivalent form "Y has ... X"
6. If the English sentence has the form "The X of Y is Z" and if Z is a proper noun, we may treat X as a relationship between Y and Z. In this case, both Y and Z represent entities.
7. If the English sentence has the form "The X of Y is Z" and if Z is not a proper noun, we may treat X as an attribute of Y. In this case, Y represents an entity (or a group of entities), and Z represents a value.

It is easy to see that Rules 1–4 and 8–9 are very similar to Abbott's rules. They just target at another representation form as Abbott's rules (ER-diagrams instead of Ada programs). Rules 5–7 create additional relations by analyzing firm expression patterns.

Saeki et al. [16] designed a tool aimed at automation of the approaches introduced above. They extract nouns and verbs from the text and build a noun table and a verb table. Then they select actions and action relations from the verb table. Although they aim at constructing an object-oriented model from a specification text, they do not perform any concept classification, which would yield a class hierarchy, but produce a flat model. An approach that performs not only concept extraction, but also classification, is presented below.

Ontology Building Technique: Syntactic text analysis techniques can be used to build an application domain ontology as well. In computer science, an ontology consists of a concept hierarchy, also called taxonomy, augmented with some more general, other than "is-a", relations. A taxonomy, in turn, consists of a term list and the "is-a"–relation, also called specialization or sub-typing. Thus, extraction of a domain-specific ontology consists of three basic steps:

1. term extraction
2. term clustering and taxonomy building, finding "is-a" relations
3. finding associations between extracted terms

These steps are explained below in detail.

Extraction of terms from requirements documents: To extract terms, each sentence is parsed and the resulting parse tree is decomposed. Noun phrases that are related to the verb of the sentence are extracted as domain concepts. For example, from the sentence "The control unit sends an alarm message in a critical situation" "send" is extracted as the main verb, "control unit" as the subject and "alarm message" as the direct object.

Term clustering: The second step clusters related concepts. Two concepts are considered as related and put into the same cluster if they occur in the same grammatical context. I.e., two terms are related in the following cases:

- They are subjects of the same verb.
- They are direct objects of the same verb.
- They are indirect objects of the same verb and are used with the same preposition.

For example, if the document contains two sentences like

1. "The control unit sends an alarm message in a critical situation"
2. "The measurement unit sends measurements results every 5 seconds",

the concepts "control unit" and "measurement unit" are considered as related, as well as "alarm message" and "measurements results".

Taxonomy building: Concept clusters constructed in the previous step are used to build the taxonomy by joining overlapping concept clusters. The emerging larger clusters represent more general concepts. For example, the two clusters "{alarm message, measurements results}" and "{control message, measurements results}" are joined into the larger cluster

{alarm message, control message, measurements results}

because they share the common concept "measurement results". The new joint cluster represents the more general concept of possible messages.

This step also aids in identifying synonyms[2] because synonyms are often contained in the same cluster. For example, if a cluster contains both "signal" and "message", the domain analyst performing the ontology construction can identify them as synonyms.

In the original approach [17], the tool ASIUM [18] was used to cluster terms and build a taxonomy. Other clustering approaches are possible as well [19].

[2] different names for the same concept

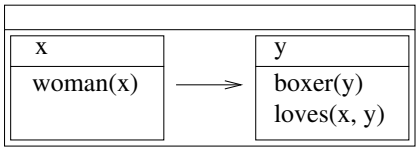

Fig. 1. Discourse Representation Structure (DRS) for "Every woman loves a boxer"

Associations/relations mining: There is a potential association between two concepts if they occur in the same sentence. Each potential association then has to be validated by the requirements engineer before being recorded as an association between concepts.
Note, that validation of the association proposed by the association mining tool automatically implies validation of the requirements document. If the tool suggests an association that *cannot* be valid, for example a pair containing completely unrelated concepts, then we have detected an evidence that the requirements document contains some inconsistent noise that must be eliminated. The tool KAON [20] can be used for this step. Maedche and Staab [21] give an in–depth treatment of association mining.

More details on the ontology extraction approach sketched above can be found in [17].

5.3 Interpreting Sentences: Semantic Approaches to Text Analysis

Semantic approaches are the most demanding on the formulation. In return, they extract the most information from text. As the name says, these approaches build a semantic representation as their results. Each of these approaches uses one of two kinds of semantic representations: discourse representation structures or mapping of verbs to predicates with their arguments.

Discourse representation structure (DRS) is a kind of first order predicate logic with explicit introduction of variables and definitions of variable scopes and accessibility. An example DRS, taken from Blackburn et al. [22], is shown in Figure 1. This DRS consists of one large scope box with two subordinate scope boxes. Each of the subordinate scopes contains some object references, represented by the variables x and y, and statements about these objects. For example, the left box introduces the object x and states $woman(x)$. The right box introduces a new object y and states $boxer(y)$ and $loves(x, y)$. The whole DRS represents the sentence "Every woman loves a boxer" and is equivalent to the formula

$$\forall x.woman(x) \Rightarrow \exists y.boxer(y) \wedge loves(x, y). \tag{1}$$

(See the technical report by Blackburn et al. [22] for the translation rules between DRSs and formulae and for other details.)

To compute the semantics-DRS, Blackburn et al. [22] define a calculus for such structures. This calculus defines operations on DRSs, like merging, conjunction, negation, and so on. In order to translate a sentence to the representing DRS, a DRS–λ–

expression[3] is associated with every word, all the word–λ–expressions are chained to one sentence–λ–expression and then this large λ–expression is evaluated according to the reduction rules of the λ–calculus.

The following example shows semantics calculation with ordinary first order formulae, but a very similar calculation can be done with discourse representation structures. The example uses ordinary first order formulae just not to over-complicate the matters. First of all, λ–expressions are introduced for every word class:

$$\begin{array}{lll}
\text{Proper names:} & Alice & = \lambda P.(P\ Alice) \\
\text{Common names:} & woman & = \lambda y.(woman(y)) \\
\text{Intransitive verbs:} & walks & = \lambda x.(walk(x)) \\
\text{Transitive verbs:} & loves & = \lambda X.(\lambda z.(X\ (\lambda x.love(z,\ x)))) \\
\text{“every”:} & every & = \lambda P.(\lambda Q.(\forall x.((P\ x) \rightarrow (Q\ x)))) \\
\text{“a”:} & a & = \lambda P.(\lambda Q.(\exists y.((P\ y) \wedge (Q\ y))))
\end{array} \tag{2}$$

It is possible to calculate the sentence semantics just by replacing every word with its λ–expression and performing standard reductions defined in the λ–calculus. For example, the semantics of “Alice loves a man” is calculated as follows:

$$\begin{array}{l}
Alice\ loves\ a\ man = \\
= \lambda_{Alice}\ (\lambda_{loves}\ (\lambda_{a}\ (\lambda_{man}))) \\
= \lambda_{Alice}\ (\lambda_{loves}\ ((\lambda P.(\lambda Q.(\exists y.((P\ y) \wedge (Q\ y)))))\ (\lambda y.(man(y))))) \\
= \lambda_{Alice}\ (\lambda_{loves}\ (\lambda Q.(\exists y.(((\lambda y.man(y))\ y) \wedge (Q\ y))))) \\
= \lambda_{Alice}\ (\lambda_{loves}\ (\lambda Q.(\exists y.((man(y)) \wedge (Q\ y))))) \\
= \lambda_{Alice}\ ((\lambda X.(\lambda z.(X\ (\lambda x.love(z,\ x)))))\ (\lambda Q.(\exists y.((man(y)) \wedge (Q\ y))))) \\
= \lambda_{Alice}\ (\lambda z.((\lambda Q.(\exists y.((man(y)) \wedge (Q\ y))))\ (\lambda x.love(z,x)))) \\
= \lambda_{Alice}\ (\lambda z.(\exists y.(man(y) \wedge (\lambda x.love(z,x))\ y))) \\
= \lambda_{Alice}\ (\lambda z.(\exists y.(man(y) \wedge love(z,y)))) \\
= (\lambda P.(P\ Alice))\ (\lambda z.(\exists y.(man(y) \wedge love(z,y)))) \\
= (\lambda z.(\exists y.(man(y) \wedge love(z,y))))\ Alice \\
= \exists y.(man(y) \wedge love(Alice,y))
\end{array} \tag{3}$$

As the above example shows, the semantics calculation is quite complicated. Furthermore, introduction of additional words in the sentence would add additional λ–expressions to the computation and would disturb it. This makes approaches of this kind extremely fragile. They are applicable to only restricted specification languages with fixed grammars.

To make this approach applicable to document analysis, it is necessary to restrict the natural language. Fuchs et al. [24], for example, introduced a controlled specification language (ACE, Attempto Controlled English). The language is restricted in the following way:

Vocabulary: The vocabulary of ACE comprises

[3] for an introduction to λ–calculus see, for example, Baader and Nipkow [23]

- predefined function words, e.g. determiners, conjunctions, prepositions
- user-defined, domain-specific content words, e.g. nouns, verbs, adjectives, adverbs

Sentences: There are

- simple sentences,
- composite sentences,
- query sentences.

Simple sentences have the form $subject + verb + complements + adjuncts$.

Firm sentence structure and the necessity to explicitly define the vocabulary in advance restrict the applicability of ACE and other λ–calculus based approaches to real requirements documents.

The other group of semantic approaches uses verb subcategorization frames for semantics representation. A verb subcategorization frame is a verb with its arguments, namely its subject and its objects. For example, for the verb "send", possible arguments are: sender, receiver, sent object. When interpreting the sentence "Component X sends message Y to component Z", in the semantic representation "component X" becomes the sender, "component Z" the receiver and "message Y" the sent object.

This idea is used by Hoppenbrouwers et al. [25] to identify domain concepts and relations between them. Hoppenbrouwers et al. define a set of roles, or semantic tags, like $agent$, $action$, $patient$ etc. The analyst marks the relevant words with these tags. For example, the sentence "Component X sends message Y to component Z" can be manually tagged as

$$(\texttt{Component X})/agent\ \texttt{sends}/action\ (\texttt{message Y})/patient\ \texttt{to}\ (\texttt{component Z})/other.$$

Sentences marked in such a way are used to find $agents$, $actions$, and $patients$.

Ambriola and Gervasi [26] go further than Hoppenbrouwers et al. and build a semantic tree representation of a sentence. To build the semantic representation, they start with a glossary. Each term in the glossary is manually furnished with an associated list of tags. These tags are then used to automatically mark every word of a sentence. For example, the sentence

> The terminal sends the password to the server

can be canonized as

$$\texttt{terminal}/IN/OUT\ \texttt{sends password}/INF\ \texttt{to server}/IN/OUT/ELAB$$

The applied tags are domain-specific.

After the tagging, a set of transformation rules is applied to marked sentences, translating the tagged sentence to a semantic tree. Figure 2, taken from Ambriola and Gervasi [26], shows an example semantic tree. It shows the representation of the sentence

> When the server receives from the terminal the password, the server stores the signature of the password in the system log.

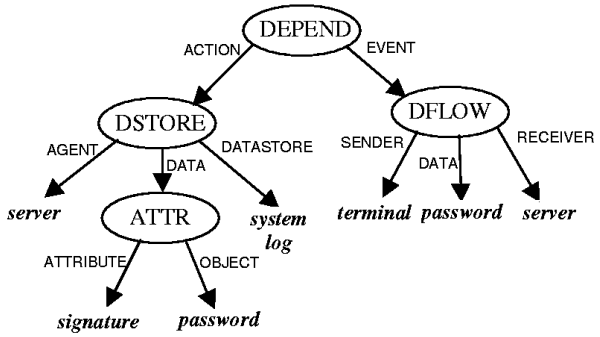

Fig. 2. Semantic tree according to Ambriola and Gervasi [26]

This tree shows dependencies between actions, namely that the left subtree depends on the right one. Furthermore, the tree shows the semantics of every action. This rich representation allows for extraction of abstract state machines, ER diagrams and other formalisms [27, 28].

The drawback of this approach is obvious: the approach is able to analyze only sentences that fit into the predefined transformation rules. The transformation rules are defined manually and it is almost impossible to cater for all the potential constructions that can occur in a real requirements document.

The above approach was further developed and improved by Gervasi and Zowghi [29]. In the improved approach the tool became interactive: for the words not contained in the user's glossary, the user has to specify first, to which category the new word belongs, for example "sender", "receiver", "message", ... Then, when the category of every word is known, the tool translates every sentence to a first-order-logic formula, based on a parse tree like the tree shown in Figure 2. Then, a theorem prover is applied to check whether the set of formulae obtained for the whole text is satisfiable, i.e., contradiction-free.

Rolland and Ben Achour [30] apply the idea of case frames, which is very similar to the approach by Ambriola and Gervasi, introduced above, to whole sequences of sentences to build the semantics of a use case description. As in the approaches by Ambriola and Gervasi and by Gervasi and Zowghi, only firm expression patterns are supported. They also define a set of expressions for temporal relations between individual sentences.

Although interesting in itself, semantics representation is not necessarily the final goal of document analysis. Vadeira and Meziane [31] use semantic text analysis and formulae representation to produce a VDM [32] model. They start with a set of logical formulae and translate them to an ER model first. To build the ER model, they assume that the predicates that build up the formulae are the relationships and predicate arguments are the entities. Then they use a set of heuristics to determine multiplicity of the relations in the basis of formulae. The final step in their approach is the translation of the ER-diagram to the formal specification language VDM.

Table 3. Rhetorical relations according to Asher and Lascarides [33]

Narration	Max fell. John helped him up.
Elaboration	He had a great meal. He ate salmon. He devoured lots of cheese.
Explanation	Max fell. John pushed him.
Result	Max switched off the light. He drew the blinds. The room became dark.
Background	John moved from Brixton to St. John's Wood. The rent was less expensive.
Contrast	-Max owns several classic cars. -No, he does not. -He owns two 1967 Alpha spiders.
Parallel	John said that Mary cried. Sam did too.

Although the idea of semantics analysis is very promising for the step from a requirements document to a system model, the approaches are not really mature yet. They are applicable solely to sentences with restricted grammar. What is lacking is a semantic broad-domain parser, putting no restrictions on allowed expression forms and able to cope with sentences that are not completely grammatically correct. It is an open question whether such a parser will ever become possible.

5.4 Logic to Capture Pragmatics

To capture pragmatics, it is necessary to understand links between sentences. To model these links, Asher and Lascarides [33] introduce seven rhetorical relations: narration, elaboration, explanation, result, background, contrast, parallel. Table 3 shows their examples for each rhetorical relation. For every relation, they introduce logical operations combining DRS representations for every sentence, of the type described in Section 5.3, to a DRS representation of the discourse.

The "contrast" relation may be especially useful in the context of goal identification. As stated in Section 3, negation of the problems with the existing system is a potential source of the goals for the system to be built. The "contrast" relation potentially identifies problems. For example, in the dialogue excerpt shown in Table 1, there is a "contrast" relation between the phrase "Technologies that could help might work well in a lab…" and the previous statement by the FAA representative. The negation of the problem, namely "Technologies that could help should work not only in the lab", identifies the goal. Unfortunately, automated recognition of the relation types is not possible at the moment.

5.5 Applicability of Different Kinds of Analysis to Goal Identification

The discussion in Sections 5.1– 5.4 makes clear that goal identification takes place on the lexical and pragmatic analysis levels. This discussion makes also obvious that the higher the analysis level, the lower the precision, and automation, as sketched in Figure 3. For lexical and syntactic analysis, 100% recall is possible, in the sense that there exist tools that assign a POS tag to every word and assign a parse tree to every sentence, even if the sentence is not completely grammatically correct. If we restrict lexical analysis to search for certain keywords, 100% precision is possible, with a `grep`-like tool.

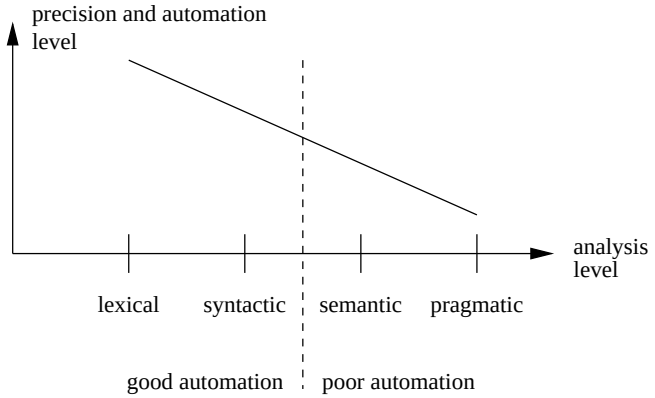

Fig. 3. Precision of different analysis levels

For syntactic analysis, there are taggers available with the precision of about 97% [12], and parsers with the precision of about 80% [34]. Beyond syntactic analysis, we either have to abandon the idea to handle broad domain language and trade recall for precision, like in Attempto Controlled English [24] and similar languages, or fall back to manual analysis. The situation gets even worse if we try to analyze pragmatics. Due to these problems it is not very likely that full-fledged goal identification be automated in the near future, or even in the far.

6 Case Study, Goal Identification by Means of Natural Language Processing

Section 4 shows how to identify goals in a text by close inspection of the text. Now we want to systematize the inspection procedure. To systematize the analysis, we apply two observations to every paragraph, motivated by the goal identification rules by van Lamsweerde (cf. Section 3):

- Phrases like “have to” and “in order to” may directly show a goal.
- If the first sentence of a paragraph does not contain any of the above phrases, the first sentence states the reason why the previous paragraph is problematic. In this case, the negation of this sentence shows the stakeholder’s goal.

6.1 Evaluation of the Rule Application

Table 4 shows the results of the application of the above rules to the case study. The application was performed manually by adhering to the rules as strictly as possible. This means that in some cases not the first sentence of the paragraph but the first meaningful one was taken into consideration. For example, statements like “come on”, “well…”, “we can deal with it” were ignored, as they do not contribute to the identification of

Table 4. Application of the hypothesis to the case study

	Sentence	**State of the art/Goal**	**Evaluation**
1	We have to ban on airplane passengers taking liquids on board in order to increase security following the recent foiled United Kingdom terrorist plot.	State of the art: we do not ban passengers taking liquids, terrorist plot like in the UK is possible. Goals: ban passengers taking liquids, increase security	
2	Technologies that could help might work well in a lab, but when you use it dozens of times daily screening everything from squeeze cheese to Channel No. 5 [sic] you get False Alarms ...	Goals: technologies should work not only in the lab, and the proportion of false alarms in daily screening should not lie above some threshold	Goal correctly identified
3	Generating false positives helped us stay alive; maybe that wasn't a lion that your ancestor saw, but it was better to be safe than sorry.	No goal identifiable. However, this sentence is not useless: It states that the threshold mentioned above is not necessarily zero.	—
4	It's not easy to move 2 million passengers through U.S. airports daily.	Goal: the screening system has to handle 2 million passengers daily	Goal correctly identified
5	We can deal with it. What if you guys take frequent breaks?	No goal identifiable	—
6	Sounds good though we do take breaks and are getting inspected.	No goal identifiable	—
7	We have yet to take a significant pro-active step in preventing another attack everything to this point has been reactive.	State of the art: We do not take pro-active steps. Goal: We have yet to take pro-active steps	Goal correctly identified
8	On each dollar that a potential attacker spends on his plot we had to spend $1000 to protect.	Goal: we should not spend too much on the screening procedure, it should remain affordable	Goal correctly identified
9	We need to think ahead. For instance, nobody needs a metal object to bring down an airliner, not even explosives.	Goal: identify other types of objects to be banned	Goal correctly identified
10	Airlines need to take the lead on aviation security.	Goal: Airlines need to take the lead on aviation security, not FAA.	Goal correctly identified
11	Sir, a lot of airlines are not doing well and are on the Government assistance.	Goal: Airlines should not be responsible for additional cost-intensive tasks.	Goal correctly identified
12	I think that enforcing consistency in our regulations and especially in their application will be a good thing to do.	State of the art: regulations are inconsistent Goal: regulations should be consistent.	Goal correctly identified
13	Ok, we had very productive discussion	No goal identifiable	—

the goals. For this reason, Table 4 sometimes lists other than the first sentence of the paragraph.

It is important to emphasize that the negations listed in Table 4 were not constructed by purely textual deletion or addition of "not" at some position in the sentence. Furthermore, negations had to be generalized. For example, "It's not easy to move 2 million passengers. . . ", statement from paragraph 4, was negated to "It should be easy to move 2 million passengers. . . " and then generalized to "The screening system has to handle 2 million passengers daily". In a similar way, "On each dollar that a potential attacker spends on his plot we had to spend $1000 to protect" was negated to "On each dollar that a potential attacker spends on his plot we should spend much less than $1000 to protect", and generalized to "The screening procedure should remain affordable". The negation performed for the second sentence, resulting in "On each dollar . . . we should spend much less than $1000 to protect", cannot be performed on the semantic level, let alone the syntactic and lexical ones. A negation on the semantic level would result in "We should not spend $1000 to protect". This negation is correct too, but it still allows unintended interpretations like "We should spend more than $1000 to protect" or "We should spend $999 to protect". Building sensible negation on pragmatic level, like "We should spend much less than $1000", requires knowledge going beyond pure sentence semantics. This knowledge is absolutely obvious for humans and extremely difficult to capture in AI applications.

It is easy to see that Table 4 contains all the goals identified by ad-hoc analysis in Section 4. However, it is necessary to bear in mind that the case study was rather small and that both analysis runs, ad-hoc and systematic, were performed by the same person, which makes the results potentially biased. Thus, to properly evaluate the rules for goal identification, a controlled experiment is necessary. In the experiment, one group of people would have to identify goals using the introduced rules, and the other group would have to identify the goals ad-hoc.

6.2 Possible Implementation

To implement the introduced procedure for goal identification, it is necessary to solve at least two problems:

- It is necessary to define what a meaningful sentence is, in order to analyze the first meaningful sentence of every paragraph.
- Negation is not always possible by simple deletion or addition of "not". Furthermore, negated sentences have to be generalized. Generalization can be seen also as the application of the "WHY"-question to the negation. (I.e., we would permanently ask the question "why is it really a problem?")

The first problem is relatively simple from the point of view of computational linguistics: We could eliminate sentences without grammatical subject, like "come on" and "well. . . ", as well as questions, like "What do you suggest?" in the case study document. This would work for most paragraphs of the considered case study, but still not for all. To achieve high precision, manual post-processing would be necessary even for this step.

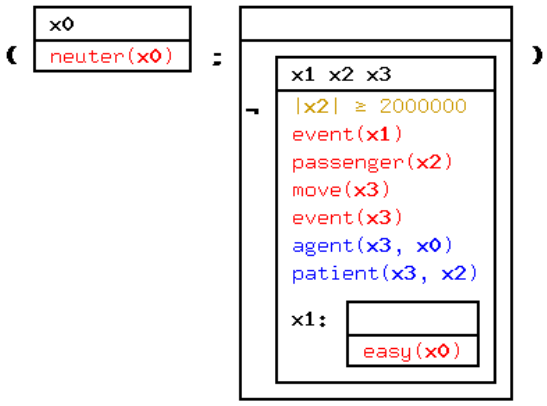

Fig. 4. DRS for the sentence "it's not easy to move 2 million passengers"

The second problem, the negation, is much more difficult. Purely syntactic negation is obviously insufficient, as we would negate "it's not easy to move 2 million passengers..." to "it's easy to move 2 million passengers...", that does not really state the goal "it should be easy to move 2 million passengers...". To go beyond pure syntactic analysis, we could represent the sentence to be negated as a discourse representation structure (DRS) (cf. Section 5.3 and [35, 36]). For the sentence "'it's not easy to move 2 million passengers..." this would result in the representation shown in Figure 4. This representation is created by the DRS tool Boxer available as a component of the C&C tool suite [37]. Then we can take a negation on the DRS level. This would be equivalent to representation of the sentence as a formula in first order logic and then taking a negation on the logical level. However, this results, again, in removing the negation from the second box ("¬"-sign) and, therefore, in the sentence "it's easy to move 2 million passengers...".

Thus, even semantic negation is not sufficient to obtain the goals and we have to move to negation on the pragmatics level. Negation on the pragmatics level would include profound knowledge of real world and knowledge of motivation for certain statements. Then we can get, for example, from "On each dollar that a potential attacker spends on his plot we had to spend $1000 to protect" to "On each dollar that a potential attacker spends on his plot we should spend much less than $1000 to protect". On this level we could also implement generalization. For example, in the case study we had to generalize "On each dollar that a potential attacker spends on his plot we should spend much less than $1000 to protect" to "The screening procedure should remain affordable". Unfortunately, this is far beyond the capabilities of state-of-the-art linguistic tools.

7 Summary

In this paper a method for identification of stakeholders' goals by analyzing stakeholders' dialogs was introduced. This method is based on two key assumptions:

– A sentence containing certain keywords directly represents a goal.
– Otherwise, if a sentence is the first meaningful sentence of its paragraph, the negation of this sentence represents a goal.

The second rule used in this paper, the negation rule, can also be seen as an application of the WHY-rule of Section 3 to the dialog: We are just asking the question, why a particular statement was made. One of the reasons to start a new dialog segment is a stakeholder's disagreement with the last statement of his opponent. In this case, the negation of the first statement of the new dialog segment shows the reason for the disagreement, which is some goal of the stakeholder.

Explicit goal identification is important for several reasons. Goals serve to achieve requirements completeness and pertinence, managing requirements conflicts, etc. [3]. The presented approach is especially suitable to manage requirements conflicts when negotiating requirements: In the Win-Win negotiation approach [2], requirements conflicts are resolved in such a way that the *goals* of every stakeholder remain satisfied. In the case of goal conflicts, such a resolution is impossible. Thus, identification of goals and goal conflicts, as in the presented paper, contributes to identification of potential problems early in the development process.

Acknowledgments. I am very grateful to Daniel Berry and two anonymous reviewers. They helped a lot to improve the paper.

References

1. Luqi, Kordon, F.: Case Study: Air Traveling Requirements Updated (Blog scenario). this volume
2. Grünbacher, P., Boehm, B.W., Briggs, R.O.: EasyWinWin: A groupware-supported methodology for requirements negotiation, `http://sunset.usc.edu/research/WINWIN/EasyWinWin/index.html`.
3. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering, pp. 249–263. IEEE Computer Society (2001)
4. Berry, D.: Natural language and requirements engineering - nu? Talk at the International Workshop on Requirements Engineering, Imperial College, London, April 25, 2001, `http://www.ifi.unizh.ch/groups/req/IWRE/papers&presentations/Berry.pdf`,
5. Goldin, L., Berry, D.M.: AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation. Automated Software Eng. 4, 375–412 (1997)
6. Maarek, Y.S., Berry, D.M.: The use of lexical affinities in requirements extraction. In: Proceedings of the 5th international workshop on Software specification and design, pp. 196–202. ACM Press, New York, NY, USA (1989)
7. Lecoeuche, R.: Finding comparatively important concepts between texts. In: The Fifteenth IEEE International Conference on Automated Software Engineering (Grenoble, France), pp. 55–60. IEEE Press (2000)
8. Abrial, J.R., Börger, E., Langmaack, H.: The steam boiler case study: Competition of formal program specification and development methods. In Abrial, J.R., Borger, E., Langmaack, H.,

eds.: Formal Methods for Industrial Applications. LNCS, vol. 1165, pp. 1–12. Springer, Heidelberg (1996), http://www.informatik.uni-kiel.de/~procos/dag9523/dag9523.html.
9. Sawyer, P., Rayson, P., Cosh, K.: Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Trans. Softw. Eng. 31, 969–981 (2005)
10. Abbott, R.J.: Program design by informal English descriptions. Communications of the ACM 26, 882–894 (1983)
11. Ratnaparkhi, A.: A maximum entropy model for part-of-speech tagging. In Brill, E., Church, K., eds.: Proceedings of the Conference on Empirical Methods in Natural Language Processing (Somerset, New Jersey), pp. 133–142. Association for Computational Linguistics, Morristown, NJ, USA (1996)
12. Curran, J.R., Clark, S., Vadas, D.: Multi-tagging for lexicalized-grammar parsing. In: 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Sydney, Australia, 17-21 July, pp. 697–704. Association for Computational Linguistics, Morristown, NJ, USA (2006)
13. Language-Independent Named Entity Recognition, http://www.cnts.ua.ac.be/conll2003/ner/.
14. Witte, R., Li, Q., Zhang, Y., Rilling, J.: Ontological Text Mining of Software Documents. In Kedad, Z., Lammari, N., Méthais, E., Meziane, F., Rezgui, Y., eds.: Application of Natural Language to Information Systems. LNCS, vol. 4592, pp. 168–180. Springer, Heidelberg (2007)
15. Chen, P.: English sentence structure and entity-relationship diagram. Information Sciences 1, 127–149 (1983)
16. Saeki, M., Horai, H., Enomoto, H.: Software development process from natural language specification. In: Proceedings of the 11th international conference on Software engineering, pp. 64–73. ACM Press, New York, NY, USA (1989)
17. Kof, L.: Text Analysis for Requirements Engineering. PhD thesis, Technische Universität München (2005)
18. Faure, D., Nédellec, C.: ASIUM: Learning subcategorization frames and restrictions of selection. In Kodratoff, Y., ed.: 10th European Conference on Machine Learning (ECML 98) – Workshop on Text Mining (1998)
19. Nenadić, G., Spasić, I., Ananiadou, S.: Automatic discovery of term similarities using pattern mining. In: Proceedings of CompuTerm 2002, pp. 43–49. Association for Computational Linguistics, Morristown, NJ, USA (2002)
20. Welcome to KAON, http://kaon.semanticweb.org/.
21. Maedche, A., Staab, S.: Discovering conceptual relations from text. In W.Horn, ed.: ECAI 2000. Proceedings of the 14th European Conference on Artificial Intelligence, pp. 321–325. IOS Press, Amsterdam (2000)
22. Blackburn, P., Bos, J., Kohlhase, M., de Nivelle, H.: Inference and computational semantics. CLAUS-Report 106, Universität des Saarlandes, Saarbrücken (1998)
23. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1999)
24. Fuchs, N.E., Schwertel, U., Schwitter, R.: Attempto Controlled English (ACE) language manual, version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich (1999), http://www.ifi.unizh.ch/attempto/publications/papers/ace3_manual.pdf.
25. Hoppenbrouwers, J., van der Vos, B., Hoppenbrouwers, S.: NL structures and conceptual modelling: grammalizing for KISS. Data Knowl. Eng. 23, 79–92 (1997)
26. Ambriola, V., Gervasi, V.: Experiences with domain-based parsing of natural language requirements. In Fliedl, G., Mayr, H.C., eds.: Proc. of the 4th International Conference on Applications of Natural Language to Information Systems, pp. 145–148. Number 129 in OCG Schriftenreihe (Lecture Notes), Oesterreichische Computer Gesellschaft (1999)

27. Ambriola, V., Gervasi, V.: The Circe approach to the systematic analysis of NL requirements. Technical Report TR-03-05, University of Pisa, Dipartimento di Informatica (2003)
28. Gervasi, V.: Synthesizing ASMs from natural language requirements. In: Proc. of the 8th EUROCAST Workshop on Abstract State Machines, pp. 212–215. Universidad de Las Palmas (2001)
29. Gervasi, V., Zowghi, D.: Reasoning about inconsistencies in natural language requirements. ACM Trans. Softw. Eng. Methodol. 14, 277–330 (2005)
30. Rolland, C., Ben Achour, C.: Guiding the construction of textual use case specifications. Data & Knowledge Engineering Journal 25, 125–160 (1998)
31. Vadera, S., Meziane, F.: From English to formal specifications. The Computer Journal 37, 753–763 (1994)
32. Jones, C.B.: Systematic Software Development using VDM. Prentice-Hall, Upper Saddle River, NJ 07458, USA (1990)
33. Asher, N., Lascarides, A.: Logics of Conversation. Cambridge University Press (2003)
34. Clark, S., Curran, J.R.: Wide-coverage efficient statistical parsing with ccg and log-linear models. Comput. Linguist. 33, 493–552 (2007)
35. Bos, J., Clark, S., Steedman, M., Curran, J.R., Hockenmaier, J.: Wide-coverage semantic representations from a CCG parser. In: COLING '04: Proceedings of the 20th international conference on Computational Linguistics, pp. 1240–1246. Association for Computational Linguistics, Morristown, NJ, USA (2004)
36. Bos, J.: Towards wide-coverage semantic interpretation. In: Proceedings of the 6th International Workshop on Computational Semantics (IWCS 6), pp. 42–53 (2005)
37. C&C Tools, `http://svn.ask.it.usyd.edu.au/trac/candc`