Improving the Quality of Requirements Specifications via Automatically Created Object-Oriented Models

Daniel Popescu
Computer Science Department
University of Southern California
Los Angeles, CA 90089, USA
dpopescu@usc.edu

Spencer Rugaber
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, USA
spencer@cc.gatech.edu

Nenad Medvidovic
Computer Science Department
University of Southern California
Los Angeles, CA 90089, USA
neno@usc.edu

Daniel M. Berry
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON N2L 3G1, Canada
dberry@uwaterloo.ca

Abstract

In industry, reviews and inspections are the primary methods to identify ambiguities, inconsistencies, and under specifications in natural language (NL) software requirements specifications (SRSs). However, humans have difficulties identifying ambiguities and tend to overlook inconsistencies in a large NL SRS. This paper presents a three-step, semi-automatic method, supported by a prototype tool, for identifying inconsistencies and ambiguities in NL SRSs. The method combines the strengths of automation and human reasoning to overcome difficulties with reviews and inspections. First, the tool parses a NL SRS according to a constraining grammar. Second, from relationships exposed in the parse, the tool creates the classes, methods, variables, and associations of an object-oriented analysis model of the specified system. Third, the model is diagrammed so that a human reviewer can use the model to detect ambiguities and inconsistencies. Since a human finds the problems, the tool has to have neither perfect recall nor perfect precision. The effectiveness of the approach is demonstrated by applying it and the tool to a widely published example NL SRS. A separate study evaluates the tool's domain-specific term detection.

1. Introduction

The typical industrial software specifier writes software requirements specifications (SRSs) in natural language (NL). Even if a final SRS is written in a formal language, its first draft is usually written in a NL. A NL SRS enhances the communication between all the stakeholders. However, on the downside, often a NL SRS is imprecise and ambiguous.

Many an organization follows a three-step review process to assess the quality of a NL SRS and to identify ambiguities and other defects in the NL SRS [1]. First, assigned reviewers try to find defects in the document. Second, in a meeting of the reviewers, all found defects are collected and rated according to their severities. Third, the reviewed NL SRS and the collected defects are sent back to the authors for corrections.

In this process, the quality of a review of a document is dependent mainly upon how effectively each human reviewer is able to find ambiguities and other defects in the document. However, a human reviewer and even a group of them have difficulties identifying all ambiguities, even when using aids such as checklists. A human reviewer might overlook some defects while reading a SRS, because he might assume that the first interpretation of the document that came to his mind is the intended interpretation, unaware of other possible understandings. In other words, he *unconsciously disambiguates* an ambiguous document [8].

When a NL SRS is large, some ambiguities might remain undetected, because ambiguities caused by interaction of distant parts of the NL SRS are difficult to detect. In any one review sessions, only an excerpt of the NL SRS can be reviewed. Any ambiguity in the reviewed excerpt that is caused by interaction with a part of the NL SRS outside the excerpt may not be detectable. Moreover, when a NL SRS is large, lack of time may prevent some parts of the NL SRS from ever being reviewed. Having a faster method for conducting reviews would permit larger chunks of the whole NL SRS to be reviewed at once. It would permit also faster reviews so that more reviews and, thus, greater coverage of the NL SRS would be possible in any duration.

A controlled, grammar-constrained NL [6,7] helps to reduce ambiguities by constraining what can be said in the NL. One possible grammar rule eliminates passive voice and, therefore, ensures that the doer of an action is always known. However, a controlled NL can address only syntactic ambiguity. Semantic ambiguity is beyond its control.

Because a semantic model omits unnecessary details, it helps to reduce complexity, and it helps the user to visualize important aspects. Therefore, with the help of a semantic model, a human reviewer can validate larger system descriptions than otherwise. Moreover, the human reviewer can focus on conceptual correctness and does not need to worry about the consistent use of concepts and correct grammar usage. Therefore, if a semantic model can be created of the system specified by a NL SRS, a thorough review of the NL SRS becomes easier.

Of course, constructing a model bears the risks of introducing new defects and of the model's not representing the original NL SRS. Researchers have tried to mitigate these risks by using automatic approaches and NL processing (NLP) [22, 18, 9]. A software tool can scan, search, browse, and tag huge text documents much faster than a human analyst can. Furthermore, a tool works rigorously, decreasing

the risk of overlooked defects and unconscious disambiguation. Each of some of the approaches tries to build a model from a NL source and may even try to reason about the content, possibly with assistance from the human user.

In any automatic approach, the recall and precision of its NL parsing and of its domain-specific term (DST) identification bound the quality of the approach. The typical domain has its own terms, abbreviations, and vocabulary. Therefore, a tool must detect these to create a correct model. Each of some approaches has difficulty to create a correct model because it relies on a semantic network that is based on a predefined domain dictionary. For many domains a domain dictionary does not exist. Each of other approaches requires the human specifiers to build a domain dictionary while writing the NL SRS. Clearly, not every requirements engineering (RE) process is so mature that in it, a domain dictionary is built. Even in a mature process, some domain terms might be forgotten, because the analysts assume that these terms are widely understood and recognized.

We have created an approach for helping a specification writer or reviewer identify ambiguities in a NL SRS, in which the approach tries to address all of the above mentioned problems. Hereinafter, the new approach is called "our approach" to distinguish it from other approaches. For our approach, we have built a prototype tool, called "Dowser". Dowser is based on one controlled NL. It can create from any NL SRS an object-oriented (OO) diagram, which can then be assessed by human reviewers.

In the first step, Dowser parses a NL SRS and extracts the classes, methods, and associations of a textual class model from the NL SRS. In the second step, Dowser diagrams the constructed textual class model. In the third step, a human reviewer can check the generated diagram for signs of defects in the NL SRS. Dowser and our approach are based on NL processing (NLP) and not on NL understanding. Dowser cannot judge whether or not the produced model describes a good set of requirements or classes. This judgement requires understanding. Therefore, in our approach, a human is the final arbiter of ambiguity. The way Dowser is applied requires Dowser to have neither perfect recall nor prefect precision.

To evaluate the effectiveness of our approach, we have implemented a prototype of Dowser and have tested it on a widely used example SRS, describing an elevator system [11]. The case study demonstrates the effectiveness of the approach. Since identifying domain terminology is required for any successful NLP-based approach, we conducted separate studies to evaluate Dowser's SDT detection. The studies show that our approach is capable of achieving high recall and precision when detecting DSTs in a UNIX manual page.

Section 2 discusses related work. Section 3 describes out approach and all of its components. Section 4 describes validating case studies, and Section 5 discusses the results and concludes the paper.

2. Related Work

Related work can be divided into four parts: (1) NLP on SRSs, (2) controlled languages, (3) automatic object-oriented (OO) analysis model (OOAM) extraction, and (4) domain-specific term (DST) extraction.

NLP on SRSs. Kof describes a case study of the application of NLP to extract and classify terms and then to build a domain ontology [14]. This work is the most similar to our approach. The built domain ontology consists of nouns and verbs, which constitute the domain's concepts. In the end, the domain ontology helps to detect weaknesses in the requirements specification. Gervasi and Nuseibeh describe their experiences using lightweight formal methods for the partial validation of NL SRSs [9]. They check properties of models obtained by shallow parsing of natural language requirements. Furthermore, they demonstrate scalability of their approach with a NASA SRS.

Controlled languages. Fuchs and Schwitter developed Attempto Controlled English (ACE) [6], a sublanguage of English whose utterances can be unambiguously translated into first-order logic. Over the years, ACE has evolved into a mature controlled language, which is used mainly for reasoning about SRSs [7]. Juristo *et al* developed other controlled languages, SUL and DUL [13]. For these languages, they defined a correspondence between linguistic patterns and conceptual patterns. After a SRS has been written in SUL and DUL, an OOAM can be created using the correspondence.

OOAM Extraction. Several tools exist that automatically transform a SRS into an OOAM. Mich's NL-OOPS [17] tool first transforms a parsed SRS into a semantic network. Afterwards, the tool derives an OOAM from the semantic network. Barker and Biskri implemented a tool that uses only syntactic extraction rules [3]. Harmain and Gaizauskas implemented another syntax-based tool, and they introduce a method to evaluate the performance of any such tool [11]. As in our approach, Nanduri and Rugaber [21] have used the same parser and had the same initial idea of using syntactic knowledge to transform a NL SRS into an OOAM. The objective of their approach was to validate a manually constructed OOAM. Our approach's main objective is to identify ambiguity, inconsistency, and underspecification in a NL SRS. The more restricted objectives of our approach enables a more detailed discussion of the problem space and contributes (1) a constraining grammar, (2) analysis interpretation guidelines, (3) additional transformation rules, and (4) DST extraction.

DST extraction. Mollá et al. developed a method for answering questions in any specified technical domain. This

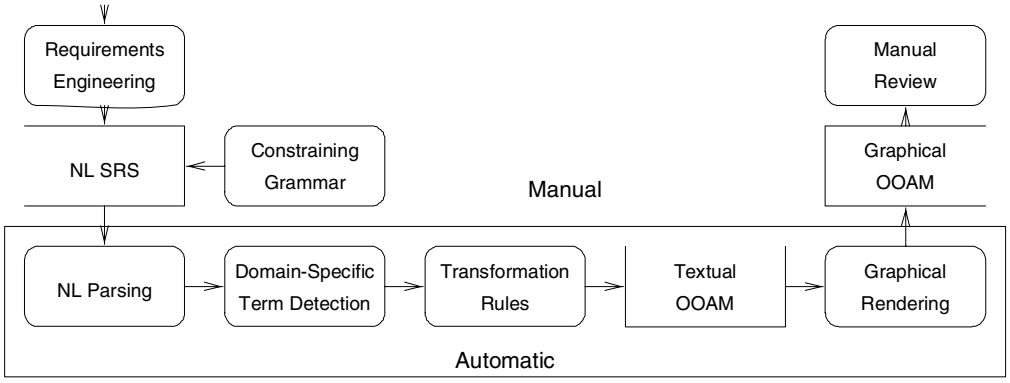


Figure 1. Flow of the Approach

work recognizes the importance of dealing with specified technical terminologies in NLP tools that are applied to SRSs [20].

3. Our Approach

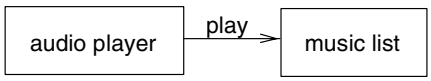
The goal of our approach is to reduce the number of ambiguities, inconsistencies, and underspecifications in a NL SRS through automation. Assuming that automation will not be perfect, we let a human make the final decision about a potential ambiguity, inconsistency, or underspecification.

While reading through a NL SRS, an engineer usually builds a mental model of the described system and reasons about the correct relations of the SRS's concepts. If an engineer could only analyze the correctness of a model, instead of having also to create it, write it down, and then analyze it, he could use his skills and time more effectively.

Considering that a reviewer could more effectively inspect a model than the complete NL SRS, we developed an automatic approach based on the following observations:

- Most software companies use a NL SRS [16] to describe software system regardless of its domain.
- An OOAM is able to show the most important concepts and relations among the concepts of a system to build.
- Many an OO design method suggests building an OOAM of a sentence by identifying the parts of the sentence and creating a class from the subject, attributes from the adjectives, and methods and associations from the verb. [2, 23]

Consider the example transformation of the sentence[1] `The audio player shall play the music list.` into an OOAM. `Audio Player` is the subject of the sentence, `play` is the verb, and `music list` is the direct object. This sentence could therefore be modeled:.



In this example, the adjectives `audio` and `music` are not broken out, because each is part of a DST.

Using this syntax-based method, the typical functional requirement sentence of a NL SRS can be transformed into an OOAM. Since this heuristic suggests using mostly syntactic information, the transformation can be automated. A NL parser can create parse trees of any NL text [24]. The OO design literature gives many rules and heuristics to transform many a syntactic construct into a textual OOAM. Off-the-shelf (OTS) software exists to diagram textual OOAMs [25].

Since NL SRSs are written for a wide range of domains such as medical, technical or judicial domains, a successful approach must be robust in identifying DSTs. Our approach addresses this need by using syntactic information and a robust parser with guessing capability.

The overall quality of any NL SRS can be improved by enforcing the use of a constraining grammar. A constraining grammar reduces the possibilities of ambiguities by constraining the allowed language constructs. At the same time, it increases the quality or parsing, resulting in more precise OOAMs.

Therefore, by using and extending existing technology, we can create a tool that automatically transforms a NL SRS into an OOAM that helps a human being to identify ambiguities, inconsistencies, and under specifications in the NL SRS.

Figure 1 shows the flow of our approach. First, Dowser parses a NL SRS according to a constraining grammar. Second, from relationships exposed in the parse, Dowser creates the classes, methods, variables, and associations of an OOAM of the specified system. Third, the OOAM is diagrammed so that a human reviewer can use the model to detect ambiguities, inconsistencies, and underspecifications.

1. A typewriter typeface is used for example text. Beware of punctuation, also typeset in a typewriter typeface, at the end of any example. It should not be considered as punctuation in the containing sentence, which is typeset in a serifed typeface.

3.1. Constraining Grammar

Any NL allows expressing the same concept in different ways, and therefore, the NL is inherently ambiguous. The same meaning can be expressed with different syntactic structures. For example, a sentence in active voice can be translated into passive voice without changing the semantics or pragmatics. However, passive voice can encourage ambiguities. For example, the sentence `The illumination of the button is activated.` leaves room for different interpretations, because it is not clear who holds the responsibility for activating the illumination. Alternatively, the sentence could be describing a state. As a consequence, a constraining grammar can be introduced to decrease the possibility of ambiguity. A constraining grammar enables formal reasoning without the disad-vantages of a fully formal language [7]. A constraining grammar has the other advantage that it is more amenable to parsing, and extraction rules based on it can be created more easily.

Our approach uses a constraining grammar that is derived from Juristo *et al*'s grammar [13]. They have developed two context-free grammars and an unambiguous mapping from these grammars to OOAMs. This mapping is explicitly defined and allows better model creation than with commonly used heuristics that are justified only intuitively. Moreover, the explicit definition enables automation.

Using a constraining grammar influences the style of a NL SRS, because a constraint grammar enforces simple sentences. The typical sentence has a basic structure consisting of subject, verb and object. Furthermore, only simple subclause constructions are allowed, such as conditional clauses, using `if`, `when`, velc. Therefore, a NL SRS will contain many short sentences if it is written according to the developed controlled grammar. Shorter, simpler sentences tend to be less ambiguous, because at the very least, they avoid some coordination and scope ambiguities

3.2. Natural Language Parsing

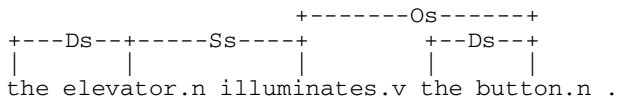
Since an OOAM is created automatically from syntactic information, we needed a parser to extract this information from the NL SRS. The parser we used was developed by Sleator and Temperley (S&T) at Carnegie-Mellon University [24]. Sutcliffe and McElligott showed that the S&T parser is robust and accurate for parsing software manuals [26]. Since software manuals are similar to SRSs [4], the S&T parser looked promising for our approach. Additionally, the S&T parser was chosen because it is able to guess the grammatical role of unknown words. This capability is used for the DST detection, which is described in Section 3.4

The parser is based on the theory of link grammars, which define easy-to-understand rule-based grammar systems. A link grammar consists of a set of words, i.e., the terminal symbols of the grammar, each of which has one or more linking requirements. A sequence of words is a sentence of the language defined by the grammar if there exists a way to assign to the words links that satisfy the following three conditions:

1. Planarity: the links do not cross;
2. Connectivity: the links suffice to connect all the words of the sequence together; and
3. Satisfaction: the links satisfy the linking requirements of each word in the sequence.

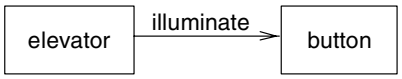
The link grammar parser produces the links for every such sentence. After parsing, the links can be accessed through the API of the link grammar parser.

Each established link in a sentence has a link type, which defines the grammatical usage of the word at the source of the link. The sentence `The elevator illuminates the button.` shows three different link types:



A `D` link connects a determiner to a noun, an `S` link connects a subject noun to its finite verb, and an `O` link connects a transitive verb to its object. A small `s` after the type of a link indicates that the target of the link is a singular noun.

From this example sentence, the first extraction rule can be derived. If a sentence contains an `S` link and an `O` link, then create a class from the subject noun and one from the object noun. Afterwards, create a directed association from the subject to the object class, which is named by the verb:



A directed association was chosen over a simple class method, because an association shows who invokes the action. If the action were modeled as a class method, the information about who causes the action would have been lost. Using this rule, Dowser would extract the classes elevator and button and a directed association illuminate from the elevator class to the button class.

To avoid having two different classes created for elevator and elevators, our approach incorporates the lexical reference system *WordNet* [19] to find the stems of nouns and verbs. Therefore, the name of each created class is the stem of a noun.

3.3. Transformation Rules

Transformation rules bridge the gap between the extracted syntactic sentence information and the targeted OOAM. Each transformation rule describes how a combi-

nation of words of identified grammatical roles can be transformed into classes, associations, attributes, and methods.

The used extraction rules were derived from Juristo *et al*'s grammar [13], the OO methods literature [21, 23], and conducted experiments. In total, the system uses 13 extraction rules. The five most frequently used are the following:

1. The most frequent applicable rule is the above described rule. If a parsed sentence contains a subject and an object link, then create two classes with an directed association named after the verb.
2. Aggregations are an important notion in UML class diagrams. One rule for extracting aggregations is similar to the first rule. The major difference is the verb. This rule is applicable only if the parsed sentence contains a subject and an object link and the verb stem is one of `have`, `possess`, `contain`, or `include`. In that case, the object is aggregated to the subject.
3. Sometimes, a subclause describes a system action without the need of an object, particularly, if the system reacts to a given event, e.g., `If the user presses the button, the elevator moves.` An event clause starts with an `if` or `when`. If the system detects an event clause, and the main clause has only a subject link, then a class from the subject link noun is created and the verb is added to the new class as a method.
4. A genitive attribute indicates two classes with an aggregation, e.g., `The system stores the name of the customer.` or `The system stores the customer's name.` If Dowser detects a genitive, it creates two classes with one linking aggregation. For either example, Dowser would create a class customer, and it would aggregate the class name to the class customer. name could have been modeled as an attribute. However, other sentences in the specification would add methods to the class `name` later. Therefore, with the syntactic information of only one sentence, it cannot be decided if `name` is an attribute or an aggregated class. This rule needs to be constrained by semantic information. For example, Dowser should not apply this rule to the sentence `The user enters the amount of money.` Although `amount` is a noun, the class amount is not desired in this case.
5. Although active clauses are preferred in NL SRSs, passive clauses are still needed. They are used to describe relations and states, e.g. as in `A husband is married to his wife.` From this sentence, two classes are created, from the subject noun and the noun of the prepositional phrase. The passive verb and the connecting word `to` link the prepositional phrase described with the association.

Two post-processing rules are applied after all possible extraction rules are executed.

The first post-processing rule converts all classes that are aggregated to another class into attributes of that other class. Only a class that lacks any attribute, method, or incoming or outgoing association is transformed. For example, one rule described above extracts two classes and one aggregation from the sentence `The system stores the name of the customer.` The rule creates a class name and a class customer. However, the class name has probably no method or association. Therefore, if a class contains no method after all rules were applied, it is transformed into an attribute of the class customer.

The second post-processing rule removes the class system from the OOAM, since all other classes together form the system; system is not a class, because it cannot be a subpart of itself.

The full set of rules, a user's manual for Dowser, and other details may be found at the Web site:

http://www.cc.gatech.edu/projects/dowser/ .

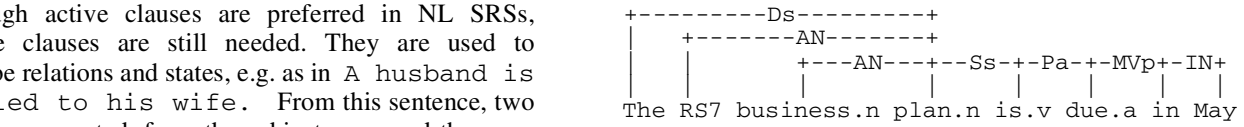
3.4. Domain-Specific Terms

In order to build the correct classes into the OOAM of a NL SRS, our approach has to be able to detect DSTs. If Dowser were to extract only the concept button from `elevator button`, Dowser would be identifying an incorrect term.

To achieve high DST recall, the parser could access a special domain data dictionary. However, for each of most domains, such a data dictionary does not exist. Software is built for financial, medical, technical, and other domains. Creating a domain dictionary for each of these rich domains is complex and difficult. Additionally, the customer of a software application might prefer to use her own terms for describing her product. A product could have arbitrarily chosen names, such as "DNAComp07". Therefore, even if a domain dictionary exists for a NL SRS, DST detection remains a challenge.

The link types of the link grammar parse can be used to identify DSTs. The typical DST happens to be built from an attributive noun or a proper noun. Therefore, in a link grammar parse, the `AN` link, denoting an attributive noun, and the `G` link, denoting a proper noun, help to identify DSTs.

Consider the link grammar parse of the sentence `The RS7 business plan is due in May.`



Thus, `RS7 business plan` is a domain-specific term.

A parser typically has problems parsing words that are not in its internal dictionary. However, the S&T link grammar parser has a guessing mode, in which it can guess the syntactic role of an unknown term. Therefore, it is often able to guess the syntactic role of an unknown term, improving its DST recall.

Since DST detection is essential for transforming a NL SRS into an OOAM, we conducted a study, described in Section 4.2, about the recall and precision of our approach.

3.5. Diagramming OOAMs

The previous steps are able to create a textual OOAM. However, it is easier for a human to understand a graphical OOAM than a textual OOAM. Using the approach described by Spinellis[25], an extracted textual OOAM is diagrammed. The tool *UMLGraph* transforms a textual description into a *dot file*, which the tool *Graphviz* can transform into any of some popular graphic formats, such as JPEG, GIF, or PNG.

3.6. Interpretation of OOAM

In the last step of our approach, a human analyst checks the created diagram for ambiguities.

Some ideas that a human analyst can use to find defects in an OOAM are:

- An association is a hint for possible ambiguities. For example, suppose that each of two different classes sends a message to the same target class. The analyst should check that the two different classes actually are to communicate with the same target class. If a motion sensor activates one type of display, and a smoke detector actives another type of display, then the class diagram should reflect this situation with two different display classes.
- Each class should reflect one concept. For example, the analyst should check that book and textbook are really two different classes when Dowser does not create a generalization of these two classes.
- If a class has an attribute, but the attribute is not of a primitive type, such as string or number, then the definition of the attribute might be missing in the original text. After a definition is added, the attribute should be represented by its own class.
- If a class has no association, then the class might be underspecified, as there are no relations or interactions between the class and other classes.

3.7. Limitations of Method

Observe that the OOAM is a model of only static relationships among the concepts mentioned in the parsed NL SRS. Modeling dynamic behavior is not possible.

4. Studies

This section describes the case studies in which we evaluated the effectiveness of our approach in helping an analyst to identify ambiguities, inconsistencies, and underspecifications in a NL SRS and in which we evaluated Dowser's effectiveness at DST identification.

4.1. Elevator Case Study

To evaluate the effectiveness of our approach, we implemented Dowser and applied it to an example NL SRS that we call "the ESD". The ESD describes the control software for an elevator system [12]. The ESD was chosen, because it could be the NL SRS of a real industrial system. At the same time, the ESD is short enough to completely described in this paper. Moreover, the ESD happens to contain enough defects that it illustrates the defect types that can be revealed with the help of Dowser.

The original ESD was:

An n elevator system is to be installed in a building with m floors. The elevators and the control mechanism are supplied by a manufacturer The internal mechanisms of these are assumed (given) in this problem.

Design the logic to move elevators between floors in the building according to the following rules:

1. *Each elevator has a set of buttons, one button for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is canceled when the corresponding floor is visited (i.e., stopped at) by the elevator.*
2. *Each floor has two buttons (except ground and top), one to request an up-elevator and one to request a down-elevator. These buttons illuminate when pressed. The buttons are canceled when an elevator visits the floor and is either traveling the desired direction, or visiting a floor with no requests outstanding. In the latter case, if both floor request buttons are illuminated, only one should be canceled. The algorithm used to decide which to serve first should minimize the waiting time for both requests.*
3. *When an elevator has no requests to service, it should remain at its final destination with its doors closed and await further requests (or model a "holding" floor).*
4. *All requests for elevators from floors must be serviced eventually, with all floors given equal priority.*
5. *All requests for floors within elevators must be serviced eventually, with floors being serviced sequentially in the direction of travel.*

First, we applied Dowser tool to the original unmodified ESD, which was not written according to any constraining grammar. Since the transformation rules have been created assuming that the analyzed text conforms to a constraining grammar, applying Dowser to the original ESD resulted in a diagram with only five classes such as these and set. None of these classes describes any domain concepts of the ESD.

To successfully apply Dowser to the ESD, the ESD had to be rewritten sentence-by-sentence to conform to the constraining grammar. No information was added or removed from the original ESD during the rewriting. Therefore, the rewriting did not introduce any new defects, which would have adulterated the results of the case study.

The rewritten ESD is:

An n elevator system is to be installed in a building with m floors.

1. *Each elevator has buttons. Each elevator has one button for each floor. When a user presses a button, the elevator illuminates the button and the elevator visits the*

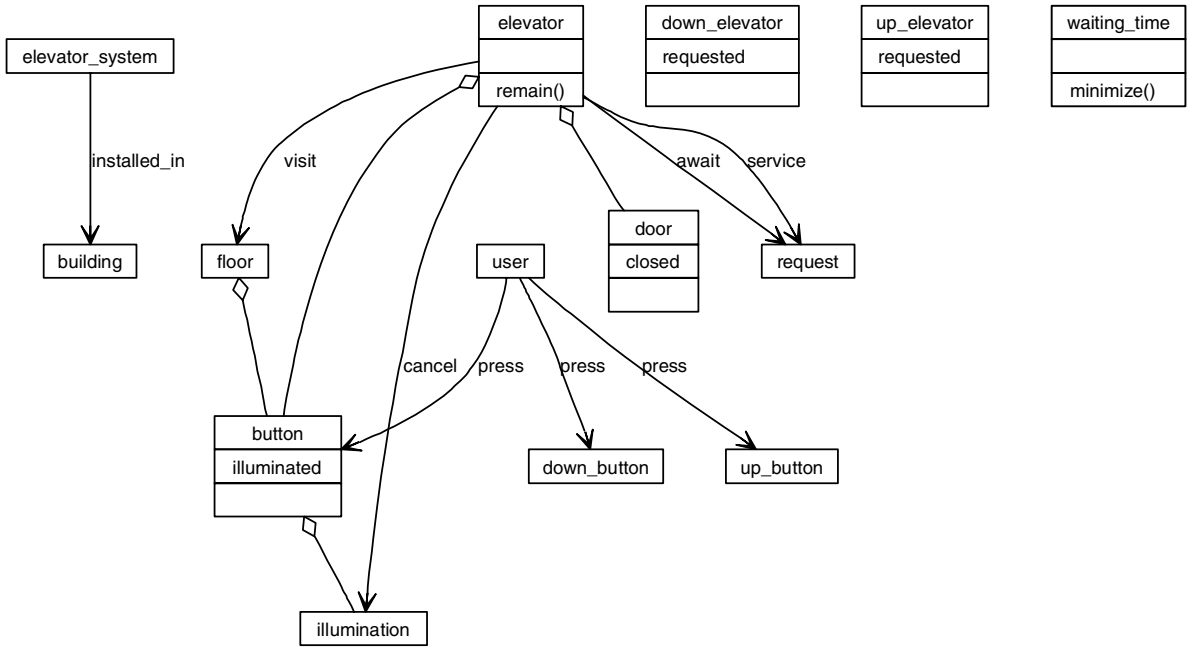


Figure 2. OOAM of ESD

corresponding floor. When the elevator visits a floor, the elevator cancels the corresponding illumination.

2. *Each floor has two buttons. (except ground and top). If the user presses the up-button, an up-elevator is requested. If the user presses the down-button, a down-elevator is requested. If the user presses a button, this button becomes illuminated. When an elevator visits a floor, the elevator cancels the corresponding illumination of the button in the desired direction. The system minimizes the waiting time.*
3. *When an elevator has not to service any requests, the elevator remains at its final destination and the doors of the elevator are closed. The elevator then awaits further requests.*
4. *The elevators service all requests from floors with equal priority eventually.*
5. *If a user presses a button within the elevator, the elevator services this request eventually in the direction of travel.*

Applying Dowser to the new ESD resulted in the diagram of Figure 2. Dowser created an OOAM containing 14 partially connected classes with attributes and methods.

The graphically rendered OOAM does not reveal defects on its own. By applying the guidelines described in Section 3.6, we identified four conceptual defects in the OOAM, which could be traced back to the original ESD.

1. The diagram shows classes up-elevator and down-elevator. Neither class has any connection to any other class, and neither has any association to the class elevator. Furthermore, the up-elevator has an attribute requested, while `elevator serves a request` indicates that each of up-elevator and down-elevator is a specialization of elevator. All this indicates that neither concept, up-elevator or down-elevator, is defined enough in the original ESD.
2. The class door in the diagram contains the attribute closed. However, the class has no method to manipulate this state. If closing and opening the door are within the scope of the system, then it is clear that the concept door is not defined enough in the original ESD.
3. In the ESD, each of the floor and the elevator has buttons. Therefore, each of the class elevator and the class floor should have an aggregated class button in the OOAM. However, the diagram indicates that both have the same type of button. Since a button in an elevator and a button in a floor have different behaviors, it is unlikely that the same class describes both types of buttons. Generalizing both types to a single class button could therefore lead to misinterpretations. Defining concepts elevator button and floor button would resolve this ambiguity and enhance the clarity of the ESD.
4. Each of the classes up-button and down-button is connected to only the class user in the OOAM. Since a user is an actor in the system, the diagram does not clarify where button belongs. The location can be derived from ESD, because `button` is mentioned in the paragraph that mentions `floor`. However, it should not be necessary to use this fact. Therefore, each concept should be specified in more detail in ESD to reduce the ambiguity in the specification.

This case study shows how Dowser can help an analyst identify defects in a NL SRS. If a constraining grammar is used to write a NL SRS, our approach can help detect ambiguities, inconsistencies, and underspecifications.

4.2. DST Detection Quality

The case study in Section 4.1 shows the importance of

detecting DSTs. For the ESD, Dowser needed to be able to detect DSTs such as floor button, elevator button, or down elevator. If Dowser were to extract only the terms button and elevator from the ESD, it would create wrong classes and associations.

Section 3.4 explains how Dowser relies on syntactic information to identify DSTs. To measure Dowser's DST detection capability, we conducted a separate study.

To measure Dowser's DST detection, the metrics *recall* and *precision* were calculated of Dowser's extraction of DSTs from a user's manual. These metrics are used to evaluate the success of many NLP tools [10] :

- *Recall* measures how well a tool identifies desired pieces of information in a source:

$$Recall = \frac{I_{correct}}{I_{total}}$$

  Here $I_{correct}$ is the number of correctly identified desired pieces of information in the source text and I_{total} is the total number of desired pieces of information in the source text.
- *Precision* measures how accurately a tool identifies desired pieces of information in a source:

$$Precision = \frac{I_{correct}}{I_{correct} + I_{incorrect}}$$

  Here $I_{correct}$ is as for *Recall* and $I_{incorrect}$ is the number of incorrectly identified desired pieces of information in the source text.

We chose the *intro man page* of the *Cygwin environment* with which to measure recall and precision of Dowser's DST detection. A manual page seems to be a suitable experiment source, because it is a technical document with a large number of DSTs.

The steps of the experiments are as follows.

1. We manually identified every DST, a noun or a compound noun, from the *intro man page*. The *intro man page* contains 52 terms like `Cygwin`, `Linux-like environment`, `Linux API emulation layer`, and `POSIX/SUSv2`. The 52 terms consists of 33 single-noun terms and 19 compound-noun terms.
2. For the first experiment, the link grammar parser extracted every term out of the *intro man page* without the capability of extracting compound-noun terms. It recognized 31 of the single-noun terms and none of the compound-noun terms. Therefore, it reached a recall value of 59.6% for all terms and of 93.9% for single-noun terms.
3. For the second experiment, compound-noun term detection was added to the link grammar parser. After this, the tool recognized 10 compound-noun terms. As the single-noun terms detection rate stayed the same, the tool recognized 41 terms. Therefore, it reached a recall value of 78.8%.

Afterwards, the undetected terms were examined. It turned out that five terms were undetected because they appeared in grammatically obscure or wrong parts in the sentence. Correcting these sentence, increased the detected terms to 46 and the recall value to 88.46%.

The five not identified terms were (1) `case-insensitive file system`; (2) `intro man page`; (3) `Linux look and feel`; (4) `Red Hat,Inc.`; and (5) `User's Guide`. The term `Red Hat,Inc.` is not recognized because of the comma, `User's Guide` cannot be detected syntactically, because if every genitive were part of a term, it would lead to an over-generation of terms. `Linux look and feel` is not recognized because of the conjunction *and* in the term. `Case-insensitive file system` and `intro man page` can be only partially detected, because `case-insensitive` is an adjective, which is only sometimes part of a term. Another example demonstrates the difficulties caused by adjectives. In `readable manual`, only `manual` is a term in most cases. Using every adjective as part of the term would lead to an overgeneration of terms. The term `intro man page` is not recognized, because the link grammar parser guesses that `intro` is an adjective. However, if it is planned that `case-insensitive file system` is a concept within a SRS, then writing it with initial upper-case letters would allow the link grammar parser to detect it as a proper noun, and thus as a DST.

Dowser extracted seven wrong terms, since it created wrong terminology for the incompletely detected terms, e.g., it extracted the term `man page` instead of `intro man page`. Overall, Dowser reached a precision value of 86.79% on the *intro man page*.

The DST detection experiment shows that using only syntactic information from the link grammar parser allows a fairly high DST detection rate.

5. Discussion and Conclusion

Dowser's effectiveness in helping a human analyst to identify ambiguity, inconsistency, and underspecification in a NL SRS and its recall and precision in identifying DSTs in a user's manual are not bad considering that Dowser was built mostly our of existing software.

Of course, Dowser's lack of perfection, particularly in the construction of an OOAM and in the DST recall, says that Dowser can be used as only one of an array of approaches and tools for identifying DSTs and for detecting ambiguity, inconsistency, and underspecification in NL SRSs. However, because of the inherent difficulty of these tasks for humans, every little bit helps!

5.1. Future Work

The lack of perfection says that more work is needed:

- How does Dowser perform on larger, industrial-strength NL SRSs? Answering this question would help to explore

the problem space and to find new unsolved research questions.

- The current Dowser cannot resolve anaphora. An anaphor is a linguistic unit, such as a pronoun, that refers to a previous unit. In `The customer can buy text books and return them.`, `them` is an example of an anaphor, which must be resolved to `text books`. While Dowser can identify anaphora, it cannot resolve them. A simple solution would be to have Dowser flag all anaphora in its input text, so a human analyst could change each to its resolution.
- DST identification can be improved. As mentioned above, syntactic information is not sufficient to detect all the DSTs within a document. Therefore, frequency analysis or baseline text analysis[15] might improve DST identification.
- Additional semantic knowledge could improve the capability of Dowser. For example, the *WordNet* lexicon contains information about hypernyms, which can indicate superclasses, and meronyms, which can indicate aggregations. This information could be used to show missing links in an OOAM.
  However, although *WordNet* is a large online lexicon, it lacks DSTs and therefore might be only a little help. Extending Dowser's dictionary with DSTs could reduce this problem.
- The current Dowser is not applicable to a NL SRS that has a functional language style, i.e., with sentences such as, `The system must provide the functionality of....` Handling such sentences would require a different grammar. Future work could examine which grammar is the most suitable for class extraction.
- The UML offers a set of different diagram types. NLP could be used to create sequence, state, or other diagrams. For example, Juristo *et al* [13] developed also a controlled grammar for specifying dynamic behavior.
- Other work has found different sources of ambiguities in NL SRS. Since there seems not to be a single perfect approach, different approaches could be integrated into a single framework for validating NL SRSs. This integration could lead to a new method of developing NL SRSs.

References

[1] IEEE Standard for software reviews and audits, Soft. Eng. Tech. Comm. of the IEEE Computer Society, 1989, IEEE Std 1028-1988.

[2] R. J. Abbott. Program design by informal English descriptions. Communication of the ACM, 26(11):882–894, 1983.

[3] S. Delisle, D. Barker and K. Biskri. Object-oriented analysis: Getting help from robust computational linguistic tools. in G. Friedl, H.C. Mayr (eds) Application of Natural Language to Information Systems, Oesterreichische Computer Gesellschaft, pp. 167-172, 1999.

[4] D. M. Berry, K. Daudjee, J. Dong, I. Fainchtein, M.A. Nelson, and T. Nelson. User's Manual as a Requirements Specification: Case Studies. Requirements Engineering Journal, 9(1): 67-82, February 2004.

[5] D.M. Berry and E. Kamsties: Ambiguity in Requirements Specification, Perspectives on Requirements Engineering, J.C.S.P. Leite and J. Doorn, eds., Boston, MA, Kluwer, pp. 7–44, 2004.

[6] N. Fuchs and R. Schwitter. Attempto controlled English (ACE). In The First International Workshop On Controlled Language Applications (CLAW), Belgium, 1996.

[7] N. E. Fuchs, U. Schwertel, and R. Schwitter. Attempto controlled english (ACE) language manual, version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich, August 1999.

[8] D. Gause and G.Weinberg. Exploring requirements: Quality before design, 1989.

[9] V. Gervasi and B. Nuseibeh. Lightweight validation of natural language requirements. Softw., Pract. Exper., 32(2):113– 133, 2002.

[10] R. Grishman. Information extraction: Techniques and challenges. In SCIE '97: International Summer School on Information Extraction, pp. 10–27, London, UK, 1997. Springer-Verlag.

[11] H. M. Harmain and R. J. Gaizauskas. CM-builder: A natural language-based CASE tool for object-oriented analysis. Autom. Softw. Eng., 10(2):157–181, 2003.

[12] M. Heimdahl. An example: The lift (elevator) problem. http://www-users.cs.umn.edu/heimdahl/formalmodels/elevator.htm, accessed on 14.12.2005.

[13] N. Juristo, A. M. Moreno, and M. Lopez. How to use linguistic instruments for object-oriented analysis. IEEE Softw., 17(3):80–89, 2000.

[14] L. Kof. Natural Language Procesing for Requirements Engineering: Applicability to Large Requirements Documents. In A. Russo, A. Garcez, and T. Menzies, eds., Automated Software Engineering, Proceedings of the Workshops, Linz, Austria, September 21, 2004.

[15] R. Lecoeuche. Finding Comparatively Important Concepts between Texts. Proceedings of the 15th IEEE international Conference on Automated Software Engineering, Grenoble, France, 2000.

[16] L. Mich, M. Franch, and L. N. Inverardi. Market research for requirements analysis using linguistic tools. Requir. Eng., 9(2):151–151, 2004.

[17] L. Mich and R. Garigliano. NL-OOPS: A requirements analysis tool based on natural language processing. In Proceedings of Third International Conference on Data Mining Methods and Databases for Engineering, Bologna, Italy, 2002.

[18] L. Mich and R. Garigliano. Ambiguity measures in requirements engineering, Proceedings of International Conference on Software---Theory and Practice (ICS2000), Sixteenth IFIP World Computer Congress, Beijing, China, pp. 39--48, August 2000.

[19] G. A. Miller, C. Felbaum, et al. WordNet Web Site. Princeton University, Princeton, NJ, USA, accessed 12 March 2006, http://wordnet.princeton.edu/

[20] D. Mollá, R. Schwitter, F. Rinaldi, J. Dowdall, and M. Hess. NLP for answer extraction in technical domains. Proceedings of the EACL 2003, Workshop on Natural Language Processing (NLP) for Questions Answering,, Budapest, Hungary, pp. 5-12, April 14, 2003.

[21] S. Nanduri and S. Rugaber. Requirements validation via automated natural language parsing, Journal of Management Information Systems 12(3):9-19, Winter 1995-96.

[22] K. Ryan. The role of natural language in requirements engineering,, Proceedings of the International Symposium on Requirements Engineering, pp. 240-242, San Diego, CA, January 1993.

[23] J. Rumbaugh. Object-Oriented Modeling and Design. Englewood-Cliffs, NJ, Prentice Hall, 1991.

[24] D. D. Sleator and D. Temperley. Parsing English with a link

grammar. Proceedings of the Third International Workshop on Parsing Technologies, http://www.link.cs.cmu.edu/link/papers/index.html, 1993.
[25] D. Spinellis. On the declarative specification of models. IEEE Software, 20(2):94-96, March/April 2003.
[26] R. Sutcliffe and A. McElligott. Using the link parser of Sleator and Temperley to analyse a software manual corpus. In Industrial Parsing of Software Manuals, pp. 89-102, 1995.