# Rule-Based Extraction of Goal-Use Case Models from Text

Tuong Huan Nguyen, John Grundy, Mohamed Almorsy
Faculty of Science, Engineering and Technology
Swinburne University of Technology
Melbourne, Australia
{huannguyen, jgrundy, malmorsy}@swin.edu.au

## ABSTRACT

Goal and use case modeling has been recognized as a key approach for understanding and analyzing requirements. However, in practice, goals and use cases are often buried among other content in requirements specifications documents and written in unstructured styles. It is thus a time-consuming and error-prone process to identify such goals and use cases. In addition, having them embedded in natural language documents greatly limits the possibility of formally analyzing the requirements for problems. To address these issues, we have developed a novel rule-based approach to automatically extract goal and use case models from natural language requirements documents. Our approach is able to automatically categorize goals and ensure they are properly specified. We also provide automated semantic parameterization of artifact textual specifications to promote further analysis on the extracted goal-use case models. Our approach achieves 85% precision and 82% recall rates on average for model extraction and 88% accuracy for the automated parameterization.

## Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirement/Specifications – *languages, methodologies, tools*.

## General Terms

Algorithms, Languages.

## Keywords

Goal-Use Case modeling, extraction, semantic parameterization.

## 1. INTRODUCTION

Requirements Engineering (RE) is an iterative process of eliciting, structuring, specifying, analyzing, and managing requirements of a software system [28]. The functionality and constraints of a target system identified in each RE iteration are usually captured in a textual software requirements specification document (SRS). Goal-Use case integration modeling (GUIM) [17, 29] has been recognized as a key approach for understanding, organizing, justifying and analyzing requirements, and facilitating early system designs [17]. GUIM helps capturing the underlining rationale and motivation of the system being developed while aligning the business objectives with the functionalities and constraints of system components. The details of system-user interactions (use cases) are also modeled and linked to system goals. Such a combination enables GUIM to provide a comprehensive view of the system [1].

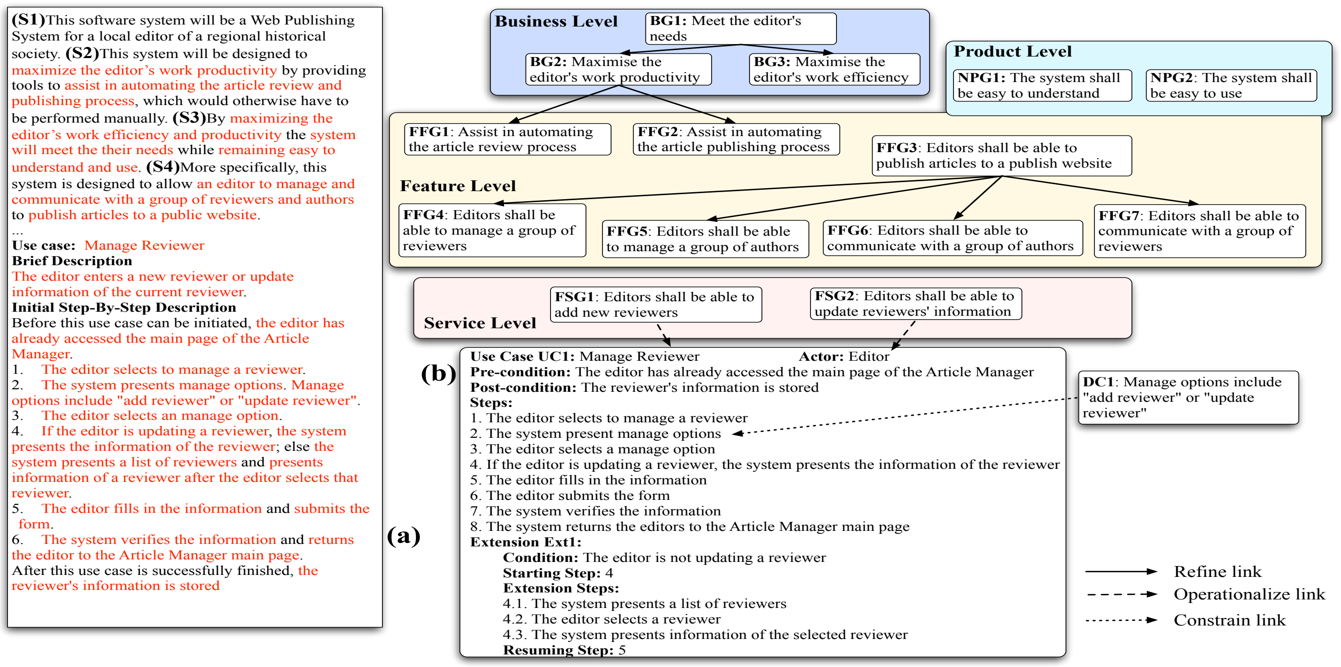
However, extracting and modeling goals and use cases from SRSs are not trivial tasks. Domain experts often find it difficult to formulate and express goals at the required abstraction levels [26]. In textual requirements documents, goals are normally buried among other (non-goal) sentences and written in unstructured styles. Furthermore, frequently use cases descriptions are not clear in requirements documents. Multiple use case steps may be combined as one (i.e., by conjunctions). Moreover, data or non-functional constraints are often mixed up with use case steps, making it hard to locate the information they need. Due to such complexities, manual goals and use cases modeling can be a tedious, time-consuming and error-prone process, especially for inexperienced requirements engineers and large requirements documents. In addition, having goals and use cases embedded in natural language documents greatly limit the capability of the automatic requirements analysis for quality problems. In fact, such automated analysis support requires requirements to be expressed in formal specifications [30] or semantically parameterized [21, 22] so that their contents can be processed by computers.

For these reasons, we propose a novel approach with tool support named **G**oal-**U**se case model **E**xtraction **S**upporting **T**ool (GUEST) to automatically extract goal and use case models from requirements specification documents. GUEST is part of our **G**oal and **U**se case **I**ntegration **F**ramework (GUI-F) that supports the elicitation and analysis of goal-use case integration models. Our technique is based on a set of extendable extraction rules that help identify goals, use cases and their relationships from texts. Moreover, relying on our goal-use case integration meta-model that provides classification and specification rules of goals based on their levels of abstraction and quality attributes, GUEST is able to automatically categorize goals and ensure they are properly specified. Furthermore, GUEST provides automated semantic parameterization of textual artifact specifications to enable the automatic analysis of extracted goal-use case models in our GUI-F framework. This paper makes the following key contributions:

(1) A rule-based approach to automatically extract goal-use case models from software requirements specifications. The extraction carries out the identification of goals, use cases (including use case steps, pre/post conditions, data or non-functional constraints) and their relationships from texts, categorization of goals, and the guaranty of proper artifact specifications in extracted goal-use case models.

(2) A technique to automate the semantic parameterization of textual artifact specifications to allow model analysis.

(3) Validating our approach with various requirements from both literature and industry. We achieved the precision and recall rates of 85% and 82% (on average) respectively for goal-use case model extraction and 88% accuracy rate for the automated semantic parameterization of textual artifacts.


(a)
(b)
Business Level
BG1: Meet the editor's needs
BG2: Maximise the editor's work productivity
BG3: Maximise the editor's work efficiency
Product Level
NPG1: The system shall be easy to understand
NPG2: The system shall be easy to use
Feature Level
FFG1: Assist in automating the article review process
FFG2: Assist in automating the article publishing process
FFG3: Editors shall be able to publish articles to a publish website
FFG4: Editors shall be able to manage a group of reviewers
FFG5: Editors shall be able to manage a group of authors
FFG6: Editors shall be able to communicate with a group of authors
FFG7: Editors shall be able to communicate with a group of reviewers
Service Level
FSG1: Editors shall be able to add new reviewers
FSG2: Editors shall be able to update reviewers' information
DC1: Manage options include "add reviewer" or "update reviewer"
Refine link
Operationalize link
Constrain link

**(S1)**This software system will be a Web Publishing System for a local editor of a regional historical society. **(S2)**This system will be designed to maximize the editor's work productivity by providing tools to assist in automating the article review and publishing process, which would otherwise have to be performed manually. **(S3)**By maximizing the editor's work efficiency and productivity the system will meet the their needs while remaining easy to understand and use. **(S4)**More specifically, this system is designed to allow an editor to manage and communicate with a group of reviewers and authors to publish articles to a public website.
...
**Use case:** Manage Reviewer
**Brief Description**
The editor enters a new reviewer or update information of the current reviewer.
**Initial Step-By-Step Description**
Before this use case can be initiated, the editor has already accessed the main page of the Article Manager.
1. The editor selects to manage a reviewer.
2. The system presents manage options. Manage options include "add reviewer" or "update reviewer".
3. The editor selects an manage option.
4. If the editor is updating a reviewer, the system presents the information of the reviewer; else the system presents a list of reviewers and presents information of a reviewer after the editor selects that reviewer.
5. The editor fills in the information and submits the form.
6. The system verifies the information and returns the editor to the Article Manager main page.
After this use case is successfully finished, the reviewer's information is stored

**Use Case UC1:** Manage Reviewer **Actor:** Editor
**Pre-condition:** The editor has already accessed the main page of the Article Manager
**Post-condition:** The reviewer's information is stored
**Steps:**
1. The editor selects to manage a reviewer
2. The system present manage options
3. The editor selects a manage option
4. If the editor is updating a reviewer, the system presents the information of the reviewer
5. The editor fills in the information
6. The editor submits the form
7. The system verifies the information
8. The system returns the editors to the Article Manager main page
**Extension Ext1:**
**Condition:** The editor is not updating a reviewer
**Starting Step:** 4
**Extension Steps:**
4.1. The system presents a list of reviewers
4.2. The editor selects a reviewer
4.3. The system presents information of the selected reviewer
**Resuming Step:** 5

**Figure 1: Example of Goal-Use Case Model Extraction From Text**

## 2. MOTIVATION

Figure 1 presents an example of a goal-use case model extracted from textual requirements. Figure 1 (a) shows some parts taken from a requirements document. Figure 1 (b) presents a desired goal-use case model to be extracted from the requirements. BG1, BG2, etc. are business goals. FFG1, FFG2, etc. are functional feature goals. FSG1, FSG2, etc. are functional service goals.

### 2.1 Identification of Artifacts from Text

Identifying artifacts from text is a key challenge because: (1) Not all sentences and not all parts of a sentence in a requirement document contain goal or use case descriptions, (2) Multiple goals or use case steps maybe mixed up in a single sentence, (3) Use case steps are often combined with constraints and (4) Alternative paths are often combined as a single step. Thus, automatically filtering important information from text is needed. So is the automatic separation of different artifacts mixed up together.

***Example 1:*** Sentence (S1) does not contain a goal description, it is rather an introduction to the system. Thus, it should be ignored.

***Example 2:*** In sentence (S2), the part *"which would otherwise have to be performed manually"* has no significance regarding the objective, functionality or quality of the system. Similarly, *"This system will be designed to"* is unimportant. In addition, S2 contains multiple phrases that can be extracted to goals BG1, FFG1 and FFG2 in Figure 1 (b). Note that the phrase *"assist in automating the article review and publishing process"* is split into two goals FFG1 and FFG2 as conjunctions (i.e., *and*) are discouraged in textual requirements to avoid ambiguity [14].

***Example 3:*** Step 2 of the use case (Figure 1(a)) contains a use case step in the first part and a data constraint (about manage options) in the second part. They need to be distinguished to guarantee a correct extraction of use cases. Additionally, step number 5 is a combination of two separated steps (*"The editor fills in the information"* and *"The editor submits the form"*).

***Example 4:*** Step 4 of the use case (Figure 1(a)) combines a use case step with an extension specification. It should be extracted into the extension Ext1 in Figure 1 (b).

### 2.2 Identification of Artifact Relationships

A goal-use case model requires the relationships between the artifacts to be specified. These relationships are often implicitly mentioned in requirements specifications.

***Example 5:*** In sentence (S2), the structure *"by providing…"* implies a refinement relationship between *"providing tools to assist in automating the article review and publishing process"* and *"maximize the editor's work productivity"*. It is then extracted as showed between BG1, FFG1 and FFG2 in Figure 1(b).

### 2.3 Classification of Goals

In goal modeling, goals need to be classified to functional or non-functional. Moreover, they need to be classified based on how abstract or concrete they are. Both such classifications are important to understanding and analyze goal models.

***Example 6:*** The goal BG1 *"Meet the editor's needs"* should be classified as a business goal and placed on the *business level* since it describes a business objective, not a functionality or quality.

***Example 7:*** The goal NPG1 *"The system shall be easy to understand"* should be classified as a non-functional goal since it describes a usability quality that the system must meet. Moreover, NPG1 should be placed on the *product level* since it is concerned about the system as a whole, rather than a specific feature.

### 2.4 Ensure artifacts are properly specified

When identifying artifacts, relevant text in the requirements document is located. However, they are often fragments of the sentences they are in and thus in many cases, cannot be used as descriptions for stand-alone artifacts. Therefore, we need to ensure those artifacts are rewritten in a sensible way after being extracted.

***Example 9:*** In sentence (S3), it is identified that *"remaining easy to understand"* contains a goal description. However, this phrase by itself is not a meaningful goal description. The context of the whole sentence needs to be considered to obtain a proper specification. As showed in Figure 1(b), this phrase is rewritten as *"The system shall be easy to understand"* to specify NPG1.

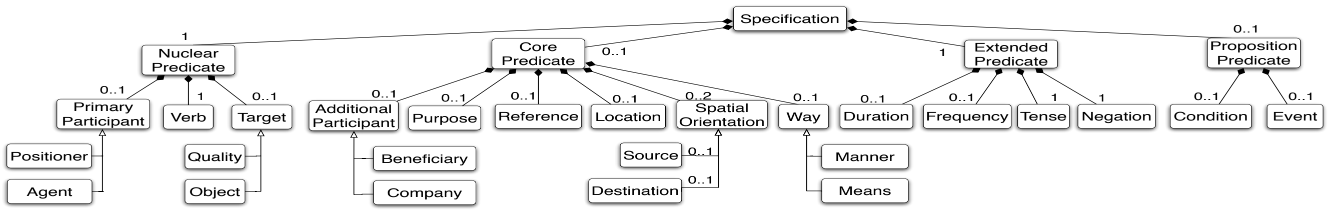


**Figure 2: The Structure of a Specification**

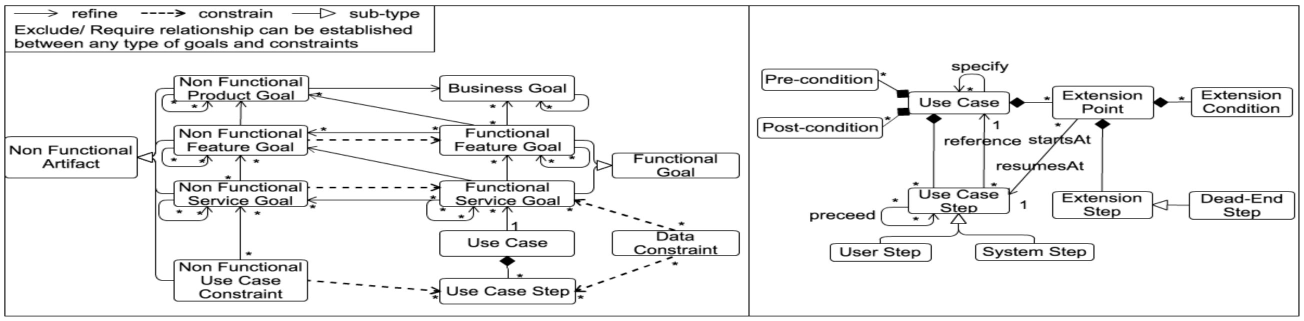


**Figure 3: Artifact Layer**

***Example 10:*** In sentence (S3), *"meet their needs"* is identified as a potential goal description. The context of the sentence is needed to recognize which noun phrase the possessive adjective *"their"* refers to. A replacement of *"their"* by *"the editor"* is necessary for a proper goal specification (BG1). This process is referred to as *coreference resolution* in the natural language processing field.

## 3. REQUIREMENTS MODEL

In this section, we discuss Functional Grammar and present our goal-use case integration meta-model on which artifact classifications and specifications are based.

### 3.1 Functional Grammar

Functional Grammar (FG) is a grammatical theory concerning the organization of natural languages [9]. In FG, a sentence contains different components with unique semantic roles called *semantic functions*. In our work, FG is the underlining theory to parameterize artifact specifications (i.e., goals, use case steps). This provides a standard way to interpret the semantic role of each group of words in a specification and thus offers a means to interpret and analyze specifications. Figure 2 presents the structure of a specification. A specification consists of four predicates. For instance, *nuclear predicate* contains elements describing which action is conducted (*verb*), on what target (*object*), etc. *Core predicate* provides details about the *beneficiary* or how an activity is performed (*manner*). Each semantic function is described by a term. For instance, nominal terms are used to describe entities while verbal terms describe activities.

***Example 11:*** The specification of goal FFG1 (in Figure 1(b)) is parameterized as *Verb(Maximize) + Object(NomTerm(Head(Work Productivity) + Possessor(Editor)))*.

***Example 12:*** The use case UC1's step 4 is parameterized as *Agent(System) + Verb(Present) + Object(NomTerm(Head(Inform -ation) + Possessor(Reviewer))) + Condition(Agent(Reviewer) + Verb(Update) + Object(Reviewer))*.

### 3.2 Goal-Use Case Integration Meta-model

Our meta-model contains two layers, the artifact layer provides the classification of artifacts while the specification layer provides specification rule for each artifact class.

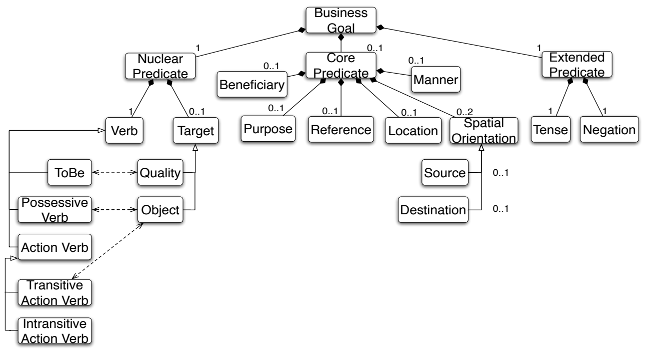


**Figure 4: Specification Rule for Business Goals**

#### 3.2.1 *The Artifact Layer*

Figure 3 depicts the key components of *artifact layer* that defines the artifact classes across levels of abstractions. For instance, *business goals* describe the business objectives of the software system (e.g., "*Maximize the editor's productivity*"). *Functional feature goals* list features the system should support in order to achieve business goals while offering no details as to what functions are needed to support a feature (e.g., "*Assist in automating the article publishing process*"). *Functional service goals* provide the details of how a feature is achieved and thus contains the description of what function a user can perform (e.g., "*Editors shall be able to add new reviewers*"). *Non-functional product goals* are concerned with quality attributes of the product as a whole (e.g., *"System shall be easy to use"*). *Non-functional service goals* specify quality constraints of associated service (e.g., *"Editors shall be able to add new reviewers easily*). *Constraints* (i.e., data constraint) and various relationships between the artifacts (i.e., require, refine…) are also defined.

#### 3.2.2 *The Specification Layer*

This layer imposes rules on how each artifact should be specified. It provides guidelines for writing artifacts as to which semantic functions should and should not be used for a certain artifact. For example, since business goals are usually high-level strategic statements, *condition* or *duration* should not be specified while other parameters (i.e., *beneficiary*) are permitted. Figure 4 shows the specification rule for business goal's specifications.

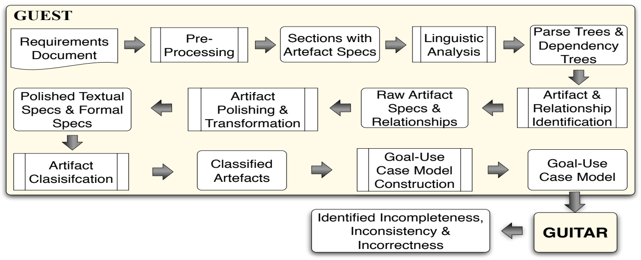

GUEST
Requirements Document
Pre-Processing
Sections with Artefact Specs
Linguistic Analysis
Parse Trees & Dependency Trees
Artifact & Relationship Identification
Raw Artifact Specs & Relationships
Artifact Polishing & Transformation
Polished Textual Specs & Formal Specs
Artifact Clasisifcation
Classified Artefacts
Goal-Use Case Model Construction
Goal-Use Case Model
GUITAR
Identified Incompleteness, Inconsistency & Incorrectness

**Figure 5: Process for Goal-Use Case Model Extraction**

```
<goal>
    <indicator>product scope, product functions, operating environment</indicator>
</goal>
<nonfunctional_goal>
    <indicator>user interfaces, security requirements, software qualities</indicator>
</nonfunctional_goal>
<constraint>
    <indicator>business rules, other requirements, environmental requirements</indicator>
</constraint>
<use_case>
    <indicator>stimulus/response sequences, scenario, use case</indicator>
</use_case>
```

**Figure 6: Example of Section Indicator List**

```
(ROOT
  (S
    (NP (DT The) (NN system))
    (VP (VBZ is)
      (VP (VBN designed)
        (S
          (VP (TO to)
            (VP (VB maximize)
              (NP
                (NP (DT the) (NN editor) (POS 's))
                (NN work) (NN productivity))
              (PP (IN by)
                (S
                  (VP (VBG automating)
                    (NP (DT the) (NN article) (NN review) (NN process)))))))))))
```

**Figure 7: Example of Parse Tree**

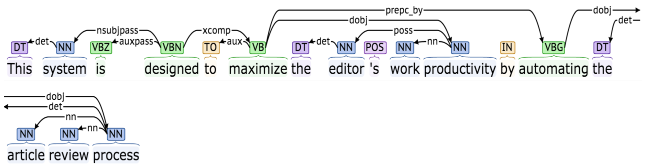

prepc_by
dobj
det
dobj
poss
nsubjpass
xcomp
det
auxpass
aux
det
nn
DT NN VBZ VBN TO VB DT NN POS NN NN IN VBG DT
This system is designed to maximize the editor 's work productivity by automating the
dobj
det
nn
nn
NN NN NN
article review process

**Figure 8: Example of Dependencies**

## 4. OUR APPROACH

Figure 5 presents an overview of our extraction approach (supported by tool GUEST) in the context of our Goal-Use Case Integration Framework (GUI-F). The extraction consists of six main steps. First, the requirements document is preprocessed to find the sections that contain goal and use case specifications. In addition, the text in these sections is analyzed to remove unnecessary information such as pictures, brackets, and multiple whitespaces. Secondly, a linguistic analysis is done on the text to resolve coreference, and obtain part-of-speech (POS) and typed dependencies of tokens (words) using the Stanford parser [15]. In step 3, our rule-based engine analyzes the outcome of the parser to identify raw artifact specifications and relationships. In step 4, these raw specifications are polished to properly specify artifacts and then parameterized. Next, the polished specifications are used to classify artifacts into different abstraction levels. In the last step, a goal-use case model is constructed. Within our GUI-F framework, the extracted goal-use case model can then be analyzed by our GUITAR tool [19], which supports the identification and resolution of inconsistency, incompleteness and incorrectness. In this paper, we focus on GUEST.

### 4.1 Requirements Document Pre-Processing

GUEST currently accepts requirements documents in .doc or .txt formats. Although there is no specific constraint on the structure of such documents, the sections in a document must be numbered in a strictly ascending order. A list of section indicators also needs to be manually created by users at the beginning. Such list contains the specifications as to which sections should be ignored, which sections should be considered as sources of goals or use cases (we call such sections *"important sections"*). Figure 6 presents an example of a simplified section indicator list in XML format. Given the indicators provided, GUEST first automatically extracts plain text from the document (i.e., remove all figures). It then analyzes the text to identify the *important sections*. It then removes unneeded details from those sections to prepare for the linguistic analysis in step 2. These include texts describing within brackets, multiple-whitespaces and symbols (i.e., ellipsis, exclamation, slashes).

### 4.2 Linguistic Analysis

The linguistic analysis includes the resolution of coreference and parsing of texts for POS and dependency information.

#### *4.2.1 Coreference Resolution*

Coreference refers to cases in which a pronoun or possessive adjective is used to replace a noun phrase in the same or nearby sentence. For instance, in the sentence *"The editor fills in the form and submits it"*, *"it"* replaces *"the form"*. In our work, we use the Stanford Coreference Resolution System [16] to resolve coreference. For instance, the resolved sentence would be *"The editor fills in the form and submits the form"*.

#### *4.2.2 Syntactic and Dependency Parsing*

To automate the identification of potential artifact specifications and relationships from a sentence, it is important for computers to *"understand"* the composition of such a sentence. Specifically, we need to identify its grammatical structure (i.e., what are the noun phrases, verb phrases, or adjectival phrases), the roles of words in the sentence (i.e., which word is verb, noun, adjective or adverb) and the relationships between words (i.e., a word is an object, or adjectival modifier of another). Based on such understanding, computers can be trained to recognize important parts while ignoring unimportant parts in the sentences in regard to goal descriptions, and identify relationships between goals.

Consider, for example, the sentence *"The system is designed to maximize the editor's work productivity by automating the article review process"*. Figure 7 shows the parse tree that contains the identification of phrases (i.e., NP – noun phrase, VP – verb phrase) and part-of-speech of words (i.e., IN – preposition, VBN – past-participle verb) in the sentence. Figure 8 presents the typed dependencies [8] between words. For example, *productivity* is the *direct object* (*dobj*) of *maximize*, *automating* is the *prepositional clausal modifier* (*prepc_by*) of *maximize*. The parsing results of these structure and dependencies are critical for our rule-based extraction technique, which will be discussed in section 4.2.3.

In our work, the linguistic parsing is done by using our extended version of Stanford Lexicalized Parser [15]. We have retrained the Stanford parser with requirements specifications data and enabled the parser to be incrementally trained with new data, without the need to re-train from scratch to accommodate new data.

### 4.3 Artifact and Relationship Extraction

The use of rules to extract artifacts and relationships is inspired by our observation that although requirements specification text is freely styled and unstructured, the identification of unimportant phrases or goal relationships usually follow certain patterns. For instance, consider again the example sentence, the phrase *"The system is designed to"* should be ignored because it contains no important information. The role of this phrase is to introduce an intention following it in the sentence (i.e., *maximize the editor's work productivity)*. If this phrase were used in another sentence, its role would not change and should still be ignored. In addition, the words in this phrase do not equally contribute to its unimportance. In fact, "system", "is", "designed" and "to" are more important than "the". This leads to the conclusion that the phrase *"system <tobe> designed to" (do something)* (with <tobe> refers to the use of *"is"*, *"are"*, *"will be"*, *"shall be"*…) should be ignored in any sentence containing it. Moreover, in this sentence the refinement relationship between "automate the article review process" and "maximize the editor's work productivity" is recognized by the structure *"do something **by** doing something"* detected in the sentence. Our observation showed that refinement relationships could be extracted by this structure in most cases.

Note that normal textual comparison cannot guarantee correct extractions. For instance, if we identify refinement relationships by looking for the exact match of *"Verb+Object+by+Verb_ing"*, then we would fail to reveal the relationship in *"maximize the editor's work productivity by **efficiently** automating the article review process"* because *"efficiently"* is now between *by* and *Verb_ing*, making the structure unmatched. It is the dependencies between the words that matter, rather than the order they appear in the sentence. In fact, the most important factor in this example is the *prepc_by* relationship between *maximize* and *automating*. This relationship would still remained unchanged regardless of what details are added into the related verb phrases of *maximize* and *automating* (the relationship between the two words would only change if the connector *"by"* is removed, or either of them is changed, or the sentence structure is modified). Therefore, we rely on the dependencies between words to define extraction rules.

**Table 1: Terminologies of Typed Dependency**

| **Term** | **Explanation** |
|---|---|
| **Node** | A word in a dependency tree |
| **Link** | A dependency between two nodes (i.e., *det(system, this)*) |
| **Universal Root** | The node that has no incoming link (i.e., designed) |
| **X sub-tree** | Sub-tree of the dependency tree that has X as its root |
| **Artifact** | Goal or use case components (i.e., step, condition) |

### 4.3.1 *Extraction Rules*

The discussion in this section is based on the examples in Figure 7 and 8. Table 1 presents some typed dependency's terminologies. Table 2 presents the syntax of our extraction rule with a list of representative rule execution actions[1]. A rule contains two parts: *condition* (specified by a list of variables and dependencies) and *actions*. In the extraction, a sentence's dependency tree is matched against the condition of a rule. If they are matched, the actions will be executed to generate a new dependency tree as the output. Since goals and use case specifications are normally located in separated sections in a requirements document and the extraction is done section by section, the extractions of them are carried out separately (except that a use case description sometimes contains information about the goal it operationalizes). We thus developed separate sets of extraction rules for goals and use cases.

#### 4.3.1.1 *Goal Extraction Rules*

There are four types of goal extraction rules as follows.

***Ignorance Rule:*** Ignorance rules are used to recognize sentences or parts of a sentence that have no important information. They thus should be ignored during the extraction process. Rule R1 (in Table 3) implies that the word *"specifically"* which is used as an *adverbial modifier* of a verb in a sentence should be ignored.

***Navigation Rule:*** a navigation rule requires the dependency tree's root to be moved to a certain node, which means every node which is not part of the new root's sub-tree will be removed. Consider rule R2 in Table 3, it can be seen that the dependency tree in Figure 8 matches this rule (X is *designed* and Y is *maximize*). Following this rule, the root (is at *designed* originally) needs to be moved to *maximize* (Y). This implies that the attention now is on the *maximize* sub-tree: "*maximize the editor's work productivity by automating the article review process"*.

[1] Full reference of our rules can be found at http://goo.gl/gCUofM

***Relationship Rule:*** relationship rules are concerned with extracting goals while identifying relationships between them. We support the specification of rules to identify sub-goal (refinement) and relevant relationships. Goals are considered relevant when they are related, but no additional information to infer more detailed relationship between them. Rule R3 can be used to identify the refinement relationship between *"automate the article review process"* and *"maximize the editor's work productivity"*.

***Splitting Rule:*** Splitting rules are used in case coordinating conjunctions (i.e., and/or) are used in a sentence. They allow a sentence to be split into two parts with sibling (if *"and"* is used) or alternative (if *"or"* is used) relationship between them. For instance, R4 can be used to extract two alternative goals *"Reader can search articles by author names"* and *"Reader can search articles by categories"* from the sentence *"Reader can search articles by author names or categories"*.

#### 4.3.1.2 *Use Case Extraction Rules*

In a section that potentially contains use cases, the use case components (i.e., use case name, steps, exceptions) can normally be identified using a list of indicators similarly to the section indicator list presented in Figure 6. For instance, the terms *"actor"* or *"primary actor"* indicate the actor specification of the use case, the terms *"main scenario"* or *"basic path"* indicate the main list of use case steps. These lists can be updated or extended depending on the needs of specific projects. However, in many cases, such indicators are missing from the use case specification, or the specification of a component contains extra information, or the components are mixed up with each other. Thus, we developed a list of extraction rules to reveal these components from texts. Below, we discuss example use case extraction rules.

***Step Extraction Rule:*** This type of rules is designed to extract use case steps combined in one single sentence (i.e., by "and/or", or "after/before"). Using a step extraction rule, not only the steps are extracted, but also their relationships (i.e., "precede") are identified. In case two steps have an "alternative" relationship, a new extension is then created to establish an alternative scenario. Rule R5 is an example of this type.

***Extension Extraction Rule:*** is used to identify extensions embedded in step description. Rule R6 can help reveal the extension embedded in step 4 in Figure 1(a). Specifically, an extension with condition *"The editor is not updating a reviewer"* and a step *"the system presents a list of reviewers and presents information of a reviewer after the editor selects that reviewer"* is extracted. This step is further extracted into three consecutive steps using our step extraction rules.

***Use Case Constraint Extraction Rule:*** is used to identify non-functional or data constraints that are combined together with use case steps. For instance, Rule R7 can be used to recognize the data constraint "*The manage options include 'Add reviewer' and 'Update reviewer'*" in the motivating example (cf. Figure 1).

***Use Case Relationship Rule:*** This is to identify use case relationships (i.e., *extend*, *include*). Consider a use case step *"Use case 'Register for membership' is performed"*, Rule R8 can be used to identify the *"include"* relationship between the currently processed use case and the "Register for membership" use case.

**Table 2: Extraction Rule Syntax**

| | **Syntax** | **Explanation & Example(s)** |
|---|---|---|
| **Generic Rule syntax** | `<Variable Declarations>`<br>`<Dependency Declarations>`<br>`-> <Action Declarations>` | The syntax of a rule contains 2 parts: variables and dependencies specify the matching *condition* of the rule, and a list of *actions* to be executed if the rule is matched. |
| **Variable declaration** | `X={a} or X={a\|b}` | Variable X has the value of a, or one of the values in a, b… **Example**: `X={designed\|aimed}` |
| | `X/AB or X/{AB\|CD}` | Variable X has the POS tag of AB, or one of the POS tags in AB, CD… **Example**: `X/{NN\|NNS}` |
| **Dependency Declaration** | `root(X)` | Specify that X is the root of the dependency tree (a root has no dependency link pointing to it) |
| | `dep_name(X, ?)` | There is a *dep_name* dependency between X and any node. **Example**: `dobj(X, ?)` |
| | `dep_name(X, Y)` | There is a *dep_name* dependency between X and Y. **Example**: `xcomp(X, Y)` |
| | `dep_name(X, ?/{AB\|CD})` | There is a *dep_name* dependency between X and any node having one of the POS tags of AB, CD, … **Example**: `nsubj(X, ?/{NN\|NNS})` |
| | `dep_name(X, {a\|b})` | There is a *dep_name* dependency between X and any node that has one of the values of AB, CD, … **Example**: `nsubjpass(X, {system\|project})` |
| | `not dep_name(X, Y)` | There is no *dep_name* dependency between X and Y. **Example**: `not xcomp(X, Y)` |
| **Action Declaration** | `ignore(X)` | Ignore the X sub-tree. If X is the root of the entire dependency tree, then the whole sentence is ignored |
| | `root(X)` | Move the root to node X, consider only the sub-tree whose root is X, ignore the rest of the tree |
| | `sub_goal(goal(X), goal(Y))` | Establish a sub-goal relationship between goal extracted from the X sub-tree and the one from Y sub-tree |
| | `split_sibling(X, Y)`<br>`split_alternative(X, Y)` | Splitting the sentence into two separated ones (based on the conjunction), build goals based on these sentences and establish a sibling (if and is used) or alternative (if or is used) between these goals |
| | `preceed(statement(X),`<br>`statement(Y))` | Specify that the statement extracted from the X sub-tree is the preceding step of the one from Y sub-tree |
| | `uc_data_constraint`<br>`(statement(Y))` | Specify the statement extracted from the Y sub-tree is a data constraint of the currently processed step |

**Table 3: Examples of Goal Extraction Rules**

| **Ignorance Rule (R1)** | **Navigation Rule (R2)** | **Relationship Rule (R3)** | **Splitting Rule (R4)** |
|---|---|---|---|
| `X={specifically}`<br>`advmod(?/{VB\|VBZ\|VBN}, X)`<br>`-> ignore(X)` | `X={designed\|aimed}    Y/VB`<br>`root(X)`<br>`nsubjpass(X, {system\|project})`<br>`auxpass(X, {be\|is\|are})`<br>`xcomp(X, Y)`<br>`-> root(Y)` | `X/{VB\|VBD\|VBG\|VBN\|VBP\|VBZ}`<br>`Y/VBG`<br>`root(X)`<br>`prepc_by(X, Y)`<br>`->sub_goal(goal(Y), goal(X))` | `X/{NN\|NNS\|NNP\|JJ}`<br>`Y/{NN\|NNS\|NNP\|JJ}`<br>`inferred_conj_or(X, Y)`<br>`-> split_alternative(X, Y)` |
| **Step Extraction Rule (R5)** | **Extension Extraction Rule (R6)** | **Constraint Extraction Rule (R7)** | **Use Case Relationship Rule (R8)** |
| `X/{VB\|VBD\|VBG\|VBN\|VBP\|VBZ}`<br>`Y/{VB\|VBD\|VBG\|VBN\|VBP\|VBZ}`<br>`inferred_conj_and(X, Y)`<br>`not prep_between(?/{NN \|NNS} , X)`<br>`not prep_between(?/{NN \|NNS}, Y)`<br>`not mark(X, {if})`<br>`not mark(Y, {if})`<br>`-> preceed(statement(X), statement(Y))` | `X/{VB\|VBD\|VBG\|VBN\|VBP\|VBZ}`<br>`Y/{VB\|VBD\|VBG\|VBN\|VBP\|VBZ}`<br>`Z/{VB\|VBD\|VBG\|VBN\|VBP\|VBZ}`<br>`mark(X, {if})`<br>`nsubj(X, ?/{NN\|NNS\|NNP\|NNPS})`<br>`advcl(Y, X)`<br>`root(Y)`<br>`advmod(Z, [10])`<br>`parataxis(Y, Z)`<br>`-> extension_condition(neg(X)), extension_step(Z), uc_step(Y)` | `X={include\|contain}`<br>`Y={choices\|options\|alternatives}`<br>`Z/{NN\|NNS\|NNP}`<br>`W/{NN\|NNS\|NNP}`<br>`root(X)`<br>`nsubj(X, Y)`<br>`dobj(X, Z)`<br>`dobj(X, W)`<br>`conj_and(Z, W)`<br>`-> uc_data_constraint(statement(Y))` | `X={case}`<br>`Y={use}`<br>`Z={performed}`<br>`W/{VB\|VBD\|VBG\|VBN\|VBP\|VBZ\|NN \|NNP\|NNS}`<br>`nn(X, Y)`<br>`nsubjpass(Z, X)`<br>`ccomp(Z, W)`<br>`auxpass(Z, {is\|be})`<br>`root(Z)`<br>`-> uc_include(W)` |

### *4.3.2 How Are Extraction Rules Used?*

Rules can conflict with each other. For instance, two rules specifying the same condition (list of dependencies), or the conditions of a rule is a sub-set of the conditions of another rule but their actions are different. GUEST is able to detect such rules and report them to end users for modification (in some cases, condition overlap is not a problem if there is a rule of higher priority than another).

In addition, there exist cases that a single sentence matches multiple rules and executing a rule before another may lead to different results. To solve this problem, rules of different types are given different priorities. For instance, the goal extraction rules are prioritized in the following order: ignorance rules, navigation rules, relationship rules and splitting rules. In the case of having multiple matching rules of the same type, GUEST executes all of them and produces alternative outputs. The tool then reports this issue to users for their decisions.

To extract artifacts from sentences, we use an iterative process to analyze each sentence in consideration of the rule ordering. For instance, the outcome (may contain multiple goals if a relationship or splitting rule is used) is then considered in the next iteration and so on. The process ends when no matching rule is found.

## 4.4 Polishing and Parameterization

In this step, extracted text artifacts in the form of dependency trees are taken to produce polished textual specifications and Functional Grammar-based parameterization.

### *4.4.1 Artifact Specification Polishing*

Specification polishing is used to ensure textual artifacts are expressed in a proper way. The cases when polishing is required include: (1) the text extracted from a sentence and its tense (i.e., continuous tense) is not suitable for a stand-alone artifact specification, (2) the text is in passive voice and (3) the text is incompatible with our meta-model's specification rules.

The first two cases require algorithms to check the dependency tree for tense (i.e., the root is a verb with VBG POS tag) or passive voice use (i.e., look for *auxpass* dependency link) and making relevant modifications in the dependency tree. However the third case requires deeper analysis. In our work, it is recommended that artifacts are specified using action verb whenever possible to enable better comparison and analysis of artifacts [19]. This view has also been adopted in many requirements engineering research (i.e., [26]). However there exist many cases in which functionalities or conditions are described in other forms, i.e., using adjective-preposition phrases. The examples below present some of these cases.

***Example 13:*** *"editors are capable of entering new reviewers"* is re-written as *"editors shall be able to enter new reviewers"*.

***Example 14:*** *"Readers without technical knowledge can search for articles"* should be re-written as *"Readers who do not have technical knowledge shall be able to search for articles"*.

***Example 15:*** *"There is more than one reviewers in the list"* should be rewritten as *"More than one reviewers exist in the list"*.

To solve this problem, we developed an extendable set of *rewriting rules* that share the same syntax with extraction rules. A text is checked for a match with a rewriting rule that then triggers the execution of the rule actions to make necessary modifications. The phrase *"shall be able to"* is added to goal specifications only. The rewriting rule for the text in example 13 can be found below.

```
X={capable}
Y/{NN|NNS|NNP|NNPS}
Z/{NN|NNS|NNP|NNPS}
T={be|is|are|were|was}
nsubj(X, Y)
cop(X, T)
root(X)
prep_of(X, Z)
-> replace_adj_root(X), add_verbal_root(W, {do}, X, Z, T,
true, false, Z), dobj(W, Z);
```

### *4.4.2 Artifact Specification Parameterization*

Parameterization of specifications enables the understanding of artifacts because it identifies the semantic role of each single word in an artifact specification. For instance, given the goal specification of *"Editors shall be able to add new reviewers easily"*, parameterized as *Agent(Editor) + Verb(Add) + Object( Head(Reviewer) + Attribute(New) + Manner(Easily)*. From the parameterization it can be identified which function the system supports (*add new reviewers*), who the function is for (*editors*), how the function is accomplished (*easily*), what the function's object is (*reviewers*), what the object's attribute is (*new*). We term such representation as *structured specification* in our framework.

To generate structured specification from a textual one, we need to identify the semantic role (function) each word or group of words plays. The input for this process is the polished dependency tree from step 4.4.1. This process starts with the investigation in the dependency tree to determine the value of the *Verb* semantic function since it is the central function in our structured specifications. Normally *universal root* constitutes the value of *Verb* except when *to-be* verb is used. The determination as to which semantic function a group of words should fall into depends on the relationship between the root of that group and the *universal root*. For instance, if X is the *universal root*, then the relationship *nsubj*(X, Y) indicates that the Y sub-tree is the value of the *Agent* semantic function. Similarly, dobj(X, Y) may indicate Y sub-tree is the value of the *Object* semantic function.

However, due to the complexity of English, there are always exceptions. For instance, in the sentence *"system notifies users about the changes"*, although *notifies* is the root and there exist the relationship *dobj(notifies, users)*, *users* is not the *Object*, but instead *Beneficiary*. In addition, there is no rule for prepositional phrases. For instance, in dependency tree of the sentence *"editors can login with their accounts"* has the *prep_with(login, account)* relationship and *"their accounts"* has the *Means* semantic role in this case. However, in sentence *"editors can communicate with reviewers"*, *"reviewers"* has the *Company* semantic role although the relationship *prep_with(communicate, reviewers)* exists. We overcome this problem by developing a set of 168 *semantic labeling rules* based on an investigation on the common English verb, adjective + preposition combinations. Below we give labeling rules for the discussed examples. The first one means that if *prep_with(communicate, Y)* exists, the Y sub-tree has the *Means* semantic role. The second one means that if *prep_of(notify, Y)* and *dobj(notify, Z)* exist, then Y sub-tree is *Reference* and Z sub-tree is *Beneficiary* semantic functions.

```
communicate, with->COMPANY
notify%, of-> A1:BENEFICIARY, REFERENCE
```

## 4.5 Goal Classification

After the extracted dependency trees are polished, textual specifications are generated from them. In this step, they are classified to determine whether they are functional or non-functional goals, and which levels of abstraction they are on.

Although a number of techniques and tools have been proposed to automatically classify requirements in the literature (i.e., [5, 7]), none of is currently available for download and use in our work. We thus selected Mallet [18], one of the best general text classifiers that are available, for artifact classification. Mallet can classify a text into a fixed set of classes, such as "functional" vs. "non-functional", based on labeled training examples. It includes implementations of several classification algorithms, including Naïve Bayes, Maximum Entropy, and Decision Trees. Mallet calculates the probability of each word for being classified into a certain class based on the labeled training data. The probability of the whole text for being classified into a certain class is determined by the probabilities of the words it contains. In this work, we extended Mallet by using text-preprocessing to improve its accuracy (to be discussed in section 5). The improvements are discussed as follows:

- ***Removal of unimportant content:*** Mallet considers every word in a text for probability calculation. However, not all words are needed in this process. In fact, we modified it to remove unimportant details such as stop words (i.e., *a*, *the*, *that*, *shall*), symbols (i.e., coma) and numbers since their existence does not determine the class of a text.
- ***Ensure the standard form of the word is used:*** We updated Mallet to consider only the standard form of words. For instance, the probability is calculated for "*present*", not "*presents*" or "*presented*" if they are used in the text.
- ***Use a set of classification keywords:*** We developed a set of 310 keywords that support the classification of artifacts. For instance, *"available"* and *"easy-to-use"* are non-functional goal keywords. *"Productivity"* is a business goal keyword.

Our classifier is composed of two separated classifiers. The horizontal classifier is used to determine if a text is business, functional or non-functional goal specifications. For a non-functional goal, it also provides the prediction as to which non-functional category it belongs to (i.e., *security*, *usability*). Currently we support the identification of 11 non-functional categories. The vertical classifier is used to identify the abstraction level a goal should belong to (product, feature, or service level). The rationale for using two separated classifiers is that they can help reducing training data size. For instance, if a single classifier were used, then we would need sufficient training data for 13x3 classes, as opposed to only 13 classes (business goal, functional goals and 11 non-functional goal categories) for the *horizontal classifier* and 3 classes for the *vertical classifier*. Additionally, the training data can be shared between the two classifiers, given they are appropriately labeled for each classifier. Up to now we have trained the classifiers with over 1200 requirements collected from multiple sources including online resources, literature and books. GUEST allows the classifiers to be further trained or re-trained.

## 4.6 Goal-Use Case Model Construction

After the artifacts are extracted and classified, and relationships are identified, the model can be constructed. The following subsections describe the main issues to be addressed in this step.

***Identify duplicate artifacts:*** sometimes goals are repeated in different sentences in requirements documents. For instance, the goal *"maximize the editor's productivity"* is repeated twice in the motivating example (cf. Figure 1(a)). Repeated goals are merged into one in the model. In GUEST, not only exactly duplicate artifacts are identified, highly overlapping artifacts are also reported to end users. The identification of overlaps between artifacts is done using their parameterizations. In fact, we compare two specifications by matching their corresponding semantic functions (i.e., match *Agent* of a specification with *Agent* of another, *Object* with *Object*). Partly overlapped artifacts are reported to users to decide if a merge of them is suitable.

**Report model construction problems:** other problems may occur during the model creating. For instance, a goal has the equivalent probabilities for two classes (i.e., *feature* and *service*). Another example is an inconsistency can exist if the goals FSG1, FFG1 are classified as functional service and functional feature goal respectively, while an extraction rule infers that FFG1 is a sub-goal of FSG1. This situation is invalid since feature goals are on a higher level than service goals and thus a feature goal cannot refine a service goal. All problems from the extraction process are reported to users for the decisions.

***Other problems:*** The emphasis of this paper is on extracting goal-use case models from what provided in requirement documents. The problems related to the extraction are identified and reported to users. However, problems inherited from the requirements documents, e.g., the model is inconsistent due to some inconsistencies exists in the original requirement documents, are beyond the scope of this paper. Such problems have been tackled in our previous work with the GUITAR tool [19, 20] which supports the analysis of goal-use case models for incompleteness, inconsistency and incorrectness. Since GUEST is integrated into GUITAR, we can ensure the seamless support for the building and analysis processes of goal-use case models.

GUEST is not aimed at fully automating the entire extraction process, as this is almost impossible since requirements documents can be found in different structures and written in uncontrolled styles. It is rather intended to assist requirements engineers in modeling goals and use cases from requirements documents. Thus, there may be cases not all artifacts and relationships can be extracted (i.e., a necessary extraction rule may be missing). To assist requirements engineers in verifying the extraction results and possibly continue the extraction manually, GUEST provides features such as: producing extraction logs that capture all steps during the extraction process, mapping artifacts with original text where they are extracted from and allowing users to modifying the extracted models.

## 5. EVALUATION

In this section, we present three evaluations conducted to evaluate the effectiveness of GUEST in extracting goal-use case models from requirements documents. Specifically, we seek to answer the following research questions:

- **RQ1:** How accurately does GUEST classify artifacts by their textual specifications?
- **RQ2:** How accurately does GUEST parameterize textual specifications?
- **RQ3:** How accurately does GUEST extract goal-use case models from requirement specifications documents?

Table 4 presents our formulas to calculate the metrics for each research questions. Our full experimental data can be found at http://goo.gl/gCUofM.

**Table 4: Formulas for Metrics**

| | **Formulas** |
|---|---|
| **RQ1** | $Precision = \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN}$ **TP:** True Positive (number of artifacts are correctly classified) **FP:** False Positive (number of incorrectly classified artifacts) |
| **RQ2** | $Accuracy = \frac{Correct\ Parameterizations}{Total\ specifications\ parameterized}$ |
| **RQ3** | $Precision = \frac{TP}{TP+FP}, Recall = \frac{TP}{TP+FN}, Fmeasure = \frac{2*P*R}{P+R}$ **TP:** True Positive (number of valid extracted artifacts or relationships) **FP:** False Positive (number of invalid extracted artifacts or relationships) **FN:** False Negative (number of artifacts or relationships not extracted) |

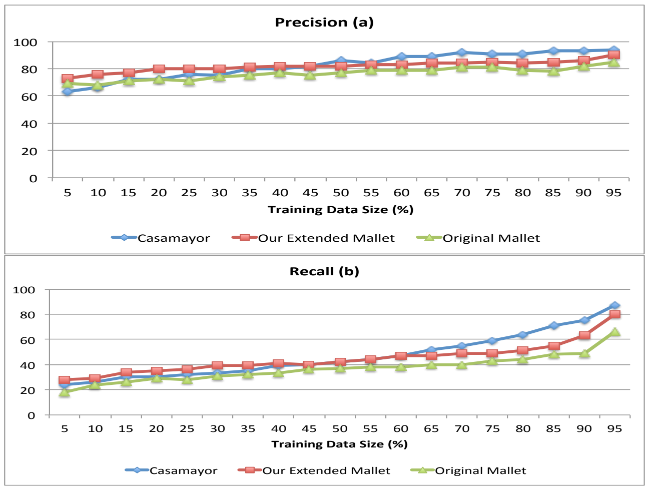


**Figure 9: Classifier Benchmark Validation Results**

### 5.1 RQ1: Artifact Classification

A benchmark validation was carried out to compare and contrast our requirements classifier with the existing state-of-the-art classifiers: Casamayor et.al. non-functional requirements classifier [5] (we term it as Casamayor) and original version of Mallet (original Mallet). The reasons for this selection were fourfold. First, Casamayor is among the classifiers developed recently and reportedly obtained high results in its evaluation. Second, Casamayor's experiment steps were described clearly and its data is available, making it possible to reconstruct the exactly same validation. Thirdly, no requirements classifier was available to download and lastly, original Mallet was selected to verify whether our improvements made to extend it were effective.

We followed Casamayor's experiment steps to run a validation on PROMISE dataset [2] that consists 625 requirements collected from 15 software development projects. Among them, 255 items are marked as functional requirements and the remaining 370 non-functional requirements are classified into 11 categories, such as Security, Performance and Usability. Only the non-functional requirements were used in this evaluation. Multiple experiments were run. In each experiment, a k-portion of data (i.e., k=90%) was randomly selected as training data and the rest used as testing data. Each experiment is run in 10 iterations to obtain the scores for the precision and recall metrics (cf. Table 2). 19 experiments were run with k was 5, 10,…, 95.

Figure 9 (a) and (b) present the comparison graph between three classifiers for precision and recall. All three classifiers obtained higher precision and recall rates when the training set size increased. It can be noticed from this comparison that our classifier produced higher-quality results than others when the training set was small. This was due to the support of our non-functional indicator keywords. When the training set size increased, Casamayor's results raised with highest rates and surpassed our classifier when the training set was over 55% of the entire data (as showed in the graphs). Original Mallet followed a very similar trend as our classifier that is due to the share of algorithms between the two classifiers. However, our classifier outperformed original Mallet with at least 5% difference in most experiments. From these results the key benefit of our classifier is that it performs relatively well with a small training dataset.

**Table 5: Parameterization Validation Results**

| | |
|---|---|
| **Round 1 (existing capability)** | 274 over 310 (88%) |
| **Round 2 (best achievable capability)** | 297 over 310 (96%) |

**Table 6: Extraction Validation Results**

| Case Study | | OPS | SPS | Average |
|---|---|---|---|---|
| **Artifact** | **D** | 175 | 172 | |
| | **FP** | 26 | 23 | |
| | **FN** | 17 | 29 | |
| | **Precision** | 85% | 86% | 86% |
| | **Recall** | 89% | 83% | 86% |
| **Relationship** | **D** | 144 | 102 | |
| | **FP** | 22 | 16 | |
| | **FN** | 36 | 26 | |
| | **Precision** | 84% | 84% | 84% |
| | **Recall** | 77% | 76% | 77% |
| **Polishing & Parameterization** | | 91% | 89% | 90% |
| **Goal Classification** | | 81% | 78% | 80% |
| D: total detected FP: False Positive FN: False Negative | | | | |

## 5.2 RQ2: Artifact Parameterization

PROMISE data was pre-processed before being used in this validation. For instance, we split a requirement into multiple ones if it contains more than one sentence. In addition, combined words such as "his/her", "himself/herself" were changed to single words (i.e., his). Moreover, each sentence with coordinating conjunctions (i.e., "and") is split. However, if the split sentences have identical structure (i.e., "*readers can search articles*" and "*readers can download articles*"), we only keep one of them to maintain the structural differences between requirements. We have randomly selected 200 requirements from all 15 projects and 110 requirements collected from the literature. Each requirement was parameterized by GUEST and the results were manually checked by us to determine the accuracy rates. A two round-validation was conducted. Firstly, we parameterized the requirements based on our existing collection of rewriting and labeling rules to verify GUEST's *existing parameterization capability*. We then identify the reason for errors found in the results. If the reason was incorrect parsing or missing supporting rules, we then trained the parser with a correct parse tree or attempted to write new rules using our defined syntax. In round 2, the failed parameterizations were re-generated with new information. The result of this round was the *best achievable capability* of GUEST in this validation.

Table 5 indicates that we obtained 88% and 96% of accuracy in round 1 and 2 respectively. There were a number of requirements unsupported by GUEST because their grammatical structures were not recognized by our meta-model. For instance, "Out of 1000 accesses to the system, the system is available 999 times".

## 5.3 RQ3: Goal-Use Case Model Extraction

We used the online publication system (OPS) and split payment system (SPS) industrial case studies in this validation. OPS case study comes with a requirements document with 31 pages and 503 sentences. SPS requirements document contains 46 pages and 586 sentences. Each requirements document follows the IEEE requirements specification template and contains a number of sections for goals and use cases. In each case study, we manually modeled goals and use cases from the given requirements document and then compare that model with the one generated by GUEST. We analyzed the results of each extraction phrase to provide detailed evaluation of the approach. The phrases analyzed were: extraction of raw artifacts and relationships, polishing and parameterization of artifacts, and goal classification.

The results are showed in table 6. We achieved 86% precision and recall rates for the artifact extraction, and 84% precision and 77% recall rate for the relationship extraction. A number of artifacts and relationships were not extracted due to the missing of relevant extraction rules. In addition, some relationships were not detected since detecting such relationships required the understanding of the entire contexts in which the artifacts were specified. For instance, *"this system is designed to allow an editor to communicate with reviewers and authors. The software will facilitate communication between authors, reviewers, and the editor via E-Mail"*. GUEST could not identify the relationship between these two goals. However, an alert regarding the possible overlap between them was generated. We achieved 90% accuracy rates of the polishing and parameterization of artifacts (to be considered correct, the artifact needed to be both correctly polished and parameterized). The common errors were due to missing relevant rewriting rules. For instance, GUEST could not properly rewrite the goal "Include support for simultaneous bills" to the desired form "Support simultaneous bills". We achieved 80% for goal classification in this validation.

These evaluation results indicate that the missing artifacts and relationships were due to missing extraction rules. The incorrectness in artifact parameterization and classification came from invalid results produced by the linguistic parser and artifact classifier respectively. Results could be improved if our extraction rules set is extended and the parser and classifier further trained.

## 5.4 Threats to Validity

Threats to external validity include representativeness of the selected subjects and quality of our parser, classifier and rules. To reduce these we increased the variability of the data by selecting requirements from different sources. For RQ2 we carried our a second round of validation to evaluate the tool in case the parser and rules were perfect for the given set of requirements. Threats to internal validity include the human factors in determining the correctness of GUEST results in each validation. In RQ3, we manually extracted goals and use cases from the requirements documents. In RQ2 and RQ3, we manually verified the tool's outputs for semantic parameterization, goals, use cases and their relationships and classifications. To mitigate, we reviewed all manual tasks twice. Using two or more people with relevant knowledge and experience in validation would further improve.

## 6. DISCUSSION AND FUTURE WORK

**Reduce effort for goal-use cases modeling:** Our approach can reduce the effort and time to model goals and use cases from requirements documents for analysts. Moreover, GUEST can potentially be used to quickly gain understanding of natural language requirements documents. GUEST rules can be used across different projects. Users can add new rules to improve the quality of extraction. Furthermore, since GUEST's underlying techniques for natural language parsing and artifact classification provide support for multiple languages, it is possible to adopt and apply our approach for requirements written in other languages. However, a new set of extraction rules that suits the grammars of each such language needs to be developed.

**Enable the Analysis of goal-use case models:** Our automated semantic parameterization enables the seamless integration of GUEST and our tool GUITAR which provides automated analysis of goals and use cases [19]. This provides comprehensive support for the modeling and analysis on goal-use case models.

**Possible application of our technique in other areas:** Our technique of automated parameterization can be used for any textual sentences. While our set of extraction rules was developed specifically for goals and use cases, its underlining concept can still be applied to support the information extraction in other areas. For instance, extraction rules can be developed in a similar way (i.e., develop rule actions and algorithms to execute these actions) to identify privacy or security policies [31] from software documents. Our rule-based technique can also provide a new approach in ontology learning [4, 6]. Specifically, similar rules can be created to detect ontological concepts, properties and their relationships to extract ontologies from natural language texts.

**GUEST's extraction accuracy depends on the quality of the Stanford parser and the artifact classifier:** a common issue for a statistical machine learning technique is that it may not produce correct results for what it has not been trained for. Therefore, it is possible to have a sentence incorrectly parsed or a specification incorrectly classified in GUEST. Such problems can be resolved by training the parser and classifier with relevant data. GUEST provides the incremental training of both the parser and classifier.

**Understanding of grammatical dependency required for rules writing:** In GUEST, the extraction and rewriting rules need to be manually written. This requires the rule writers to have knowledge of grammatical dependency and thus some training would be required for end users to be able to extend the rules repository. We plan to overcome this problem by developing an algorithm that semi-automates the generation of a rule from the associations between sentences and lists of desired information to be extracted from them. In addition, a visual rule editor would be developed.

**Unidentifiable artifact relationships:** a number of relationships between artifacts in different sentences are not detectable. Detecting such relationships requires the understanding of the entire context in which they are specified. Our future work will thus will focus on resolving these types of problems.

**Lack of evaluation of the approach's usefulness:** although having promising results in our case study-based evaluation, GUEST has not been validated for a real software project. We thus plan to carry out an evaluation with our industry partners to evaluate the approach's usefulness in requirements engineering.

## 7. RELATED WORK

To the best of our knowledge, no technique has been proposed to automatically extract goal-use case models from natural language documents. In this section, we discuss existing techniques that extract requirements or use cases separately.

**Natural language requirements Extraction and formalization** Rauf et.al. [25] proposed a technique to identify sections that contains logical structures (LSs) such as requirements or use cases and logical components (LCs) such as actor or use case extensions in a requirements document. Their objective is to locate where requirements and use cases are located, similar to what we achieved by using a list of section indicators. This work, however, does not extract individual requirements, use cases and their relationships. Niu and Easterbrook [23] semi-automatically extract product line requirement asset from natural language requirements documents based on the use of Functional Requirement Profile, domain terminologies, and heuristic rules. However, this only focuses on functional requirements in the format of Verb+Direct Object. Ghosh et.al. [12] transforms textual requirements into Linear Temporal Logic based on the use of dependency parsing, domain specific terminologies and a set of transformation rules. It has no support for extracting requirements or goals from texts. Breaux et.al. [3] developed a semantic parameterization technique to formalize natural language goal specifications with Description Logics. They provided a set of templates in which a statement contains a number of semantic components such as *subject*, *object* and *location*. This is similar to semantic functions in our work. However, we provide a larger set of semantic roles (i.e., our *frequency*, *duration* roles are not supported in their work). Also, this work does not automate the semantic parameterization.

**Extraction of use cases from natural language documents** Ilieva [13] used linguistic analysis to extract use case paths model from uncontrolled natural language use case specifications. This work is only concerned with identifying actor, action (a single verb) and their ordering without considering other information (i.e., object, target of a use case step, pre/post condition or data constraints). Drazan and Mencl [11] developed a technique to identify use case steps from textual use case descriptions. Similar to our work, it is able to identify multiple steps combined using coordinating conjunctions. The limitation of this work is that it assumes use case descriptions are written in a restricted natural language. In addition, it does not allow the extraction of other use case components (i.e., pre/post condition or extensions). Sinha et.al. [27] developed a linguistic analysis engine to get the understanding of textual use case descriptions. The engine is able to identify components in use case steps (actor, action) and classify steps into a number of predefined classes (i.e., UPDATE, INPUT, OUTPUT) using a domain dictionary. This work, however, does not deal with the cases when multiple artifacts (i.e., step and constraint) are mixed together. In addition, various use case components (i.e., conditions, extensions, constraints) are not detectable. Rago et.al. [24] focused on extracting sequenced use case steps from textual use cases and identifying duplication among them. It however does not support the identification of other use case components and relationships between use cases.

## 8. SUMMARY

We have developed a semi-automated rule-based approach to extract goal-use case models from unformatted textual requirements documents. It incorporates various techniques to locate goals, use cases and their relationships from text, ensure they have proper textual specifications, classify goals and provide semantic parameterization of their textual specifications. Evaluation results are very promising. In two selected case studies, GUEST achieved 86% precision and recall rates for goals and use cases extraction, 84% and 77% of precision and recall rates for relationships extraction. It obtained 88% accuracy for the automated parameterization with PROMISE data. The evaluation showed that our artifact classifier entirely outperformed Mallet and was better than Casamayor's classifier with smaller training datasets. GUEST is integrated with our previous work on goal-use case automated analysis (GUITAR), providing a comprehensive framework for goal-use case extraction and analysis.

## 9. ACKNOWLEDGEMENTS

The authors gratefully acknowledge support from the Victorian Government under the Victorian International Research Scholarships scheme, Swinburne University of Technology, and the Australian Research Council under Linkage Project LP130100201.

## 10. REFERENCES

[1] A.I. Anton, R.A. Carter, A. Dagnino, J.H. Dempster, and D.F. Siege. *Deriving goals from a use-case based requirements specification.* Requirements Engineering, 6(1): p. 63-73. 2001.

[2] G. Boetticher, T. Menzies, and T. Ostrand, *The PROMISE Repository of Empirical Software Engineering Data*, 2007.

[3] T.D. Breaux, A.I. Antón, and J. Doyle. *Semantic parameterization: A process for modeling domain descriptions.* ACM Transactions on Software Engineering and Methodology (TOSEM), 18(2): p. 5. 2008.

[4] P. Buitelaar, D. Olejnik, and M. Sintek, *A protégé plug-in for ontology extraction from text based on linguistic analysis*, in The Semantic Web: Research and Applications. 2004, Springer. p. 31-44.

[5] A. Casamayor, D. Godoy, and M. Campo. *Identification of non-functional requirements in textual specifications: A semi-supervised learning approach*. Information and Software Technology, 52(4): p. 436-445. 2010.

[6] P. Cimiano and J. Völker, "Text2Onto", in *Natural language processing and information systems*. 2005, Springer. p. 227-238.

[7] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. *Automated classification of non-functional requirements.* Requirements Engineering, 12(2): p. 103-120. 2007.

[8] M.-C. De Marneffe and C.D. Manning. *Stanford typed dependencies manual.* URL http://nlp.stanford. edu/software/dependencies manual. pdf. 2008.

[9] S.C. Dik, *The theory of functional grammar*. Walter de Gruyter. 1989.

[10] S.C. Dik, K. Hengeveld, E. Vester, and C. Vet. *The hierarchical structure of the clause and the typology of adverbial satellites.* Layers and levels of representation in language theory: p. 25-70. 1990.

[11] J. Drazan and V. Mencl, *Improved processing of textual use cases: Deriving behavior specifications*, in *SOFSEM 2007: Theory and Practice of Computer Science*. 2007, Springer. p. 856-868.

[12] S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, and W. Steiner. *Automatically Extracting Requirements Specifications from Natural Language.* arXiv preprint arXiv:1403.3142. 2014.

[13] M. Ilieva. *Use Case Paths Model Revealing Through Natural Language Requirements Analysis*. in IC-AI. 2007.

[14] I.J. Jureta, S. Faulkner, and P.-Y. Schobbens. *Clear justification of modeling decisions for goal-oriented requirements engineering.* Requirements Engineering, 13(2): p. 87-115. 2008.

[15] D. Klein and C.D. Manning. *Accurate unlexicalized parsing*. in Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1. Association for Computational Linguistics. 2003.

[16] H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky. *Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task*. in Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task. Association for Computational Linguistics. 2011.

[17] J. Lee, N.-L. Xue, and J.-Y. Kuo. *Structuring requirement specifications with goals.* Information and Software Technology, 43(2): p. 121-135. 2001.

[18] A.K. McCallum. *MALLET: A Machine Learning for Language Toolkit*. 2002.

[19] T.H. Nguyen, J. Grundy, and M. Almorsy. *GUITAR: An ontology-based automated requirements analysis tool*. in Requirements Engineering Conference (RE), 2014 IEEE 22nd International. IEEE. 2014.

[20] T.H. Nguyen, J.C. Grundy, and M. Almorsy. *Ontology-based automated support for goal–use case model analysis.* Software Quality Journal: p. 1-39. 2015.

[21] T.H. Nguyen, B.Q. Vo, M. Lumpe, and J. Grundy. *REInDetector: a framework for knowledge-based requirements engineering*. in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM. 2012.

[22] T.H. Nguyen, B.Q. Vo, M. Lumpe, and J. Grundy. *KBRE: a framework for knowledge-based requirements engineering.* Software Quality Journal, 22(1): p. 87-119. 2014.

[23] N. Niu and S. Easterbrook. *Extracting and modeling product line functional requirements*. in International Requirements Engineering, 2008. RE'08. 16th IEEE. IEEE. 2008.

[24] A. Rago, C. Marcos, and J.A. Diaz-Pace. *Identifying duplicate functionality in textual use cases by aligning semantic actions*. Software & Systems Modeling: p. 1-25. 2014.

[25] R. Rauf, M. Antkiewicz, and K. Czarnecki. *Logical structure extraction from software requirements documents*. in Requirements Engineering Conference (RE), 2011 19th IEEE International. IEEE. 2011.

[26] C. Rolland, C. Souveyet, and C.B. Achour. *Guiding goal modeling using scenarios.* Software Engineering, IEEE Transactions on, 24(12): p. 1055-1071. 1998.

[27] A. Sinha, A. Paradkar, P. Kumanan, and B. Boguraev. *A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases*. in Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on. IEEE. 2009.

[28] I. Sommerville. *Software engineering.* Software Reuse, based on Software Engineering. 2012.

[29] S. Supakkul and L. Chung. *Integrating FRs and NFRs: A use case and goal driven approach*. framework, 6: p. 7. 2005.

[30] A. Van Lamsweerde, R. Darimont, and E. Letier. *Managing conflicts in goal-driven requirements engineering.* Software Engineering, IEEE Transactions on, 24(11): p. 908-926. 1998.

[31] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie. *Automated extraction of security policies from natural-language software documents*. in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. ACM. 2012