CHARLES UNIVERSITY IN PRAGUE
FACULTY OF MATHEMATICS AND PHYSICS

MASTER'S THESIS

Jaroslav Dražan

Natural Language Processing of Textual Use Cases

Department of Software Engineering
Advisor: RNDr. Vladimír Mencl, Ph.D.
Study Program: Computer Science

I would like to thank my advisor Vladimír Mencl for his important support and priceless comments. I also thank Daniel Zeman and Václav Honetschläger for their help with a linguistic approach to my issues.

I declare that I wrote the thesis by myself and listed all used sources. I agree with making the thesis publicly available.

Prague, September 30, 2005 Jaroslav Dražan

Název Práce: Natural Language Processing of Textual Use Cases
Autor: Jaroslav Dražan
Katedra: Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Vladimír Mencl, Ph.D.
E-mail vedoucího: mencl@nenya.ms.mff.cuni.cz

Abstrakt: Požadavky na funkčnost navrhovanému systému jsou obvykle zachycovány pomocí use case psaných v přirozeném jazyce. Formát use case není standardizován, nicméně věty use case mají tradičně jednoduchou strukturu a popisují buď komunikaci mezi aktéry a navrhovaným systémem, nebo interní akce. Přirozený jazyk se používá, neboť je srozumitelný pro zainteresované strany a je dostatečný k zachycení většiny požadavků, bohužel ale díky jeho volnosti není možné snadno automatizovat jeho analyzování.

Vladimír Mencl použil současné lingvistické nástroje, aby získal popis chování navrhovaného systému z use case napsaných v přirozeném jazyce. Specifikaci chování zachycuje ve formě pro-case. Z jeho práce vyplývá, že je to možné, ale také se objevilo několik problémů, které daná práce neřeší. A právě některé z těchto problémů řešíme v této diplomové práce. Popisujeme v ní algoritmus odvozený od původního Menclova algoritmu, který umožní zpracovat více typů use case než ten původní, a navrhujeme metriku, která měří kvalitu parse stromů. Tato metrika umožňuje výběr nejlepšího parse stromu use case věty z několika parse stromů vytvořených pomocí různých lingvistických parserů. To pomáhá eliminovat problémy s tím, že jeden parser vrátí chybný výstup.

Klíčová slova: případ užití (use case), zpracování přirozeného jazyka, pro-case

Title: Natural Language Processing of Textual Use Cases
Author: Jaroslav Dražan
Department: Department of Software Engineering
Supervisor: RNDr. Vladimír Mencl, Ph.D.
Supervisor's email address: mencl@nenya.ms.mff.cuni.cz

Abstract: Use cases written in a natural language are usually employed for specifying of functional requirements. The format of a use case is not standardized but use case sentences traditionally adhere to a simple structure and describe actions which are either communication actions (among actors and a designed system), or internal actions. The natural language is used because it is comprehensible for stakeholders and is universal enough to capture most of the requirements but it is difficult to analyze it in an automated way.

Vladimír Mencl employed state-of-the-art linguistic tools to extract a behavior of a system under design from textual use cases. The behavior specification is described in form of pro-cases. His work shows that this is possible but he met several issues. In this thesis, we solve some of the issues. We propose an algorithm based on the Mencl's algorithm which allows to process more use cases than the Mencl's one and we describe a metric which evaluate a quality of parse tree. The metric helps to select the best parse tree of a use case step from parse trees generated by different linguistic parsers. It addresses the issue of eliminating an incorrect parse tree returned by a single parser.

Key words: use case, natural language processing, pro-case

Table of Contents

Chapter 1

Introduction

1.1. Motivations

The complexity of software projects became quite a big problem. Functionality is even more required by software users but the systems must be developed within a shorter and shorter space of time. To mange this complexity, many methodologies and software development processes have been introduced, such as the Rational Unified Process [21] or the Extreme Programming. Regardless of the applied process, its developers have to know "what" system shall do and "how".

The question of "what to do" is usually answered by users' requirements on the system. The process of requirement management is relatively well-explored. The most common type of requirements - functional requirements - are being kept in documents called use cases. The advantage of the use cases is that they are so widely used that many guidelines and recommendation for use case writers are proposed and the content of use cases is more or less unified.

The question of "how to do the system to meet the stakeholders' requirements" is solved in disciplines of analysis and design. The big issue of analysis and design is "where to start". The recommendation proposed in [9] is to use a technique of System Sequence Diagrams (SSD). The Sequence Diagrams are a part of the Unified Modeling Language [16] which shows a behavior of the system and communication of the instances in time sequences. SSD is the sequence diagram which captures and visualizes a scenario of a use case. The other issue is how to verify that the proposed solution meets the requirements. It is not an easy task because there are a plenty of use cases and the use cases are written in the natural language so it is difficult to automatize this task.

The possible way how to start with analysis and design is to convert the use cases into protocol use cases (pro-cases) [18]. The pro-cases capture the behavior of the system as defined in use cases in a compact format which is comprehensible for human readers. Moreover the pro-cases are based on regular expressions so they can be easily machine processed and can be converted into other useful formats, such as state machines or sequence diagrams, thus the SSDs can be also created from them.

It is useful to have pro-cases which describe the system but the question is how to do that. The one possible way is to extract them from use cases manually. The solution of this task is proposed in [18]. The better solution shall be to convert the use cases to pro-cases in an automated way. Vladimir Mencl [14] employed the state-of-the-art linguistic tools to fulfill this task. His work shows that it is possible to convert use cases to pro-cases in an automated way but still some open issues remain and it is the purpose of this thesis to solve some of them.

1.2. Goals

1.2.1. Prerequisites

In [14], Vladimir Mencl describes how readily available tools for natural language processing can be employed to accomplish transforming a textual use case into a pro-case in an automated way. The premises of the work are: 1) A step of use case describes either communication between an actor and SuD, or an internal action. 2) Each action is described by a simple English following a simple (SVDPI) pattern. For analysis of a use case sentence he uses a statistical parser [6] developed by M. Collins at the University of Pennsylvania. The parser has been chosen for its high accuracy (over 88% constituent accuracy and over 90% accuracy of dependencies on generic English text).

The key goal of this thesis is to prevent failure of analysis of a use case step described in [14] which can be a consequence of natural language parser error or because the use case sentence does not match the premises of the analysis. The thesis should be supported by proof-of-the-concept implementation and should not be tested only on the use case set of the original work but on new use case sets as well.

1.2.2. Objectives

1. The first goal is to propose a metric to evaluate how much the parse tree matches the rules for a use case sentence. The metric helps us to select the best parse tree which we obtain when employing two or more parsers to parse a use case sentence. This should minimize the risk that a use case step analysis returns no or incorrect pro-case action because a single parser returns an incorrect parse tree of the use case step.

2. The second goal is to propose an algorithm based on [14] which enables processing of more use cases than the Mencl's one. There is no exact guideline for use case writing but only some recommendations [5, 9] which can be sometimes contradictory. This causes that many use cases do not match the SVDPI pattern. Thus, they cannot be transformed by the algorithm proposed in [14]. The proposed algorithm should address mainly the following problems: A use case step starts with a condition, a use case sentence describes more than one action, or contains more than one direct and/or indirect objects.

1.3. Structure of the Thesis

In chapter 2, we provide the background of the thesis. We explain the core terms, such as use case and pro-case, and summarize the algorithm proposed in [14]. In chapter 3, we propose our solution of the goals given and in the chapter 4, we demonstrate it in the case study. In chapter 5, we evaluate our results and conclude our solution. In chapter 6, we propose the alternative ways how to solve the goals given and in chapter 7, we summarize the related work on extracting information from natural language requirements specifications.

Chapter 2

Background

2.1. Use Cases

Requirements specification is one of the disciplines of the software development process. It is a very important discipline because it specifies the criteria which must be fulfilled by the system under design (SuD) and underestimating of this discipline is one of the most important causes of the software projects' crashes.

The special kind of requirements on the SuD are so-called functional requirements. The functional requirements specify assumed behavior of the SuD and projected functionality to meet the stakeholders' needs. The other type of requirements are non-functional requirements which specify other constraints which must be met by the SuD like performance requirements, technological constraints, usability constraints, etc. The non-functional requirements can be captured using for example FURPS+ method [9, 21].

The most common way how to capture and manage the functional requirements is to write and keep them in the form of use cases. Use case is a document which describes the behavior of the SuD in the form of a communication between the primary actor and the SuD, and the SuD and the supplementary (secondary) actors. Moreover the use cases should capture the internal actions of the SuD which are done to ensure the needs (goals) of the other stakeholders than the primary actor [5, 9, 21]. The example of a use case can be seen in Fig. 1.

Actors are the entities with their own behavior, such as the persons (often identified by their roles), computer systems or organizations. The primary actor is an actor that triggers the use case (sometimes it is defined as an actor whose goal is fulfilled by the use case). The secondary (sometimes supplementary or supporting) actor is the actor that is involved in the use case, e.g., the *Credit card authorization center* is used by the use case "Process sale" [9] to validate a credit card.

Use Case UC1: PROCESS SALE
Primary Actor: Cashier
Supporting Actors: Accounting System, Inventory System
Main success scenario (or basic flow):
1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Extensions (or alternative flows):
*a. At any time, Systems fails:
  To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.
  1. Cashier restarts System, logs in, and requests recovery of prior state.
  2. System reconstructs prior state:
     a. System detects anomalies preventing recovery:
        i. System signals error to the Cashier, records the error, and enters a clean state.
        ii. Cashier starts a new sale.

3a. Invalid identifier:
  1. System signals error and rejects entry.

...

Figure 1. Textual use case borrowed from [9] (simplified version)

The template for use cases is not fixed but the common outline of use cases as proposed in [5, 9, 21] is:

1. **Name** - name of a use case; usually equals the name of the goal which the use case fulfills.
2. **Brief Description** - brief description of what the use case does.
3. **Primary Actor** - name of the primary actor of the use case.
4. **Triggering Action** - situations which cause the start of the use case.
5. **Basic Flow** - sequence of steps which describes the main success scenario of the use case.
6. **Alternative Flows (extensions and variations)** - sequence of steps which describes alternative or error situations.
7. **Special Requirements** - non-functional requirements specific for the use case.

The use cases as proposed above are used for capturing of the functional requirements of the designed system but they can be also used for the documentation of behavior of an existing system or they can describe the business processes (the use cases are then called business use cases).

2.2. Pro-cases

Pro-case (Protocol use case) is a notation for specifying component behavior proposed in [18] which is based on protocols [20]. The pro-cases allow us to capture a "whole picture" of the behavior of the system and to cover the interactions between the subsystems and internal actors with the parent system.

Pro-case specifies behavior in terms of atomic events emitted (!), absorbed (?), and internally processed (τ) by an entity; its syntax stems from regular expressions. A primary expression specifies a single event; among the operators are (in priority order): "*" for iteration, ";" for sequencing, and "+" for alternative [20].

In this work, we use "#" instead of "τ" and we call pro-case atomic events (e.g., ?SL.submitItem) as *pro-case actions*. A sample pro-case is shown in Fig. 2.

```
?SL.submitItem ; #validateDescription;
( #cond2a
+  (NULL + #cond2b ; !SL.providePriceAssessment ) ;
   ?SL.adjustPrice ; #validateSeller ; #verifySeller ;
   ( #cond5a
+  !TC.validateOffer ;
   ( #cond6a
   +   #listOffer ; #respondAuthorizationNumber)
   )
)
```

Figure 2. Automatically created pro-case borrowed from [14]

2.3. Use Case to Pro-case Conversion

In this section, we describe automated conversion of textual use case to pro-case as proposed by Vladimir Mencl in [14]. He employs the state-of-the-art linguistic tools to obtain a parse tree for each textual use case's step (fig. 3). Consequently, he analyzes the parse trees and acquires an action and its principal attributes for every parse tree. This analysis is based on uniform structure of the use case steps. Finally, he converts all the acquired actions into a pro-case.

The use case writing guidelines [5, 9] prescribe the uniform structure for a use case step. From these core works on writing use cases, Mencl derived premisses which use case step should follow to be processed correctly. The premisses are: (1) A step of a textual use case describes either (a) communication between an actor and SuD (a request being sent or information passed), or (b) an internal action. (2) Such action is described by a simple English sentence following a uniform pattern. The uniform pattern is defined as the SVDPI pattern - "*Subject ... verb ... direct object ... indirect object*".

To acquire a parse tree of a use case step, he employs the well-accepted Collin's statistical parser [6]. The Collin's parser is chosen for its high accuracy on generic English text (over 88% constituent accuracy and over 90% accuracy on dependencies) and for robustness of its parsing algorithm. Before the text can be processed by the parser, it has to be tagged with POS (part of speech) tags; the MXPost tagger [22] is employed for this task. In addition, the morphological tools [15] are employed to obtain the lemma (base form) of the word.

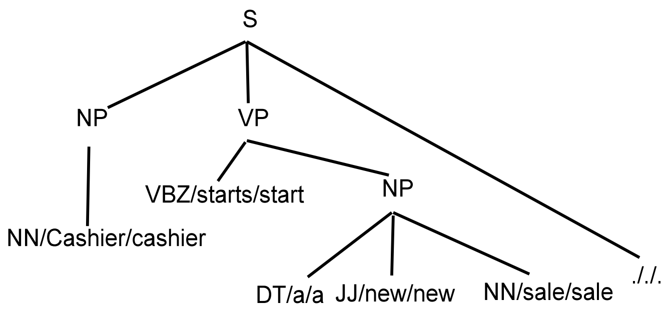

Figure 3. Parse tree of a "*Cashier starts a new sale*" (step 2 of use case UC1 in [9]). The first part of leaf node is a POS (part of speech) tag, the second is a word of the sentence and the third is a lemma of the word. S node represents a whole sentence, VP node a verb phrase, and NP nodes represent noun phrases of sentence.

The goal of a use case step analysis is to determine an action which is described by the use case step. Mencl distinguishes the following actions: an operation *request*, either *received* by the SuD from an actor, or *sent* by the SuD to an actor, or an *internal action*. Moreover a use case step can either change the flow of control in the use case, then the step describes *goto* action, or can terminate the use case, then the use case step describes a *terminate* action. The goto and terminate actions are overall called *special actions*.

In a sentence following the SVDPI pattern, a subject is the entity performing the action, a verb describes the action, a direct object describes the data being passed, and an indirect object (if specified) represents the entity receiving the request.

The most important issue of the analysis of a use case step is to determine the type of action. The action type depends on a subject and possibly on an indirect object of the use case step. If the subject is an actor or the keyword “user”, the step describes a receive action; if the subject is the SuD or the keyword “system” then the use case step describes either an internal action if the step contains no indirect object, or describes a send action if the step contains the indirect object - in this case the indirect object represents the receiver of the action. The event token (description of the action in terms of pro-cases) is constructed from the principal verb (usually predicate) and the direct (representative) object.

The special action is described by a simple sentence closely following a predefined pattern (recommended in [5]). The typically used constructs are “Use case aborts” (for a terminal action) or “Resume with step <label>” (goto action). In these patterns, the subject is a predefined keyword such as “Use case”, or the sentence has no subject at all. The verb is also one of just a few predefined words.

In an SVDPI sentence, the main sentence node (“S”) of the parse tree contains a *noun phrase* node and a *(main) verb phrase* node (in this order); the first noun phrase is the **subject** of the sentence. The **indirect object** of the sentence is determined as the noun phrase subordinate to the main verb phrase which matches the name of an actor or the “User” keyword. The principal **verb** is the headword of the top verb phrase; an exception to this rule is a padding verb. A padding verb is only a syntactical construct, while the actual task is described by the subsequent verb; in this case the principal verb is the next verb in the phrase. The identified padding verbs are “ask”, “be”, “select to”, and “choose to”. The multiple padding verb is not considered and the padding verb rule is not applied if there are no subsequent verb phrases in the sentence. The **direct (representative) object** is identified as the first basic noun phrase subordinated to the principal verb, the phrases already used as the subject or the indirect object are not considered.

Finally, when all use case steps are analyzed, the finite automaton representing the use case is constructed from the acquired actions and afterwards, the pro-cases are derived as a regular expression generating the same language as the automation.

Chapter 3

Use Case Step Analysis

This chapter describes precisely an algorithm for conversion of a use case step into pro-case actions. The algorithm solves the problems described in Goals section.

We assume that every use case step should be written to match the uniform sentence structure. First, we use the available linguistic tools to obtain possible several parse trees of the sentence. Consequently, actions and their principal attributes (i.e. action type, actor, and event token) are obtained from every parse tree. Finally, we select the parse tree which matches best our expectations about a use case sentence and then the analyzed step is converted to the selected parse tree's actions.

A proof-of-the-concept prototype has been developed to test the proposed algorithm. As testing data, we have used use cases borrowed from [9, 19, 21]. In further text we will call the testing data LarMenRup data (use cases). Details about LarMenRup data are placed in appendices.

The way to get actions and their principal attributes from a single parse tree (the solution of the second goal) is described in Sect. 3.4. The metric proposed to select the best parse tree (the solution of the first goal) is defined in Sect. 3.5.

3.1. Use Case Sentence Structure

The analysis of the LarMenRup use cases shows that premises defined in [14] are often broken by industry use cases. The probable reason is that recommendations for use case writers are sometimes ambiguous and use cases written in industry are sometimes finished under a time pressure or/and are written by novices. The example of ambiguous recommendations can be seen in Cockburn's [5] and Larman's [9] books. Cockburn says that a use case should capture just communication between the SuD and actors or an internal action but the use case borrowed from Larman contains steps describing communication between two actors. Moreover both books [5, 9] allow to break their recommendations about use cases in order to be more comprehensible for users and/or for programmers/designers.

Modification of Mencl's algorithm [14] ”to be able to process industrial use cases” is our key goal. First, we identify the most common causes of violations of Mencl's premises in the LarMenRup use cases and then we redefine the premises.

The Mencl's premise 1 is broken in the following cases: A use case step starts with one or more conditions, which say when to perform the step, or the step describes more than one logical action. If a use case step describes more than one logical step, each logical step is described by plain sentence and the plain sentences are compound in one compound sentence. If both plain sentences have the common subject, the subject can be excluded from the second and next sentences.

A primary actor often enters more than one object in one step and the SuD sometimes communicates with more than one supplementary actors in the same way. The recommendations [5, 9] say that in this case the events can be described in one step in order to write compact and not too long use cases. The use case steps following this recommendation break the Mencl's premise 2. The premise 2 says that each action is described by a sentence which follows a simple (SVDPI) pattern but in this case the sentence contains multiple direct and/or indirect objects. The other violation of the premise we have observed is the usage of passive voice. It is strictly recommended to avoid the use of passive voice in use case steps by guidelines [5, 9]. For this reason we consider the usage of passive voice as an error.

When parsing the LarMenRup use cases we have detected that the used linguistic parsers have problems with parsing of long sentences or of sentences containing parenthesis, quotes, nouns in their possessive form, or words containing “-”. The parsers returned incorrect parse trees of such sentences. It has caused us serious problems with use case step processing.

To be able to process industrial use cases and to avoid problems with parsers mentioned above, we define new premises of use case step structure based on [14]. The analysis of a use case step, which is described in this chapter, assumes that every use case step keeps the premises. The premises are:

Premise 1: A step of a textual use case is a single sentence. It is a plain, compound, or complex sentence.

Premise 2: A step of a textual use case starts with zero to n conditions and describes (a) either one or more communications between an actor and SuD, (b) one or more internal actions or (c) special actions.

Premise 3: A use case step is an English sentence and actions (a) and (b) as defined in premise 2 follow the uniform SVDPI pattern as defined in [14] except the sentence can contain multiple direct and/or indirect objects.

Moreover we define an additional requirement which can be assured by a simple transformation: A step of a textual use case does not contain parenthesis, quotes, possessive form of nouns, and words separated by "-" and "_".

The premise 1 and the additional requirement are important to avoid problems with parsers but many use cases breaks them (even our LarMenRup use cases do so, see Fig. 4) and there is no reason why use case writers should keep them. To process the use cases which break the premise 1 and the additional requirement we propose generic transformation which changes use cases to match them.

The premise 1 is broken if and only if a use case step is described by more than one sentence. Our transformation keeps the first sentence and deletes the others. This approach is possible because the other sentences do not usually describe other actions but only refine the action(s) described by the first sentence (e.g., "*System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.*" borrowed from [9]). The better approach would be splitting the use case step to separate steps (one sentence to one step) to avoid problems when the sentences describe more than one action.

To keep the additional requirement the transformation does the following modifications. It deletes the text in parentheses (including the parentheses). The text in parentheses only refines the use case step and would not be used by our algorithm. The text in quotes is concatenated (the first letter of every word except the first one is capitalized) and the quotes are deleted. The words containing "-" and "_" are processed in a similar way as the quoted text; "-" and "_" are deleted and the words are concatenated (see Fig. 4). Nouns in a possessive form are deleted.

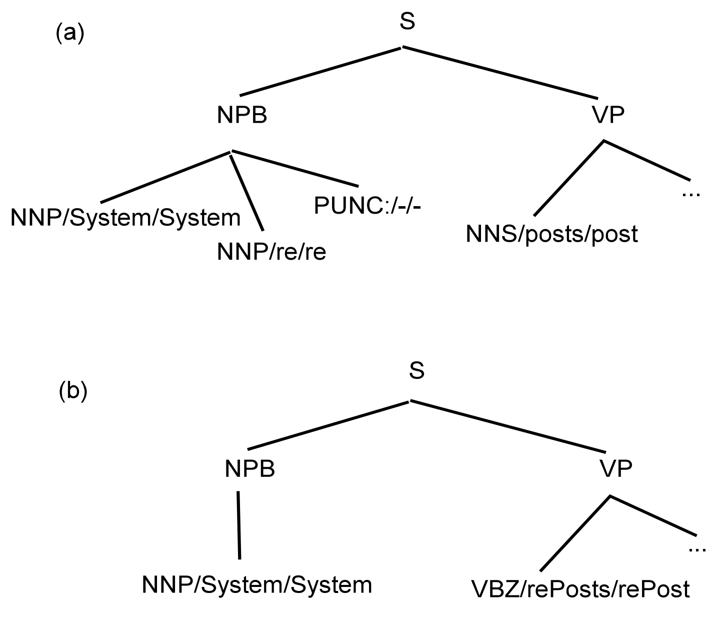

Figure 4. Extract of parse trees of "*System re-posts the story to the list of available content for paging initialization.*" (step 2a7 of use case CSPS - Approve Story in [21]).
This figure shows fragments of the same use case step before (a) and after (b) removal of "-" from the verb re-post.

3.2. Linguistic Tool Employment

We need to acquire parse trees of every use case sentence. To acquire the parse trees we employ the same linguistic tool and in the same way as in [14], but we employ not only the Collin's parser [6], but also two more parsers - the Charniak's [4] and the Dan Bikel's [2] ones. We use multiple parsers to acquire multiple parse trees for every use case step. It should increase the probability to obtain at least one correct parse tree. We hide identity of the parsers which have created the parse trees shown in the presented figures as we do not aim to compare the quality of the individual parsers and the identity does not impact our results.

There is no specification of parse tree format so every parser uses its own one. After obtaining the parse tree from the specific parser, we modify it to match our unified format. Luckily, all three parsers return the tree in Lisp notation and the set of possible tree nodes' types is derived from the Penn Tree Bank node set [1] so we use the Penn Tree Bank notation.

The difference between our format and the format used in [14] is that Collins[6] adds a headword to every non leaf node of a parse tree. The headword is the most important word of the node's subtree and it is often used in Mencl's algorithm. Because the Collin's parser is the only one which adds headwords to parse trees, we must exclude the headwords from the parse trees created by the Collin's parser and propose the further parse tree processing not using them.

3.3. Actions

When converting a use case into a pro-case, we retrieve information needed for pro-case actions creation from use case steps. This work is not focused on pro-case creation but the main goal is to improve use case step analysis to be able to process use cases which do not match the premises of the [14] and to retrieve more information about the actions than the analysis proposed in [14].

The output of our algorithm is not supposed to be pro-cases but the information acquired by analysis of a use case step. We keep the information in elements which we call actions. The actions have principal attributes holding the acquired data. Our actions extend pro-case actions, i.e., if a use case step describes a pro-case action then the acquired action keeps the same data as the pro-case action (thus, we call the elements keeping obtained information *actions*).

Not all the use case steps describe pro-case actions or we cannot always acquire the described pro-case actions for some reason (e.g., because of an incorrect sentence structure or linguistic parser failure). We need to keep the information about such use case steps so we define also actions which do not correspond with pro-case actions. Several kinds of such actions have already been defined in [14]; they are called special actions there.

Mencl in his work [14] does not define the actions so precisely as we do and he does not make any difference between pro-case actions and the actions as we have defined in this section. He does it for simplicity because he does not detect the error states (he always tries to convert the use case step although the result pro-case action is obviously incorrect) so he does not need actions to mark the error state and one of his actions is directly mapped to one pro-case action (except the special actions). In our work, one action can describe more than one pro-case action or it keeps information about special or error states. When using terms "action" later in this work, we always mean our changed/extended actions as defined in this section.

In [14], the following types of actions are introduced: *operation request* actions (*send* or *receive*), *internal* actions, and *special* actions (*goto* and *terminate*). Moreover extension and variation conditions are used in the use case to pro-case conversion. In this work, we add some new action types and we change the principal attributes of the original ones. We also allow a use case step to describe more than one action. In this case, the list of actions acquired from a single step is called the *action set*.

Send, *receive*, and *internal* actions are used in the similar way as defined in [14]; Mencl's action consists of a subject, verb, direct object, and optionally of an indirect object. There is a difference in the fact that our *operation request* or *internal* action can contain zero to n direct objects. Furthermore we change the way to obtain a direct object which affects a format of the direct object (it consists not only of nouns but also of adjectives). If an action contains n (n>0) direct objects, it can be converted to n Mencl's original actions - one Mencl's action for one object. Thus, our action with more than one direct object is called *compound* action. E.g., our action *#createObject1_Object2* can be converted into two Mencl's actions *#createObject1*, *#createObject2* (the syntax used in this example is described at the end of this section). We allow an action to contain no direct object because we want to understand use case steps containing no object to pass through like *"User logs in."*.

We use the *special* (*goto* and *terminate*) actions exactly in the same way as defined in [14].

In our work, we do not directly convert use cases to pro-cases but we try to identify use case sentence structure violation and we also focus on selecting the best parse tree of the same use case sentence. Therefore we need to define new auxiliary actions which have no usage in use cases to pro-cases conversion but which help us with these tasks. The new actions are:

Illegal action - When we find out that the use case step violates the use case sentence structure, we convert the use case step to this action. The principal attribute of illegal action is a message which describes how the sentence violates the structure.

Condition action - In our work we allow to define conditions at the beginning of the use case step. We convert any equal condition to the condition action. The principal attribute of this type of action is a parse tree of the condition (a subtree of the original use case sentence parse tree).

Unknown action - When we cannot assign any of the actions defined above to a use case step or to its part, we assign an unknown action to the use case step or to its part. In other words, this action means that we do not know what to do with a use case sentence or its part.

When presenting our action sets, we use the following format: *[action_1,...,action_n]*. The format of a single action is the same as a pro-case action format used in [14] except we allow a single action to contain more direct objects; if a single action contains more than one direct objects, the format of an event token is: *verbObject1_Object2..._ObjectN*. A *condition* action is described by a tree in a lisp like notation which corresponds to an identified condition.

3.4. Parse Tree Analysis

In this section, we propose an algorithm which obtains one or more actions (an ordered list of actions) from a parse tree of a use case sentence. The proposed algorithm solves the second goal of the thesis and can replace the original Mencl's algorithm if only one parse tree of a use case step is available. The algorithm for selecting from more than one parse tree is described in Sect. 3.5.

3.4.1. Analysis Brief Description

The input of the analysis is a single parse tree. The result is an ordered list of actions which were obtained from the parse tree. The key idea of the algorithm is to divide the parse tree to subtrees which describe single actions and then convert the subtrees into the actions. Furthermore we continuously test the correctness of the parse tree and if some error is identified, an *illegal* or *unknown* action will be returned. The next paragraph describes the algorithm more precisely.

First we test whether the given parse tree can match a use case sentence pattern or not. If the test fails, an *unknown* action will be assigned to the parse tree and we will terminate the analysis. Otherwise, we try to find all the conditions of the use case step (if any) and look up all plain sentences which should describe a single action. We process any equal sentence. Processing of the plain sentence returns an action and its principal attributes for the given sentence. Moreover a condition action is being returned for each found condition.

3.4.2. Use Case Sentence Test

From premises 1 and 2, we know that each use case step should be a plain sentence, a compound sentence, or a sentence starting with condition. This sentence parse tree's root should be node ("S"). If the root node is not ("S"), the analysis finishes and action is not recognized (we assign an *unknown* action to this sentence).

The other root nodes of a parse tree which we have seen during our experiments were ("FRAG") and ("NP") nodes in case that a parser did not find the predicate of the sentence (fig. 5). The cause why the parser cannot identify the predicate of the sentence is that it considers a verb as a noun (the verb "*requests*" in Fig. 5). This is the limitation of this algorithm. The limitations are described in Sect. 3.6. in more details.

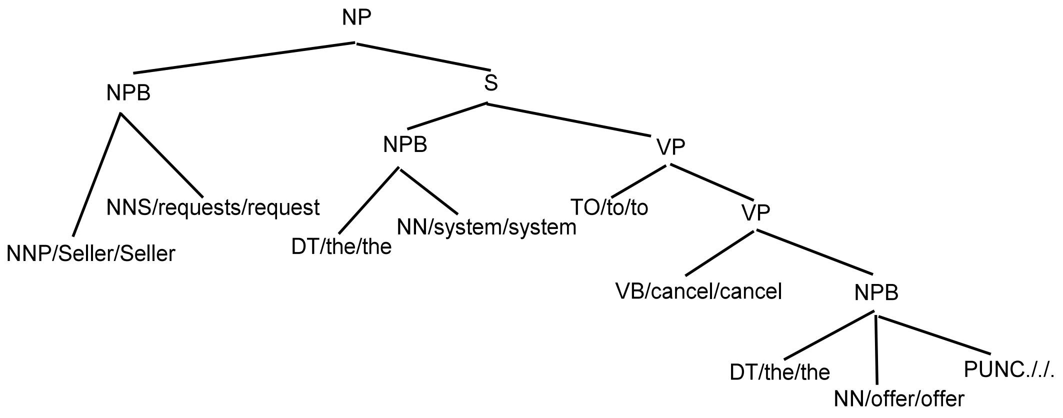

Figure 5. Incorrect parse tree of "*Seller requests the system to cancel the offer.*" (Step 2 of use case M2 in [14]). The reason of the defect is tagging the verb "requests" as a noun.

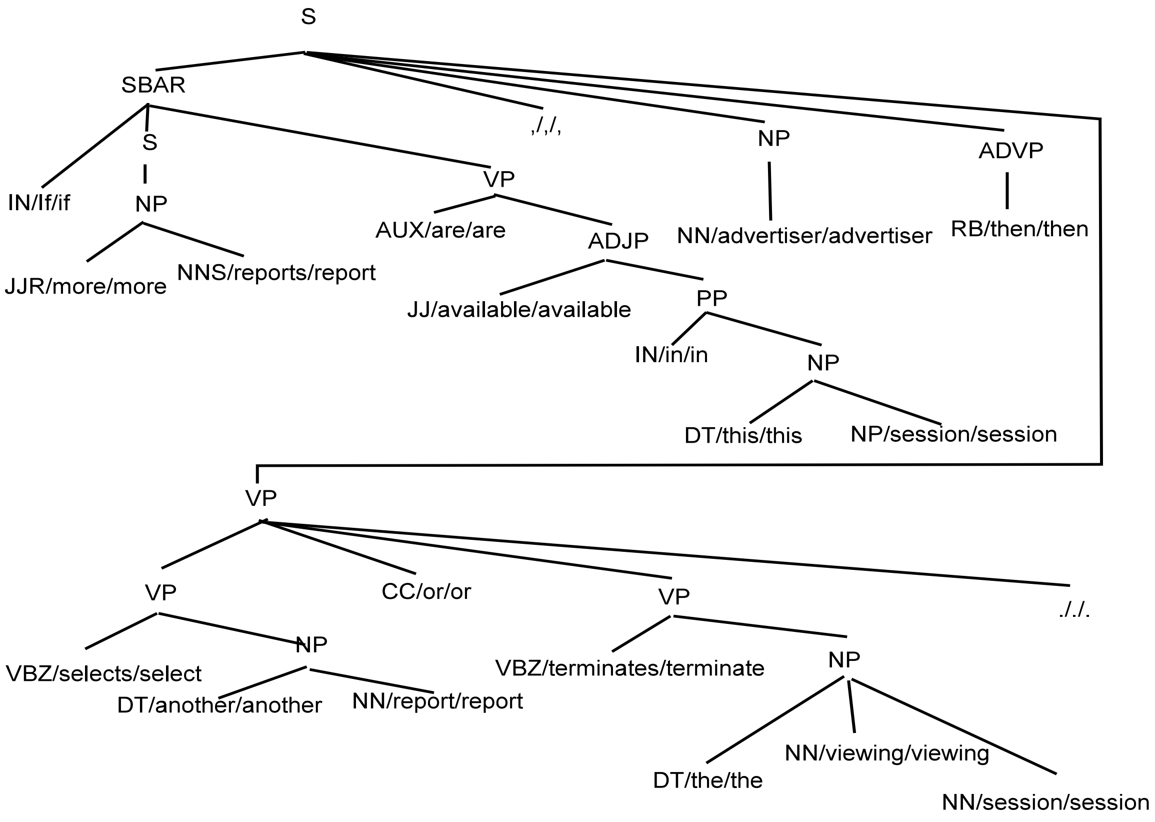

Figure 6. Parse tree of "*If more reports are available in this session, advertiser then selects another report or terminates the viewing session.*" (step 10 of use case CSPS - Print Advertiser Reports in [21]). This figure shows a parse tree of a use case step starting with a condition.

3.4.3. Conditions Searching

We assume that possible conditions are hypotactic sentences placed at the beginning of the use case sentence. The condition should start with a subordinate conjunction like “if”or “when”. The hypotactic sentence is marked with (“SBAR”) node and subordinate conjunction with (“IN”) node (fig. 6). Thus, all conditions selected by the algorithm are all subtrees of the parse tree root node whose root node is (“SBAR”) and the first child node is (“IN”) node, and which are placed at the beginning of the sentence and are separated by a comma - (“,”) or (“PUNC”) node - or by a coordinating conjunction - (“CC”) node. We assign a *condition* action to every identified condition. In Fig. 6, the condition “*If more reports are available in this session,*” is identified.

3.4.4. Plain Sentences Acquirement

We consider two possibilities. First, a use case sentence is a plain sentence (fig. 7); its parse tree root node has no child (“S”) nodes. In this case, the whole sentence is selected. Secondly, the sentence is a compound (fig. 8) sentence; its parse tree root node has at least one child (“S”) node. In this case, every child sub tree starting with (“S”) node is selected as a plain sentence. The action set is then obtained from every selected plain sentence. The result of the processing is the union of the obtained action sets. In Fig. 8, two plain sentences are identified - “*Customer pays*” and “*system handles payment*”.

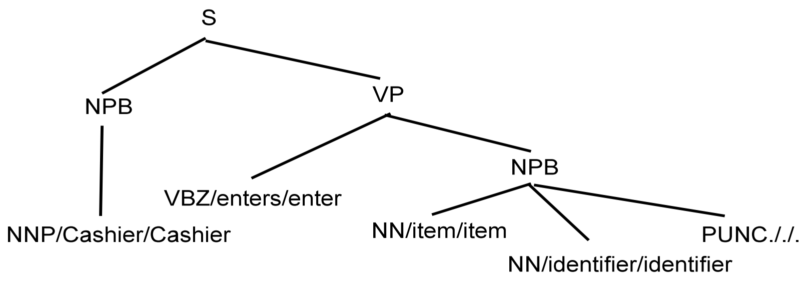

Figure 7. Parse tree of “*Cashier enters item identifier.*” (step 3 of use case UC1 in [9]). This figure shows a parse tree of a plain sentence.

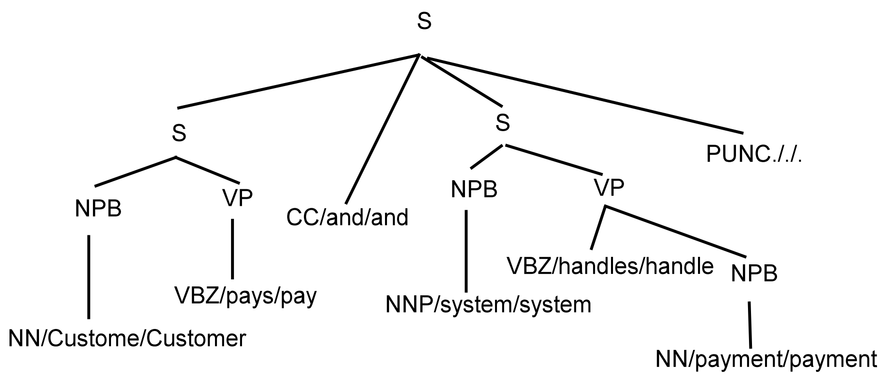

Figure 8. Parse tree of "*Customer pays and system handles payment.*" (Step 7 of use case UC1 in [9]). This figure shows a parse tree of a compound sentence.

3.4.5. Plain Sentence to Actions Conversion

From premises 2 and 3, we conclude that a plain use case sentence defines either an operation request (one *receive* or more *send*) actions, an internal action, or a special action. The following two sections describe how to acquire actions of these types from a plain use case sentence.

3.4.6. Operation Request and Internal Actions

Likewise in [14], if a use case sentence defines an operation request, we assume that the subject of the sentence is an entity which initializes the action, the direct objects of a sentence describe data being passed and the indirect objects of the sentence are the entities receiving the request (if specified). The name of the action is determined by the predicate of the sentence.

The original algorithm proposed in [14] identifies a subject, verb, direct and indirect objects of the sentence and then constructs one action from them. If the sentence contains more than one predicate or direct or indirect object, the original algorithm will select only the first one.

We try to find all predicates, direct and indirect objects. If the sentence contains a multiple predicate or a multiple indirect object, we will construct an action for each of them. Our operation request and internal actions allow to keep a multiple direct object for the action, so if the sentence contains more than one direct object, we keep all of them as an attribute of the action.

Subject. We identify the subject of the sentence in the same way as proposed in [14]. If the subject is found and if it is an actor or a predefined key word ("*user*" or "*system*"), we will continue with searching operation request or internal action principal attributes, otherwise we will try to identify the use case sentence as a special action. We do not allow a use case sentence to have a multiple subject otherwise we select only the first one and the others are supposed to be ignored.

The only difference between our approach and the Mencl's approach is that in order to find the subject we require an actor's name or the predefined key word to match the identified noun phrase exactly but Mencl requires only the actor's name or keyword to be involved in the noun phrase. We are more strict because the analysis of the LarMenRup data shows that the subject does not match the actor's name or the key word only if the parser marks the predicate of the sentence (it follows the subject) as noun and not a verb (this problem is being discussed in Sect. 3.6). Because the predicate of the sentence cannot be identified correctly, we consider further analysis useless. The only exception is, that if the badly marked predicate is the padding verb, the analysis can succeed but we will not handle this situation.

Predicate. We describe in this paragraph not only how to obtain the predicate or predicates of the plain use case sentence but we also describe how to acquire the verb phrases from which to read direct and indirect objects. We usually get the direct and indirect objects from the same verb phrases but they can differ in case that the predicate of the sentence is a padding verb [14] and we can find a verb different from the padding one. The indirect objects are always gotten from the predicate verb phrase but the direct objects are gotten from the verb phrase containing another verb if any (otherwise the predicate verb phrase is used). We also describe how to recognize usage of passive voice or of modal or auxiliary verb.

As mentioned in Sect. 3.1, a plain use case sentence usually describes one logical step but can also describes more logical steps in case that the logical steps have a common subject. Thus, first we identify how many logical steps the sentence describes and if it describes more steps, we will identify the predicate and related verb phrases from each of them. The precise algorithm is defined in the following paragraphs in this section.

A plain sentence's parse tree should have the following structure: the root node is ("S") and its children are a noun phrase ("NP*") and a verb phrase ("VP"). If the structure differs, we do not recognize the action (we assign an *unknown* action to the plain sentence). Otherwise we select all predicate verb phrases. If the sentence contains more predicates which have the common subject (e.g., "System validates user and returns required data."), the root verb phrase contains no child verb node and it contains verb sub-phrase for each predicate (fig. 9) so we will select all verb sub-phrases, otherwise we will select the root verb phrase. In fig. 9, two verb phrases are identified - "*creates first report*" and "*prompts user to save or view*".

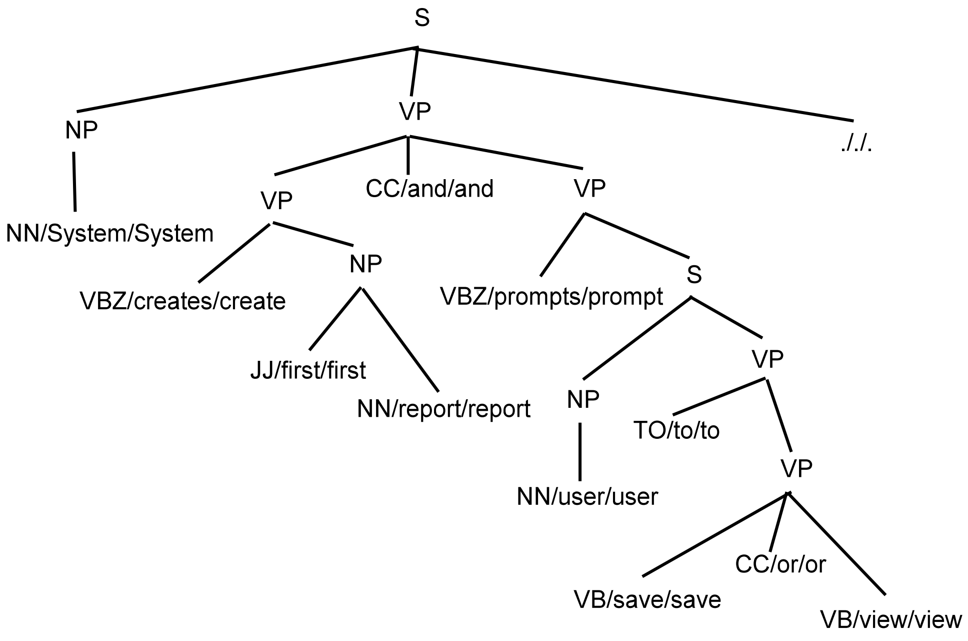

Figure 9. Parse tree of "*System creates first report and prompts user to save or view.*" (step 4 of use case CSPS - Read Content on Web site from [21]).
This figure shows a compound sentence with a common subject ("System").

We process all predicate verb phrases separately; we identify the predicate of every predicate verb phrase and if the predicate is a padding verb, we will try to find another verb and its verb phrase which will be later used for finding direct objects of the sentence. Because of this approach we can miss the direct objects which are tied to more than one verb. This limitation is being discussed in Sect. 3.6 in more details. Notice that an indirect object is always bound to just one verb so this limitation does not affect indirect object processing.

If the verb phrase subtree contains a verb node (i.e., ("VB*"), ("AUX"), or ("MD") node) as a child node, the verb node is the predicate of the sentence. If the predicate is a common verb (("VB*") node) and it is not a padding verb or the verb "be", the predicate is the main verb which will be used as returned action attribute and the given verb phrase is the verb phrase where we look up the direct and indirect objects. If the predicate is the verb "be" or if it is a modal (("AUX") node) or auxiliary verb (("MD") node), we evaluate the use case step as invalid (we assign the illegal action to the processed plain use case sentence). A parse tree of a sentence with a modal verb ("*can*") is shown in Fig. 10.

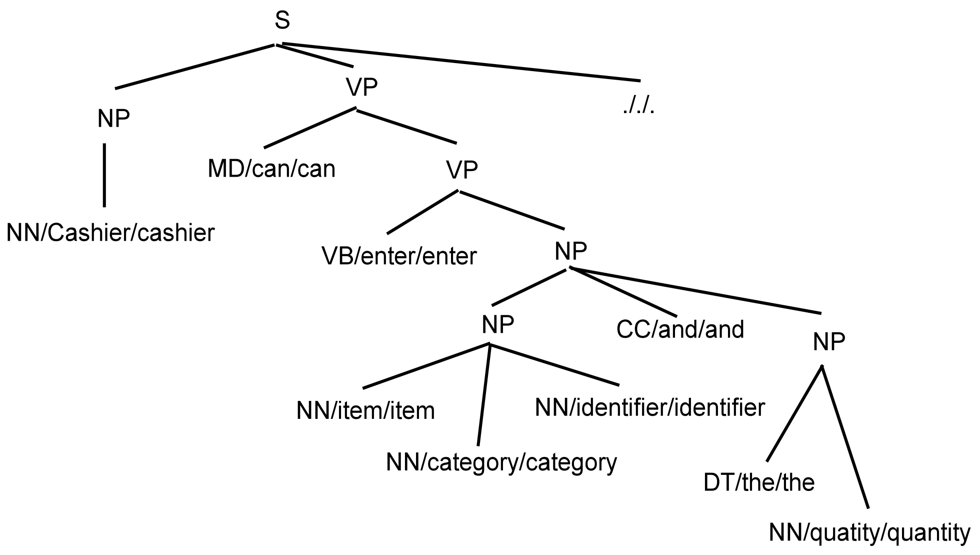

Figure 10. Parse tree of "*Cashier can enter item category identifier and the quantity*" (step 3b1 of use case UC1 in [9]).
This figure shows a sentence with modal verb "*can*".

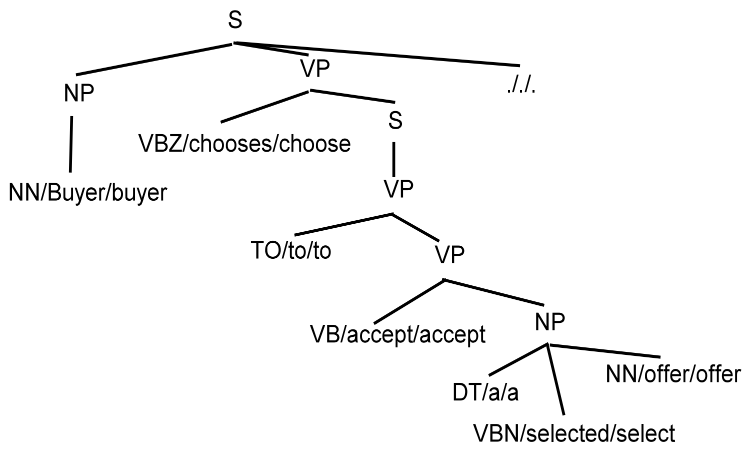

Figure 11. Parse tree of "*Buyer chooses to accept a selected offer.*" (step 1 of use case M3 in [14]).
This figure shows a sentence with a padding verb "*choose*".

If a padding verb is used (fig. 11 - the padding verb "*choose*"), we try to find the first descendant verb phrase of the processed verb phrase which contains a common verb ("VB*" node) as a child ("*accepts a selected offer*" in Fig. 11). If a such verb phrase has not been found, the analysis continues as if the padding verb is a common verb as described above. If such a verb phrase has been found, the verb node will be taken as the new predicate of the sentence and the new verb phrase will be used for looking up the direct objects of the sentence. The indirect objects are selected from the original verb phrase - e.g., in the sentence "*System asks the Supervisor to validate the seller.*" The indirect object is "*Supervisor*"and it is a part of the main verb phrase containing the padding verb "*ask*".

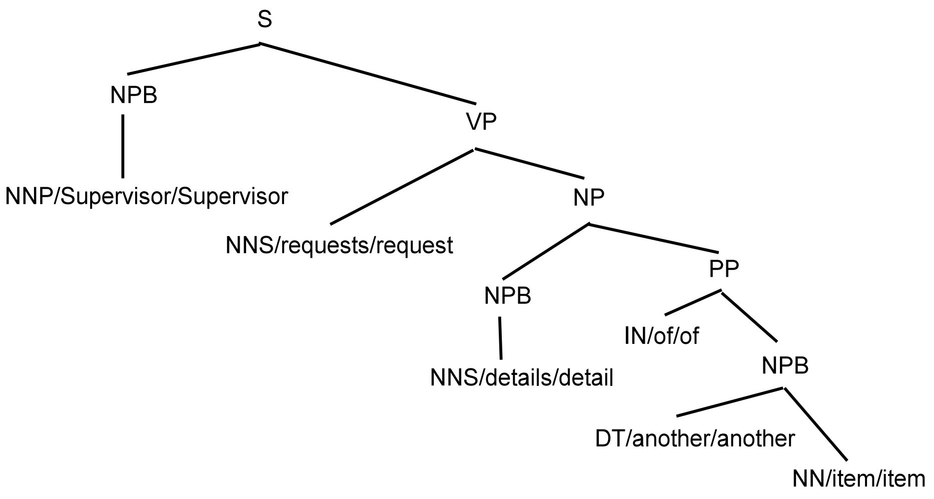

Figure 12. Parse of "Supervisor requests details of another item" (step 2b of use case SU2 in [14]).
This figure shows sentence with a predicate "*request*" tagged as a noun.

Sporadically, the linguistic parser returns the parse tree where the predicate verb phrase is marked correctly but the predicate is marked as a noun and not as a verb (fig. 12). We make a little hack to the algorithm which corrects it. When we identify the predicate phrases of the sentence but there is no verb child of the predicate verb phrase, we try to find the first ("NN*") child node (the node "*NNS/requests/request*" in a tree in Fig. 12). If we find any, we change it to the ("VB") node, and the algorithm continues with a predicate identification (the node "*NNS/requests/request*" is changed in the node "*VB/requests/request*"). This hack helped us to process more sentences of the LarMenRup use cases but it caused no invalid identification.

The algorithm of searching a verb described in this section seems similar to the algorithm described in [14] but there are some hard to notice but very important changes so we emphasize them in this paragraph. The Mencl's algorithm considers only the root verb phrase as a predicate verb phrase so it ignores multiple predicates. His algorithm obtains the predicate of the sentence from a headword of the verb phrase. The parse trees, which are processed by our algorithm, do not contain headwords (sect. 3.2) so we have to obtain a verb node with a predicate. Our approach enables us to check the type of the verb and if it is a modal or an auxiliary verb we know that the sentence breaks our premises (as well as Mencl's in this case) so we can use this sort of information in further processing or to inform the user. The last but not least difference is that if the padding verb is used and the alternative verb is found, we then search the direct objects in the alternative verb's parent verb phrase but Mencl searches them in the original verb phrase. We suppose our approach to be better because the direct objects are tied to the verb describing the action (the alternative verb) and not to the padding verb. Because the alternative verb's parent verb phrase is descendant of the original verb phrase, both approaches can select the same direct objects if the original verb phrase does not contain any noun phrase which differs from the noun phrase related to the indirect objects of the sentence (more about selecting the objects of the sentence in the following sections).

Indirect object. Indirect objects of the use case sentence identify the entities which receive the request. In general, it is very difficult to identify the indirect objects (linguistic approaches to this and the similar task are being discussed in Sect. 6.3). We do not parse the general sentences because we know that the indirect objects of the use case sentence, if any, are secondary actors. Thus, the indirect object is represented by descendent noun phrase of the predicate verb phrase which matches the name of a secondary actor (fig. 13). The way to find a predicate verb phrase has been described in the previous section. The precise algorithm identifying indirect objects will be described in the following paragraphs.

First we try to find the first indirect object ("*external Accounting System*" in Fig. 13). The first indirect object is the first noun phrase which is subordinated to the predicate verb phrase and which corresponds to an actor. A noun phrase corresponds to some actor if any of the noun phrase's children are nouns ("NN*") and a concatenation of some actor name is a substring of the concatenated nouns and adjectives ("JJ") lemmas which are the children of the noun phrase (this is a similar algorithm as proposed in [14]).

If we find the first indirect object, we try to find all indirect objects at the same level of the parse tree ("*external Accounting System*" and "*Inventory System*" in Fig. 13). All the selected indirect objects are noun phrases which are children of the first indirect object's noun phrase's parent and which correspond to some actor (i.e., at least the already found indirect object is selected).

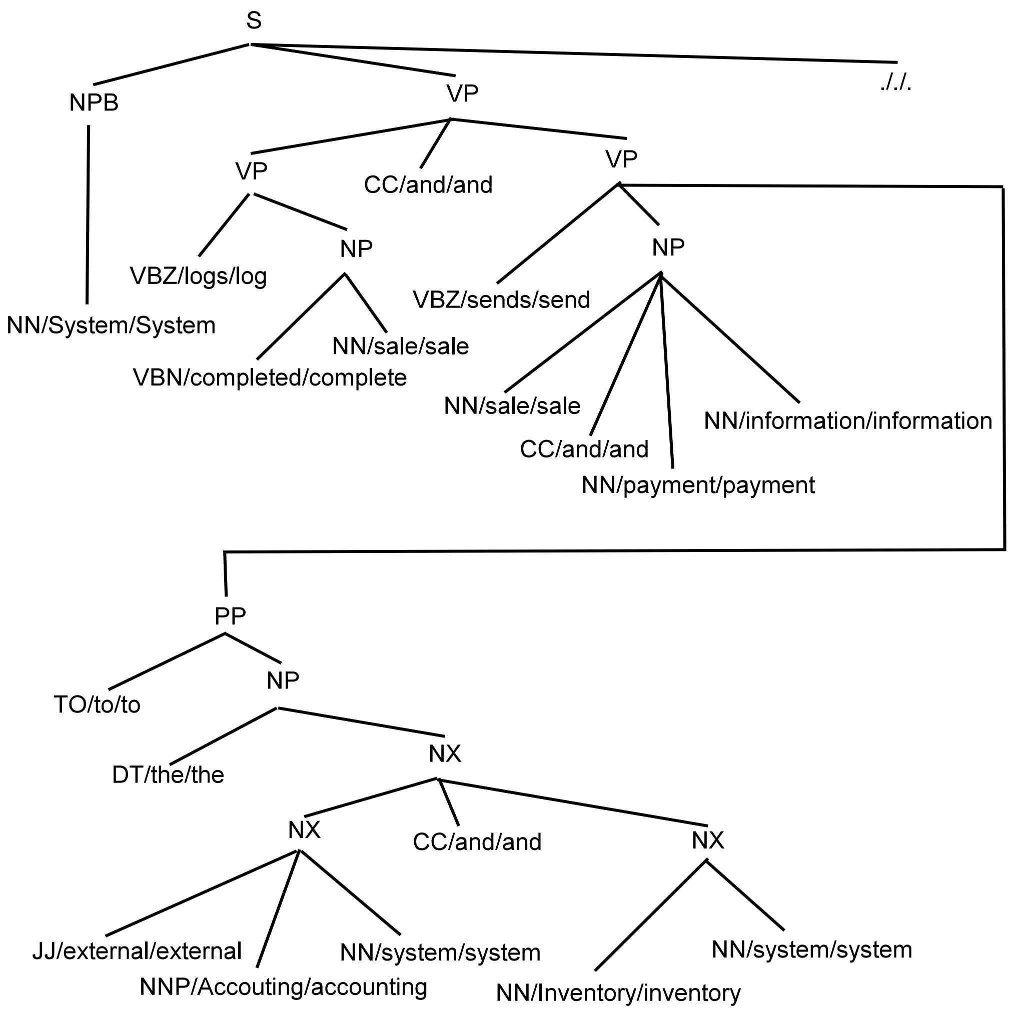

Figure 13. Parse tree of "*System logs completed sale and sends sale and payment information to the external Accounting system and Inventory system*" (step 8 of use case UC1 in [9]). This figure shows a sentence with two indirect objects: "external Accounting system" and "Inventory system".

We select only the noun phrases which are at the same level as the first identified indirect object and we do not consider other noun phrases which can also correspond to an actor. The reason is that the structure of the sentence is not usually simple and the sentence contains other parts as subordinated relative clauses etc. which often clarify the meaning of the step to the human reader. These parts of a sentence can contain an actor's name as well but these names are not the indirect objects of the sentence. The first identified indirect object does not have to be the indirect object as well, but this is the approach borrowed from [14] which usually works (exceptions are described in Sect. 3.6) because the indirect objects are usually placed at the beginning of the sentence (see the premise 3).

The main difference between our approach and the approach proposed in [14] is that we do not search only the first indirect object but we try to find the other indirect objects. Moreover we allow adjectives to be a part of an actor's name.

Direct object. The direct objects, if any, represent the data being passed through by a request. The identification of the direct objects and mainly their distinguishing from the indirect objects of a general sentence is not a simple task (see Sect. 6.3). Our approach presumes that the plain use case sentence has the simple structure (see premise 3) and that we have already identified the indirect objects before searching the direct objects. The idea of our approach is to find the first direct object and then look up all objects at the same level (fig. 14). The algorithm is described in more details in the following paragraphs.

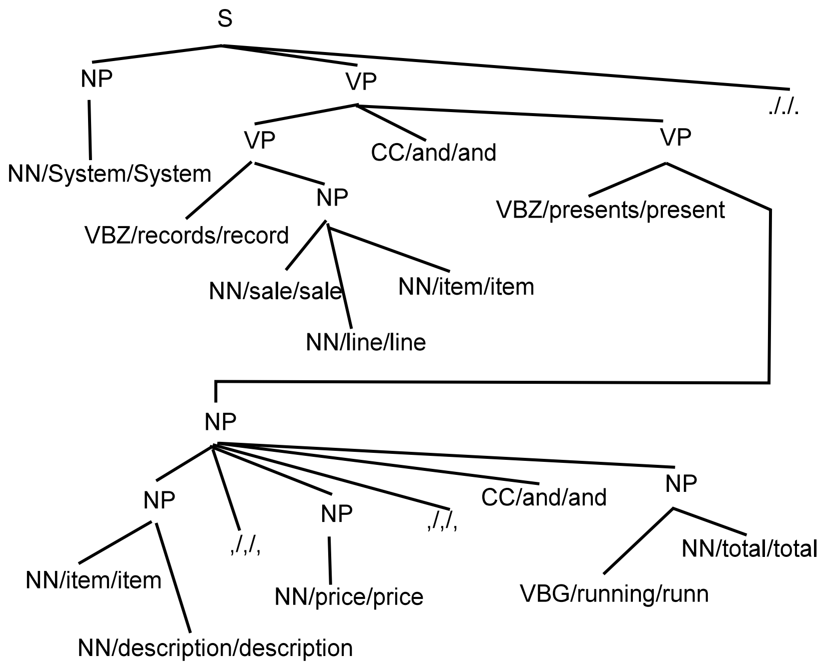

Figure 14. Parse tree of "*System records sale line item and presents item description, price, and running total.*" (step 4 of use case UC1 in [9]). This figure shows a sentence with multiple(triple) direct object "*item description, price, and running total.*".

We try to find all direct objects for every verb phrase which was selected by the algorithm for searching the predicate of the sentence (Details can be found in the previous section which describes this algorithm). We try to find the first direct object. The first direct object is the first noun phrase which is subordinate to the verb phrase and which matches the direct object pattern ("*item description*" in fig. 14). A noun phrase matches the direct object pattern if any of the noun phrase's children are nouns ("NN*") and the noun phrase or some of its siblings in the parse tree are not noun phrases identified as the indirect objects. This is a similar algorithm as proposed in [14] except we also test siblings of the candidate noun phrase not to be the indirect objects. If we find any, we try to select all noun phrases which are children of the first noun phrase's parent and which match the direct object pattern (i.e., at least the first direct object is selected); in Fig. 14, "*item description*", "*price*", and "*running total*" are selected.

The main difference between our approach and the approach proposed in [14] is that we try to find all direct objects but Mencl selects only the first one and if the predicate of the sentence is the padding verb, we look up the direct objects in the verb phrase of the alternative verb's parent's verb phrase (advantages of our approach are being discussed in foregoing section which describes selecting the predicate). Our algorithm always finds at least the same direct object as the original algorithm (except when a padding verb is used) but it can find other direct objects, if any. Reasons why other direct objects searching does not succeed are being discussed in Sect. 3.6. Thus our algorithm for finding direct objects is always at least as good as the Mencl's one.

3.4.7. Special Actions

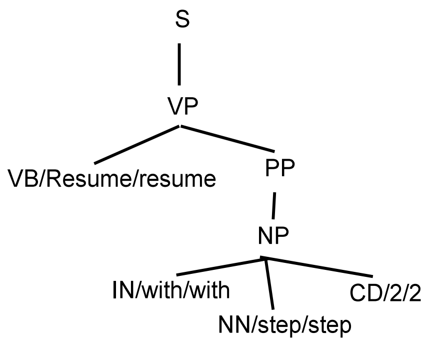

Figure 15. Parse tree of "*Resume with step 2*" (step 2b2 of use case M2 in [14]). This figure shows a use case step describing a goto action

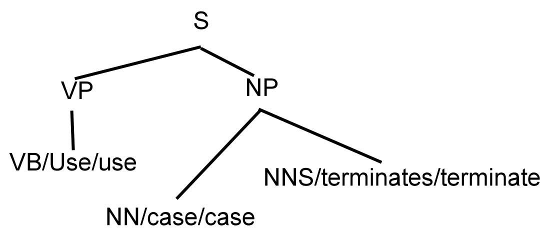

Figure 16. Parse tree of "*Use case terminates*" (step 1a1 of use case CS8 in [14]). This figure shows an incorrect parse tree of a use case step describing a terminate action.

If a plain use case sentence cannot describe an operation request action or an internal action (fig. 15), i.e., no subject of the use case sentence is found, or if the subject is not an actor or a special keyword ("*user*" or "*system*"), we try to process the sentence with the algorithm for identifying special actions proposed in [14] with the exception that we cannot use a headword of verb phrases (sect 3.2). Thus, we search the main verb of the verb phrases as mentioned in Sect. 3.4.6. If the sentence does not describe a special action (or its parse tree is incorrect, see Fig. 16) and no certain violation of the premisses is recognized, the analysis of the plain use case sentence fails (we assign an *unknown* action to the plain sentence).

3.4.8. Illegal Use Case Sentence Detection

One of the extensions of the algorithm proposed in [14] is our ability to identify some common violations of our premises caused by the use case writers (not by the linguistic tools). We attempt to identify the cause of the violation rather than the violation itself. If the sentence does not match the SVDPI pattern or the pattern for the special actions, we know that it is the violation of the premises 2 or 3 but that is not any interesting information. We want to recognize which recommendation for the use case writers was broken, if possible, to be able to recommend the use case writer how to correct the violation of our premises.

The identification of the violations is done continuously during a use case sentence analysis. If some violation is found, the algorithm provides a reason (it assigns the illegal action to the use case sentence or its plain sub-sentence) and terminates processing of the sentence or of its plain sub-sentence. If the algorithm does not recognize the concrete cause of the violation, it assigns an *unknown* action to the sentence or its sub-sentence and terminates its processing.

Notice that although an illegal action is assigned to a sub-phrase, we continue in processing of other sub-phrases. E.g., in the sentence "*User enters data and the data are processed by system.*", we correctly recognize the *send* action "*?US.enterData*" and we also recognize the usage of passive voice in the second sub-sentence. This helps us to identify more errors in one sentence or we can help the user at least with conversion of correct sub sentences.

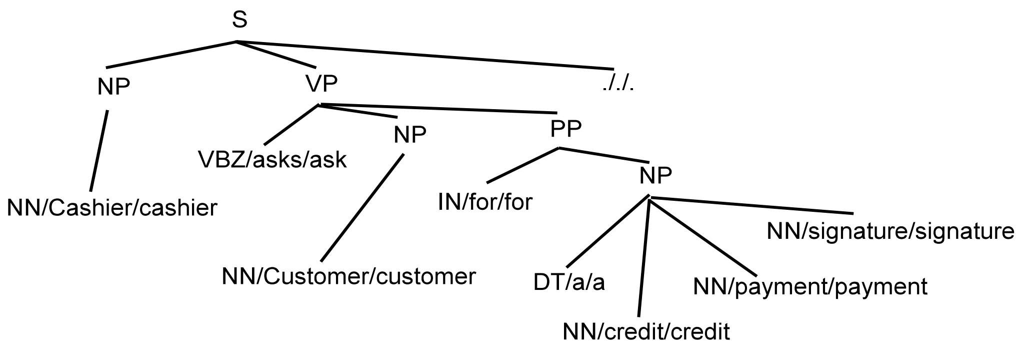

Figure 17. Parse tree of "*Cashier asks customer for a credit payment signature.*" (step 7b6 of use case UC1 in [9]).
This figure shows a use case steps describing actor o actor communication.

We can identify the following causes of our premises violations:

- Incorrect special action - no use case step number found, an illegal terminate/goto action verb found, and an illegal special action subject found.
- Actor to Actor communication (see Fig. 17).
- Usage of a modal or an auxiliary verb or usage of passive voice. This issue is described in Sect. 3.4.6 (*predicate* subsection) in more details.

We focus on identification of problems caused by a use case writer and not on problems caused by the linguistic tools because the use case writer can correct his/her mistakes but nobody (who transforms use cases to pro-cases) can correct the linguistic tools. The reason why we do not try to identify more problems like internal actions of actors or communication between entities which are not the actors of the processed use case is that we do not fully understand the use case sentence. To be able to understand the use case sentence much more we should consider other parts of the parse tree. This would cause that our algorithm will rely more on the correctness of the parse tree and become less robust because of the precision limits of the linguistic parsers.

Let us consider the example which shows why the detection of other violations of our premises caused by a use case writer is complicated. If we want to detect that the communication is initialized by the other entity than an actor we could propose the following simple algorithm.

If the subject does not match an actor's name and the sentence does not describe a special action, the step describes the communication initialized by other entity than an actor. This direct way is not correct, because the subject of the sentence can differ from an actor in case that a parser marks the predicate of the subject as a noun (details in section 3.6.). A parser's error usually completely breaks the structure of a sentence (especially if the sentence is long or describes more actions). Thus, we do not identify a communication outside a SuD scope.

3.5. Parse Tree Selection

Processing of the LarMenRup use cases shows that if one parser returns an incorrect parse tree of a sentence, another parser can often return a correct parse tree and vice versa. We employed the three state-of-the-art statistical parsers so we acquire three parse trees for every use case sentence. The issue is how to select the best parse tree, i.e. the parse tree which is correct or which is at least the closest to the correct parse tree.

This section describes our solution of the first goal; it describes how to select the best parse tree from a set of parse trees of a single use case sentence and it explains how we can understand that one parse tree is closer to the correct parse tree than another one. Our goal is to propose a solution which should work practically (We tested it on the LarMenRup use cases).

3.5.1. Algorithm Overview

In this section, we describe how to select the best parse tree of a use case sentence. First, we acquire an action set for every parse tree of the use case step by the algorithm as described in Sect. 3.4. Secondly, we compute the score for every action set by the algorithm proposed in Sect. 3.5.2. Finally, we select the action set with the highest score. If more action sets have the same (highest) score, we choose the arbitrary one. The best parse tree is the tree whose action set has been selected.

3.5.2. Computing Action Set Score

In this section, we describe how to calculate a score of an action set. The score of the action set is a sum of scores of the single actions included in the set. The score of a single action depends on its type and its principal attributes.

The score of an **unknown** action is -1000; -100 is assigned to an **illegal** action, 1 to a **condition** action, 5 to a **terminate** action, and 6 to a **goto** action. We define the score of direct objects assigned to a single action (*direct-objects-score*) as direct objects' words count + count of direct objects. The score of an **internal** action is 3 + *direct-objects-score*, the score of a **send** action is 4 + *direct-objects-score*, and the score of a **receive** action is 3 + *direct-objects-score* + an indirect objects' words count.

```
UC1-1.  score: 6, actions: [?CUS.arrivePoCheckout]
UC1-2.  score: 6, actions: [?CAS.startNewSale]
UC1-3.  score: 6, actions: [?CAS.enterItemIdentifier]
UC1-4.  score: 17, actions: [#recordSaleLineItem, #presentItemDescription_Price_Total]
UC1-5.  score: 5, actions: [#presentTax]
UC1-6.  score: 10, actions: [?CAS.tellTotal, ?CAS.askPayment]
UC1-7.  score: 8, actions: [?CUS.pay, #handlePayment]
UC1-8.  score: 25, actions: [#logSale, !AS.sendSalePaymentInformation, !IS.sendSalePaymentInformation]
UC1-9.  score: 5, actions: [#presentReceipt]
UC1-10.         score: 6, actions: [?CUS.leaveReceiptGoods]
UC1-*a1         score: 13, actions: [?CAS.restartSystem, ?CAS.log, ?CAS.requestRecovery]
UC1-*a2         score: 6, actions: [#reconstructPriorState]
UC1-*a2a1       score: 18, actions: [!CAS.signalError, #recordError, #enterCleanState]
UC1-*a2a2       score: 6, actions: [?CAS.startNewSale]
```

Figure 18. Actions generated from selected steps of use case UC1 borrowed from [9] and their scores.

Examples of action set scores are shown in fig. 18. To illustrate the computation we describe here the score computation of the step UC1-8 *([#logSale, !AS.sendSalePaymentInformation, !IS.sendSalePaymentInformation]*). The score of the action "*#logSale*" is 5; 3 (internal action) + 2 (*direct-objects-score*). The score of the action "*!AS.sendSalePaymentInformation*" is 10; 4 (receive action) + 2 (*Accounting System's* words count) + 4 (*direct-objects-score*). The score of "*!IS.sendSalePaymentInformation*"is 10; the computation is the same as in the previous case. The score of the whole action set is 25 (a sum of the actions' scores).

The reasons why we compute the score in the described way will be given in the following section.

3.5.3. Quality of the Proposed Metric

The proposed metric is based on heuristics and observations done while analyzing the LarMenRup use cases. The tests show that the proposed function works fine (see Chap. 5), although it does not always find the best solution (this is very rare).

Linguistic parsers fail mainly on long and complex/compound sentences. If the sentence is parsed badly, we usually acquire an incomplete action set or even no action is recognized. More complex sentences usually describe multiple or more complex actions. Thus, if we get multiple or more complex actions from one parse tree than from the other, we can suppose that the first parse tree is better than the second one. The exceptions to this rule are likely to happen but they are highly improbable because it would mean that we acquire more complex action set from the incorrect parse tree than from the correct one. The previous reasons explain why we construct a metric which evaluates the action set obtained for a parse tree instead of trying to evaluate the parse tree directly.

The metric assigns the whole number (score) to every acquired action set. A score expresses how many actions the action set contains and how much the actions are complex. If the score is positive, we obtain a meaningful action set from a parse tree so the action set can be used for further analysis. The negative score means that we do not understand the parse tree (or some of its subtrees) or that we have recognized some violation of our premises (sect. 3.4.8).

We assigned a very high negative number (-1000) to an **unknown action** because if the action set contains an unknown action, i.e. we do not understand the whole or a part of the parse tree, the use case step is not recognized so its score must be lower than a score of any parse tree which we fully understand.

If the action set contains an **illegal action**, i.e. we have recognized our premises violation, the score must be negative, but this parse tree must be preferred to unknown parse trees in order to to be able to notify a user about the violated rules. Thus, we assigned -100 to the illegal action. Notice that a score such as -95 is a legal score (e.g., "*User is notified about the error and use case aborts.*" has such a score). The reason is that we process all sub-sentences and sub-phrases even in case that one of them is unknown or illegal (details are being discussed in Sect. 3.4.8).

Other types of actions are assigned with positive values which reflect the complexity of those actions. We assign fixed values (5, 6) to the special actions (**terminate** and **goto**). The algorithm for identifying special actions is very simple so it fails rarely and it's not necessary to use some more sophisticated evaluation. The value of an **operation request** and **internal** action's score consists of a predefined constant and direct and indirect objects' scores. The indirect object score is the indirect object's word count; a longer indirect object implies a higher score. The direct object score is a sum of the direct objects' word count and direct objects count; actions which contain longer and more direct objects than the others are being preferred.

3.6. Known Limitations

We detect some situations which cause failures of our algorithm and we do not know how to solve them. On the other hand, these situations are rare and the process of use case to pro-case conversion cannot be still fully automated so this does not question our algorithm. We divide limitations into two groups; the limitations caused by weaknesses of the used linguistic tools and the limitations caused by weaknesses of our algorithm.

3.6.1. Limitations Caused by Linguistic Tools

The detection of multiple direct/indirect objects can fail if any of the objects contain a preposition phrase (("PP") node). The preposition phrase causes that the object's noun phrase structure does not match our expectations and the algorithm finds only some objects of the multiple object (usually only the first one).

The other limitation is caused by phrasal verbs. The linguistic parsers divide the root of the verb and the preposition. That is why we create actions like *#lookPhoneNumber* instead of *#lookUpPhoneNumber* which would sound better. We cannot do anything about it because we cannot distinguish a preposition which is a part of a phrasal verb and a preposition which is colocated with a verb.

Another limitation is using the lemmatized form of words. We can sometimes get an incorrect lemma of the word (e.g., data can be lemmatized as both datum or data). In some special cases, the usage of the original word can be even better than the usage of the lemma, regardless we use the lemma. E.g., the abbreviations like POS [9] ending with "s" are lemmatized by cutting off the ending "s".

The next limitation in this group is that some words like "*request*" can be both verbs and nouns. The statistical parsers sometimes tag them with an incorrect tag. This usually disturbs a parse tree of the sentence. Because all our parsers are trained on the same data [2, 4, 6], they often make the same mistake in this case. We tried to detect and repair irregularities in the parse trees caused by this error but our attempts have had no success in case that the sentence structure was more complex than in a simple sentence like "User requests data.".

The last limitation in this group is caused by coordinating conjunctions like "*and*". The linguistic parser cannot sometimes distinguish multiple objects and multiple modifiers. E.g., "*price, billing and contact information*" is parsed as if "*price*" is a modifier of "*information*". This causes that we detect multiple direct objects incorrectly.

3.6.2. Limitations Caused by Our Approach

Use case writers do not sometimes write the exact name of actors. E.g., the term "*system*" is used instead of "*computer system*" (in case that the system does not represent the SuD but a secondary actor). Another problem is faced as for multi word actors which are not written in the correct way, e.g., "*Charge Validation System*" is used in the form of "*System for charge validation*" etc. To avoid problems with use case ambiguity, the recommendations for use case writers say that the names of actors should be strict so we have decided not to try solving this limitation. This caused troubles with processing of some of the LarMenRup use cases.

Another and even the most serious limitation is the usage of secondary actors in a different role in a use case sentence than as an indirect object. The secondary actor is sometimes used in a role of a direct object (e.g., "*System validates user.*"). We see no way to solve this issue yet because we determine a noun phrase as an indirect object if it matches an actor's name. We do not employ linguistic tools for this task because of reasons discussed in Sect. 6.3.

In some rare cases, a use case sentence contains a direct object (or objects) which belongs to more than one verb. E.g. in the sentence "*System authenticates and validates user*", the direct object "*user*" is bound with the verbs "*authenticate*" and "*validate*". In our algorithm, we ignore such direct objects because they are not part of verb sub-phrases which we use to find direct objects (In our example, "*user*" is not in verb phrase directly containing the verbs "*authenticate*" and "*validate*"). This limitation is not serious because this case is very rare and it does not influence our ability of action type recognizing. Moreover, we can miss only direct objects because indirect objects are tied to just one verb.

The last limitation is the communication outside SuD scope. We can detect communication between actors but we cannot recognize internal actions of actors and we cannot detect communication between an actor and other entities which are not actors from the SuD's point of view. In this case, a use case step violates the premise 3 but our algorithm cannot detect it.

We can see also some insignificant problems like not being able to distinguish between conjunctions like “and” and “or” etc. but consideration of all details of a natural language, such as the mentioned ones are outside the scope of this work.

3.7. Use Case Step Analysis Summary

We have introduced a new algorithm (based on [14]) for acquirement of actions and their principal attributes from textual use cases. The algorithm enables processing of larger number of use cases than the original one because we have weakened the premises on a use case step proposed in [14]; we allow a use case step to describe more actions, to contain multiple direct and indirect objects, to start with conditions, etc.

We have proposed a metric which evaluates a quality of a parse tree using the information about the actions which can be acquired from the parse tree by our algorithm. This enables us to employ more than one parser and obtain a multiple parse tree for every use case step so we minimize the risk that our analysis returns incorrect actions because of an incorrect parse tree returned by a single parser.

Finally, our algorithm recognizes some kinds of errors done by use case writers which lead to failure of our analysis so we can notify the user about such errors.

Chapter 4

Case Study

We have tested the algorithm proposed in the previous chapter on data borrowed from [9, 14, 21]. In this chapter, we will show how the algorithm processes the use case UC1 (Process Sale) borrowed from [9] to illustrate how the algorithm really works and to demonstrate some of the limitations of our algorithm. Because of the length of this thesis, we do not show processing of the whole use case but only some selected steps.

The primary actor of the processed use case is the "*Cashier*". In generated actions, we mark him/her as *CAS*. The secondary actors are: "*Accounting system*" (*AS*) and "*Inventory system*" (*IS*). The entity "*Customer*" (*CUS*) is not considered as an actor in [9] because the *Customer* is out of the scope of the use case UC1 but we consider him/her as a secondary actor when processing the use case, because he or she interacts with the SuD in step 7b1 ("*Customer enters their credit account information.*").

Consideration of *Customer* as an actor also helps us to uncover the illegal steps where *Cashier* and *Customer* interact together because we recognize it as an actor to actor communication. If *Customer* is not an actor, a communication of *Cashier* to *Customer* is not detected (sect. 3.6).

When describing a parse tree or subtree in the text, we use a lisp like notation because it is widely used by linguists for tree description.

4.1. Simple Step

In this section, we will show how the algorithm processes a very simple use case step - step 3 ("*Cashier enters item identifier.*").The parse tree of this step is in Fig. 19. We will show only one parse tree because all parse trees of this sentence generated by the parsers are equal. Thus, we also do not show how our algorithm selects the best action set because all action sets are equal.

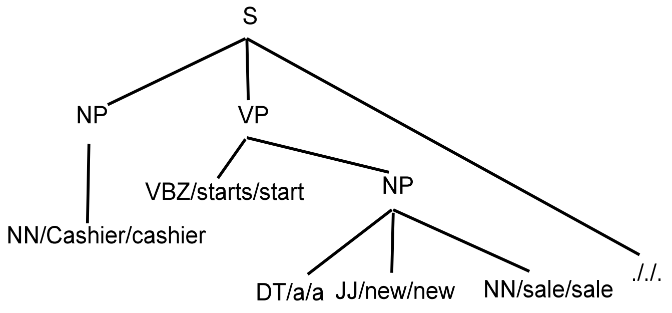

Figure 19. Parse tree of a "*Cashier starts a new sale*" (step 2 of use case UC1 in [9]).
This figure shows a parse tree of a simple use case sentence.

First, the algorithm checks whether the root node of the tree is "S". If it is, we try to find all conditions and then it identifies all plain sentences. The parse tree contains no "SBAR"node as a child so no condition is found and it also has no child "S" node so the whole tree is selected as a plain sentence.

The subject is found as the first noun phrase of the sentence which contains some noun node. In this case the subject's subtree is (NP (NN/Cashier/cashier)), thus the subject is identified as "*Cashier*". Cashier is a primary actor of the use case so the processed plain sentence (the whole step in this case) describes one or more *receive* actions.

The main verb phrase contains the "VB*" node, so the analysis decides, that the plain sentence contains only one predicate and the candidate for predicate is the lemma of the word represented by the "VB*" node; in our case the candidate for a predicate is *"enter"*. The candidate for a predicate (*"enter"*) is not either a padding verb, an auxiliary verb, nor a modal verb, so no other candidate verb is searched and "enter" is determined as a sentence predicate.

The predicate verb phrase contains only one child noun phrase (NP (NN/item/item) (NN/identifier/identifier)). The noun phrase does not match the supporting actor so it is selected as the first direct object ("*item identifier"*). No other direct object is found (the parent node of the fist direct objects' noun phrase does not contain any other noun phrase as a child) so the first direct object is the only one direct object.

Because the use case step contains no more plain sentences, the analysis terminates and the following action set is assigned to the use case step: [?CAS.enterItemIdentifier]. The score of this action set is 6 but it is not important in this simple case because processing of all parse trees returns the same action sets.

4.2. Internal Action of an Actor

In this section, we will show how our algorithm returns an incorrect action. We demonstrate it on the step 1 *“Customer arrives at POS checkout with goods and/or services to purchase.”* (fig. 20).

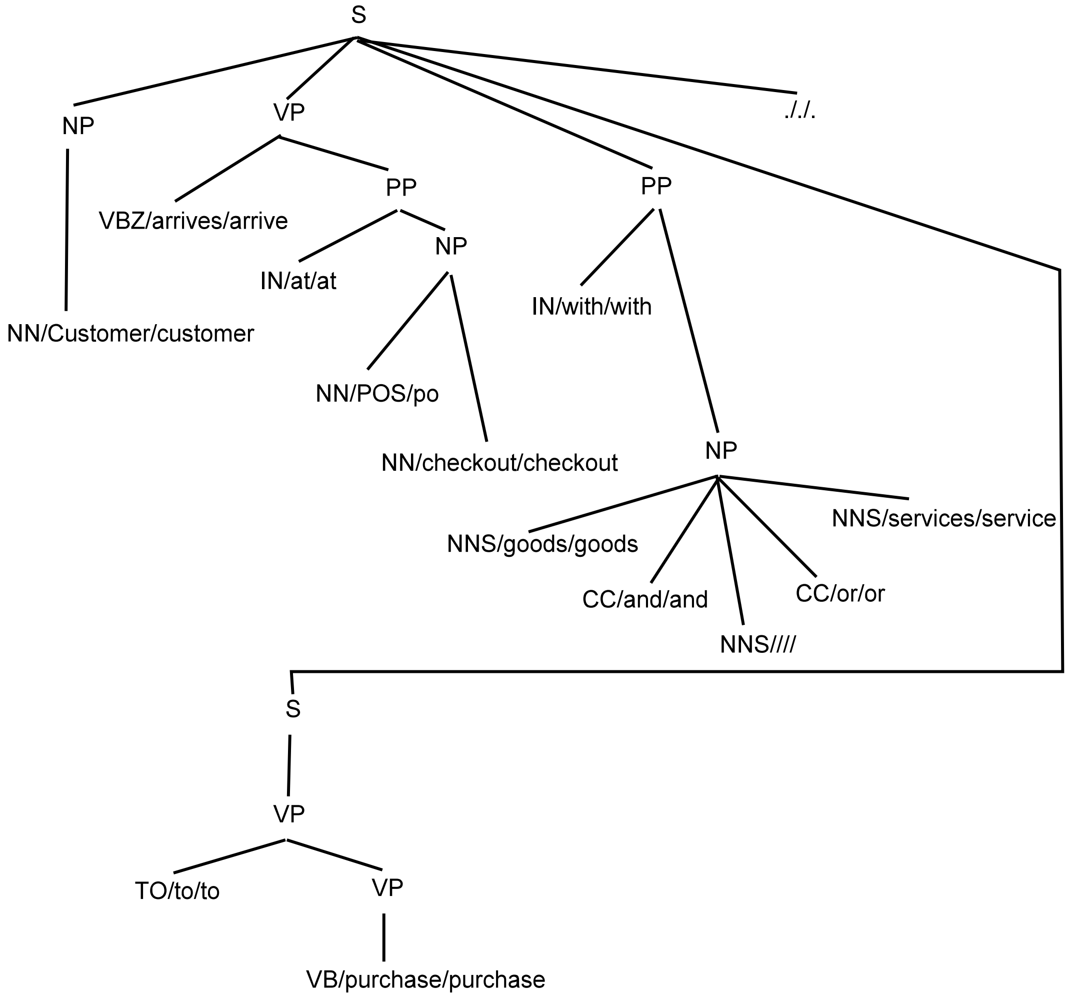

Figure 20. Parse tree of *“Customer arrives at POS checkout with goods and/or services to purchase.”* (step 1 of use case UC1 in [9]).
This figure shows a parse tree of a use case step describing an internal action of an actor *“Customer”*.

The actor “Customer” is a subject of the sentence so the algorithm supposes that the step describes a *send* action. The predicate of the sentence is the verb “arrive” and “POS checkout” is intended to be a direct object. The algorithm constructs the following action set:

[?CUS.arrivePoCheckout]. Here can be seen the other limitation of our approach (sect. 3.6) - a shortcut POS (Point of Sale) is lemmatized as "po" which is obviously incorrect.

This step describes an internal action of the actor "*Customer*" but our analysis evaluates the step as a *send* action because we do not presume that a use case step can describe internal actions of actors. We do not try to recognize such actions, because a sentence structure of the actor's internal actions does not differ from a sentence structure of *send* actions (compare Fig. 20 and Fig. 3).

4.3. Compound Sentence

This section describes how we analyze the compound sentence - step 7 ("*Customer pays and System handles payment.*"). The parse tree of this step is in Fig. 8.

The algorithm tries to find the conditions but there the sentence does not contain any of them. Then it tries to identify all compound sentences. The root node contains two child "S" nodes so the algorithm recognizes that the sentence is compound and identifies all child "S" node sub-trees as the parse trees of the plain sentences describing the independent actions. The identified plain sentences are processed as illustrated in Sect. 4.1. and the following action set is acquired: [?CUS.pay, #handlePayment] and its score is 9 but it is not so important because all obtained parse trees of the step are similar.

Some interesting thing can be seen there. The first plain sentence is evaluated like describing a *send* action and the second sentence an *internal* action. From the human's point of view, we would assume that a *Customer* pays to a *Cashier* and *Cashier* asks the system to handle the payment so it seems that our algorithm fails because the desired actions should be: an *illegal* action - "Actor to actor communication" and ?CAS.handlePayment. We suppose that this is a vagueness from the use case writer which does not matter if the use cases are processed by a human but it leads to errors during machine processing. In this case, we consider the returned action set as correct, because it describes exactly what is written in the use case step.

4.4. Compound Sentence with Common Subject and Multiple Indirect Object

This section describes our analysis of the compound sentence with a common subject and with a multiple indirect object. The analyzed step is the step 8 ("*System logs completed sale and sends* *sale and payment information to the external Accounting and Inventory system.*"). Two different parse trees generated by two different parsers are shown in Fig. 13 (tree 1) and Fig 21. (tree 2).

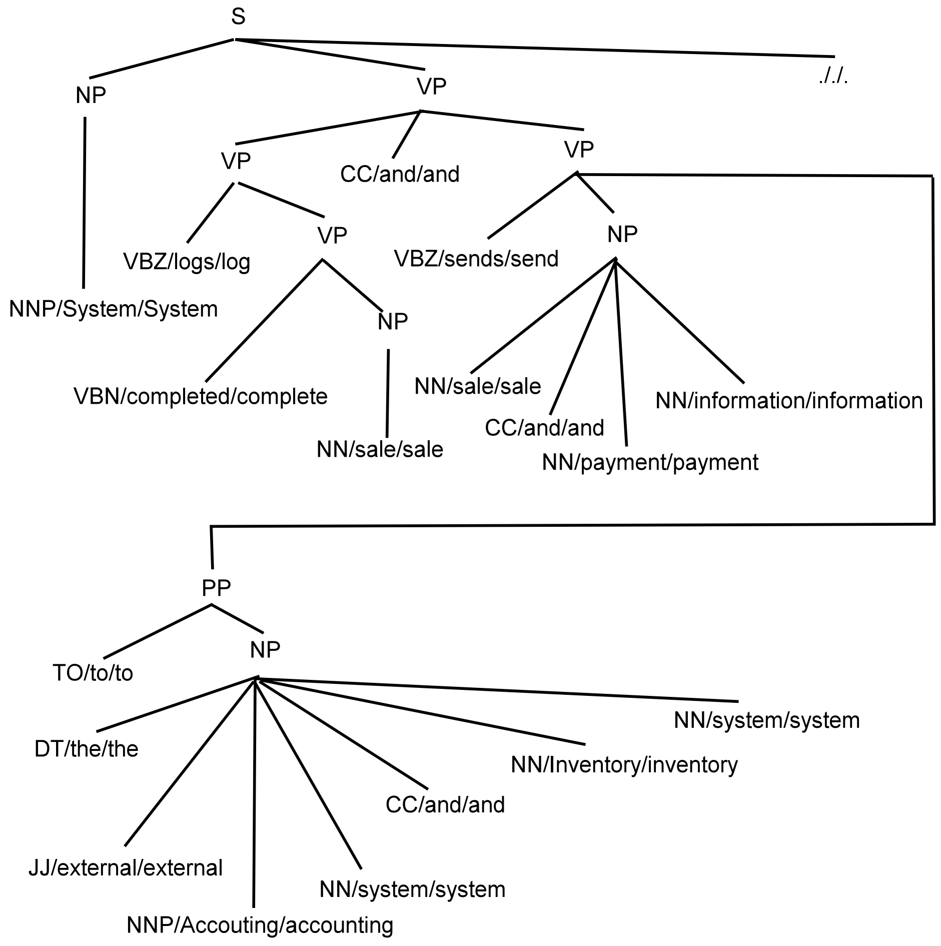

Figure 21. Parse tree of "System logs completed sale and sends sale and payment information to the external Accounting system and Inventory system." (step 8 of use case UC1 in [9]). This figure shows an incorrect parse tree of a sentence containing multiple indirect object (*Accounting system and Inventory system*).

The shown parse trees are very similar. They differ only in noun phrases representing indirect objects so we do not distinguish between described parse trees unless it is important for understanding.

First, the analysis recognizes a subject "System" so the use case step describes one or more *send* or *internal* actions. The further analysis discovers that the root verb phrase does not contain either ("VB*"), ("MD"), or ("AUX") sub-tree but it contains two verb sub-phrases. Each verb sub-phrase describes either one or more actions. The analysis of the first verb phrase obtains a predicate in a similar way as illustrated in Sect. 4.1. and the result of processing of the first verb phrase is an action "?CUS.pay".

The analysis of the second verb phrase seems to be more interesting. First the algorithm tries to identify all indirect objects. It selects all noun phrases which do not contain any noun sub-phrases and matches them with the list of actors. The first noun phrase which matches an actor's name is selected as the first indirect objects. In case of the tree1, the first indirect object is (NX (JJ/external/external) (NNP/Accounting/accounting) (NN/system/system)) and in the case of the tree 2 it is (NP (DT/the/the) (JJ/external/external) (NN/Accounting/accounting) (NN/system/system) (CC/and/and) (JJ/Inventory/inventory) (NN/system/system)).

After the first indirect object has been found, the algorithm tries to find another one. In the tree1, it finds the second indirect object (NX (NN/Inventory/inventory) (NN/system/system)). In the tree 2, no other indirect object has been found.

The direct object of the second verb phrase is identified as described in Sect. 3.4.6 and it is a noun "SalePaymentInformation". We find interesting, that although the analysis of the second verb phrase of the tree 1 has identified two indirect objects and the analysis of the tree 2 only one, the actions created for both the trees are the same (!AS.sendSalePaymentInformation, !IS.sendSalePaymentInformation). The reason is, that we search all actors who can be matched with the acquired noun phrase and our matching algorithm requires to an actor's name be a substring of the identified noun phrase. This causes that one noun phrase can be matched with two or more actors.

4.5. Sentence with Multiple Direct Object

This section shows processing of a sentence which contains more than one direct object. It also demonstrates how the proposed metric helps us to select the best action set. We parse the step 9a1 ("*System presents the rebate forms and rebate receipts for each item with the rebate.*"). The two different parse trees are shown in figure Fig. 22 (tree 1) and Fig. 23 (tree 2).

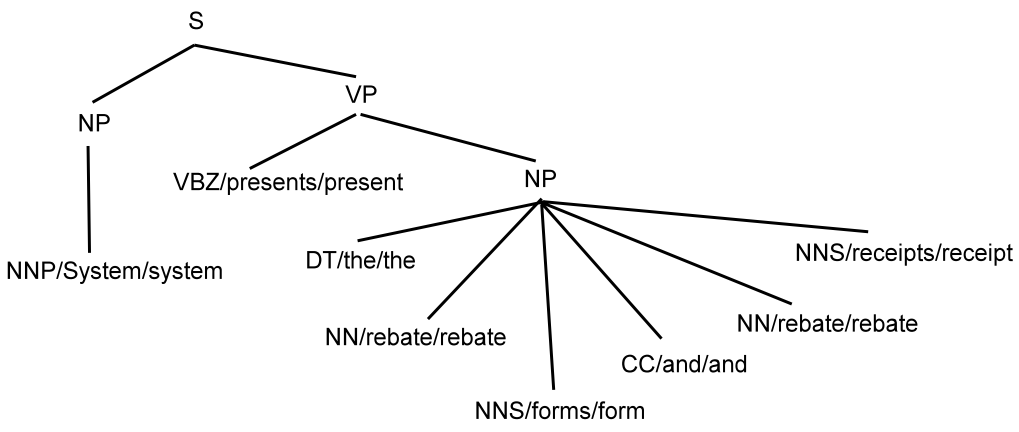

Figure 22. Fragment of parse tree of "*System presents the rebate forms and rebate receipts for each item with the rebate.*" (step 9a1) of use case UC1 in [9].
This figure shows a fragment of an incorrect parse tree of a sentence with multiple direct object.

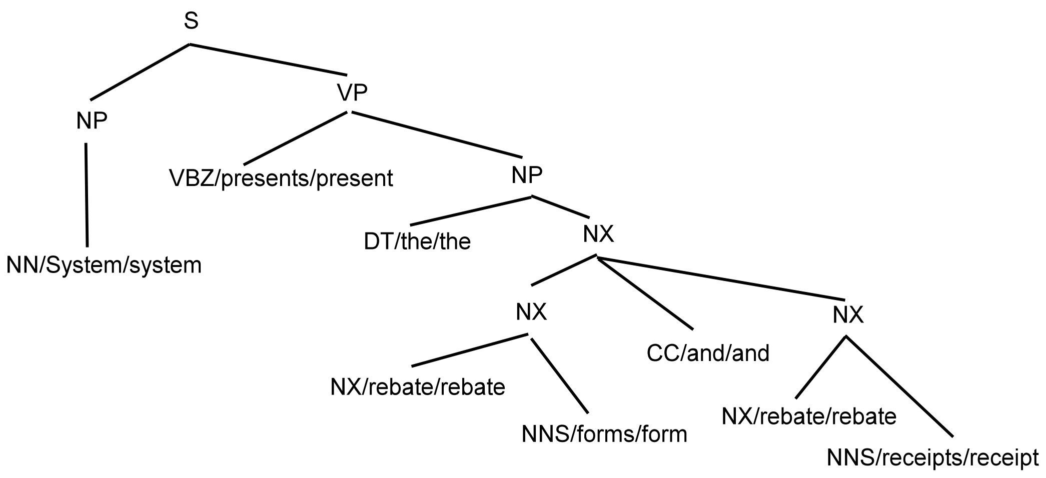

Figure 23. Fragment of parse tree of "*System presents the rebate forms and rebate receipts for each item with the rebate.*" (step 9a1) of use case UC1 in [9].
This figure shows a fragment of a correct parse tree of a sentence with multiple direct object.

The analysis identifies a subject of the sentence ("System") and a predicate ("presents") as it was illustrated in the previous sections. The first identified direct object of the tree1 is a noun phrase containing "the rebate forms and rebate receipts". The first direct objects of the tree 2 is a noun phrase containing only "rebate forms". No other direct object is found in the tree 1 but in the tree 2, the second direct object (a noun phrase containing "rebate receipts") is found.

The parse tree 1 produces an action set [#presentRebateFormRebateReceipt] with one action which contains just one but a longer direct object. The parse tree 2 produces an action set with only one action too ([#presentRebateForm_RebateReceipt]) but the action contains 2 direct objects. The score of the first action set is 10 but the score of the second action set is 11 so the correct action set is selected because the direct object's count is a part of the actions' scores.

4.6. Proof-of-the-concept Implementation

We support this thesis with a proof-of-the-concept implementation (prototype) of the proposed algorithm. The prototype, the LarMenRup data (except data borrowed from [21]), and output of the prototype are available on attached CD. The guidepost of the CD and installation instructions are described in file *readme.html* placed in a CD's root.

For better orientation in a source code, we describe here the main classes of the prototype. The launching class is called `thesis.Main`. The class implementing the algorithm described in Sect. 3.4 is called `thesis.procasor.Procasor`. Classes representing actions are placed in the package `thesis.procasor.actions`. The action classes contain method `getScore()` which computes the score as defined in Sect 3.5.

We have used XPath (query language for XML) to search the requested nodes in parse trees. This approach is good for the prototype because it makes the implementation more comprehensible but it can cause performance problems in a core implementation (the most serious performance problems are with creating XML representation of the parse tree).

Chapter 5

Evaluation

In this section, we define the criteria for evaluation of our algorithm and for comparison of our algorithm with the original algorithm proposed in [14]. Consecutively, we compare results yielded by our algorithm with the results of the original algorithm. We have tested the algorithms with testing data used in [14]. The testing data are published in [19] and generated actions have been attached in the appendix B.2. Then we evaluate our algorithm on testing data borrowed from [9, 21] and finally we conclude the evaluation. The term "original algorithm" means the algorithm described in [14].

The results of our algorithm are created by the prototype we have written. The results of the original algorithm are generated by the prototype which was implemented by Vladimir Mencl to support his ideas in [14].

5.1. Evaluation Criteria

We evaluate the algorithm by analysis of action sets created by the evaluated algorithm for the given use case steps. The main criterion of evaluation is the number of correctly generated action sets. Our definition of a correct action set is subjective, i.e. we do not define the objective criteria but we just compare a use case step and the generated actions and if we think that the action set fits the use case step, we say that the algorithm succeeds and that the action set is correct. The evaluation fails if it does not succeed.

Although, our evaluation of an action set is subjective, we keep the following rules during the analyses: We say that an algorithm fails if it returns an unknown action or if the recognized action(s) is (are) incorrect. If a use case step describes more than one action, the algorithm succeeds if and only if it recognizes all actions (i.e. the algorithm proposed in [14] always fails on such use case sentences). If the sentence violates the premises defined in this work (Sect. 3.1), the algorithm succeeds if and only if it recognizes such violation. The exception to this rule is made if passive voice is used in steps describing special actions. In this case we tolerate that the original algorithm recognizes the special action instead of notifying the user about the usage of passive voice.

Moreover when we compare the new and the original algorithm, we say that the result is more precise if it matches our expectations much better; e.g., if the sentence's direct object is an "item description", the original algorithm creates the action token "*respondSystem*", but our algorithm creates "*respondSystemResponse*". The second token describes data being passed through much better. On the other hand if the direct object is "whole offer" we mark both action tokens (*validateOffer*, *validateWholeOffer*) as correct and we do not distinguish which of them is a better one.

5.2. Original and New Algorithms Comparison

For every use case step we have compared the action sets generated by both algorithms and we have obtained results presented in the following table.

Criteria	**Results**
Both algorithms fail	4
Our algorithm succeeds but original algorithm fails	18
Our algorithm fails but the original succeeds	3
Both algorithms succeed but our algorithm returns more precise result	4
Both algorithms succeed but original algorithm returns more precise result	0

The results of our test (although the evaluation is subjective) show that the new algorithm yields results much better than the original algorithm.

To clarify the point 3, the original algorithm acquires a better action set in case that all linguistic parsers do not mark the predicate of the sentence as a verb but as a noun (sect. 3.4.6). Even in this case, a headword of the predicate verb phrase is the correct predicate and the original algorithm succeeds because it uses the headword information but our algorithm does not (sect. 3.2).

We compare the algorithms only on the data used in [14], because the testing data borrowed from [9, 21] are not written to match the Mencl's premises. It implies that the original algorithm cannot process them correctly and the result of such comparison would be useless if we consider the sentence beginning with condition or describing multiple action; the original algorithm cannot return the correct action set for such a sentence.

5.3. Processing of Other Testing Data

We evaluate our algorithm on data borrowed from [9, 21]. We use the criteria proposed in Sect. 5.1. We want to emphasize that if the algorithm detects an illegal action properly, we consider it as a correct output, so the presented results cannot be interpreted as we can transform almost all the use case steps.

When the algorithm returns an incorrect output, we distinguish two cases:

1. The algorithm fails because of undetectable premise violation.
2. The algorithm fails because of limitations of our algorithm (details discussed in Sect. 3.6).

The processing of a use case borrowed from [9] returns 7 errors of the first type and 4 errors of the second type. The use case contains 53 steps. The main problems are caused by the usage of a secondary actor in the use case sentence in a different role than in an indirect object (sect. 3.6).

Processing of the use cases borrowed from [21] returns 4 errors of the first type and 10 of the second type. The use case set from [21] contains 113 use case steps. The main causes of problems are the usage of secondary actors as described above, incorrect tagging of words which can be both a verb or a noun, and not keeping exact names of secondary actors (these problems are being discussed in Sect. 3.6).

5.4. Summary

In this section, we summarize the results of our work. Our first goal was to propose a metric which helps to evaluate the quality of a parse tree. The solution of this task is proposed in

Sect. 3.5 where we do not only define the metric but we integrate it into our algorithm of use case to pro-case conversion. The results obtained on testing data shows that the metric identifies the best part trees correctly in the most cases.

Our second goal was to propose an algorithm which would be better than the algorithm proposed in [14]. We think that we will fulfill the task because our algorithm (sect. 3.4) accepts more types of use cases than the previous one and it identifies direct objects more precisely. This is supported by results of processing testing data discussed earlier in this chapter.

If we consider use cases borrowed from [21] as the example of industrial use cases, we can say that we succeed and although the original algorithm cannot process many steps in such use cases, our advanced algorithm correctly processes most of them. This is some progress but still we must consider that we cannot process all steps because of the limitations of the algorithm and also because the use case writers sometimes break our premises. The requirements specifications (including use cases) are often incomplete and use case specifications do not contain all the information about the SuD so the full automation (without human participation) of the use cases to pro-cases conversion has not been possible nowadays. However we believe that our algorithm can be used in some interactive tools for the use case to pro-case conversion.

Chapter 6

Alternative Approaches

In this chapter we discuss alternative ways that we could use for the use case analysis. We have not tried these methods in practice, we have used only the information from papers. Trying of these methods seems to be a possible topic for our future work.

6.1. Parser Combining

We have decided to create more parse trees for every use case step to avoid failures caused by an incorrect parse tree returned by a single parser. When we have more than one parse tree of a use case sentence, we deal with a problem how to select the best one.

John C. Henderson and Eric Brill [8] solve the problem how to produce more accurate parse tree of a sentence if they have more than one parse tree of the sentence. They use only syntactical information presented by the parse trees. They even propose methods how to create a new parse tree which should be better than any of the original ones.

We try to describe the main ideas of their approach. The method which tries to create the new parse tree from the original trees which should be better than the originals is called parse hybridization. We describe only the easier variant called Constituent Voting. They first divide the trees to tuples (label, start, end) where every tuple is related to some tree node. The tuples are called constituents. Then they construct the new set of constituents from which to construct the new parse tree. The constituent is included in the new constituent set if it appears in the output of majority of the parsers. Finally they construct the new parse tree.

The second algorithm which tries to select the best parse tree is called parser switching. We describe only the easier variant which is called Similarity Switching. In this case, they construct the constituent set for every parse tree in the same way as proposed above. Then they count the score for every constituent set which measures a similarity between parses by counting the constituents they have in common. They pick the parse with the highest score. This parse is the most similar to the other parses.

The precision in case that the parser hybridization is used is higher than if the parser switching is used but the usage of parser hybridization has one big disadvantage. The disadvantage is that this approach can return the parse tree which does not match the grammar of the correct parse trees. Thus this method cannot be used in situations that a receiver of the parse tree expects a correct parse tree (e.g., the algorithm proposed in this thesis).

We do not use their approach because we expect a (grammatically) correct parse tree so we could use only methods of parser switching. Our method of selecting the best parse tree uses the added information that the parsed sentence is a use case sentence but the parser switching does not so we suppose that our approach is better in our case but we have not tested it so we cannot be sure.

6.2. Extracting Syntactic Relations Using Heuristics

The structure of the plain use case sentence should mach the simple patterns (see premise 3). We are not interested in the whole parse tree of the sentence, we only need to obtain a subject, a verb, and direct and indirect objects of the sentence. That is why we consider to parse the sentence by ourselves using some simple rules.

Although a use case sentence structure is simple, we think that it is impossible to define the rules which would help to obtain all objects of the sentence especially if we allow to a use case sentence to be a compound or a complex sentence or to contain a multiple verb. This approach would be possible if we could use some valence dictionary (the dictionary which describes collocations of the verbs), but such dictionaries are hard to obtain and some verbs allow more possibilities so they would not be easy to use.

The attempt of extracting syntactic relations is described in [24]. They are not able to obtain the whole parse tree but they provide only the predefined relations. The most important relations from our point of view are relations: subject-verb, object-verb and indirect object- verb. The heuristics for acquirement of these relations are: Subject is the first head noun of a noun phrase chunk to the left of the verb, the object the first to the right and the indirect object the second to the right.

Their work shows that, although with the simple rules, interesting results can be yielded (accuracy about 50%) but their approach is not possible for us because of the low accuracy. The statistical parsers' accuracy on English text is about 90% that is much more than 50% yielded by their method.

6.3. Extracting Syntactic Relations Using State-Of-The-Art Linguistic Tools

We are not only the ones who need to obtain a subject, predicate, direct and indirect objects of the English sentence. This issue is solved in the area of machine translation by linguists. In their case, they are interested not only in subject, predicate, etc. but they focus on the grammar and functional dependencies in general. The possible solutions of this issue are proposed in [3, 10, 23].

The solution proposed in [23] is based on combination of statistical approach and rule-based approach. They created a parser implementing their approach. The most relevant dependencies detected by the parser are: subject-verb, verb-first object, and verb-second object. The parser processes 100 words per second on the common PC. It shows that parser's performance is state-of-the-art. The precision of identifying subjects or objects is about 90%. The recall (portion of the treebank constituents that are hypothesized) is about 80%.

Another solution is proposed in [3]. This work is focused on extracting the functional tags from the Penn Tree Bank [1] in an automated way and further using this information in natural language processing. Functional tags are tags which extend the low level tags as noun phrase (NN), verb phrase (VP) etc. and refine on their meaning. The relevant tags for us are mainly tags for a subject (SBJ) and a dative (DAT) but also other tags could be used to improve natural language use cases processing.

The last discussed way to acquire the grammatical dependencies is to use the Minipar parser [10]. This parser is very efficient (It parses about 300 words per second on an average computer from 1998) and it achieves about 88% accuracy and 80% recall. Moreover the parser can be installed on both Windows and Linux and even Solaris. The most important grammatical tags provided by Minipar are a subject of the verbs, an object of the verbs, and a second object of the distransitive verbs.

The problem why we do not employ these tools and proposed approaches is the low standardization of the outputs generated by the parsers and the low standardization of provided information in general. Furthermore we have tried to parse some simple use case sentences by the Minipar [10] parser and the provided parse trees were incorrect but the parsers we have used [2, 4, 6] provided the correct parse trees. Employment of these tools is a possible topic for our future work.

6.4. Controlled languages

The different approach to processing requirements written in natural language in an automated way is the restriction of grammar of the natural language. The restricted language is called controlled language (CL). The processing of the CL is much easier than processing of the natural language. This approach to requirements processing is proposed in [17].

The key problem of the CL is that it can restrict the language too much so some facts cannot be expressed by the CL or their expression can be too difficult. The other problem is that the user of the CL has to learn it first.

We do not use this approach because we want to process the industry use cases which must be comprehensible for analysts as well as for users so it would not be easy to define such language and it would make an expansion of our algorithm more difficult. Furthermore the existing use cases are not written in controlled language so we would not be able to process them.

Chapter 7

Related Work

7.1. Application of Linguistic Techniques for Use Case Analysis

In this section, we discuss work [7]. It is not focused on acquirement of behavior of the system from use cases as our work but it uses similar linguistic methods to evaluate the quality of use cases.

According to [7], the use case formalism is an effective way of capturing both business processes and functional requirements in a very simple and easy-to-learn way. System behavior can be effectively specified by structured Natural Language (NL) sentences but the usage of NL is a critical point, due to the inherent ambiguity originating from different interpretations of natural language descriptions.

In [7], the use of methods, based on a linguistic approach, is being discussed. The aim is to collect quality (of use cases) metrics and detect defects related to such inherent ambiguity.

The detected defects of textual use cases are grouped into three main categories:

- *Expressiveness* category: it includes characteristics dealing with the understanding of use cases by humans.
- *Consistency* category: it includes characteristics dealing with the presence of semantics contradictions and structural incongruities in use cases
- *Completeness* category: it includes characteristics dealing with the lack of necessary parts within the requirements specifications.

The metrics proposed in [7] help with identifying defects from the mentioned categories and evaluate the quality of use cases. The metrics are based on indicators that can be acquired from use cases using linguistic tools. Some of the possible indicators are usage of imperative (shall, must ...), usage of directives (e.g., note,...), usage of options (can, may,...), etc. in the use case steps. The examples of proposed metrics are “Average number of words per sentence” or “Optionality” - the ratio of number of sentences containing the option indicator and all sentences count.

The work [7] employed linguistic tools to process textual use cases as our work but both works use acquired information for something else. We use the acquired data to obtain behavior specification of the SuD but the authors of [7] use it to measure the quality of use cases. To avoid confusion, the metric proposed in this thesis is totally different from the metrics proposed in [7].

7.2. Autonomous Requirements Specification Processing Using Natural Language Processing

The work [13] is focused on extracting information, which can be useful for system analysts or software engineers, from the requirements specifications using linguistic tools. The requirement specification document does not have to be a use case but it can be any artifact written in natural language.

Many of the problems encountered in software systems can be tracked back to shortcomings in the processes and practices used to gather, specify and manage the end product requirements. The cost of correcting defects is often significantly greater than cost that would have been incurred to ensure that requirements correctly represented the users’ need.

Whilst the generation of a complete and non-ambiguous set of requirements reduces the risk in any given project, there is still a risk that the requirement set is not transformed into an appropriate design. The transformation of requirements into a design is usually done by software engineers and depends primarily on the software engineer’s knowledge. The aim of the discussed work is to extract information from the natural language requirements specifications and to provide it to a software engineer to uncover shortcomings in the requirements specification and to reduce the risk that the design does not meet the requirements.

The authors of [13] have introduced the interactive system that processes the requirement specification and provides information as mentioned above. The system consists of two subsystems - a parsing system and a term management system.

The term management system uses natural language processing to obtain parse trees of sentences of the requirement specification document. First, it identifies all noun phrases which contain no noun sub-phrase. Secondly, it extracts all noun terms from the identified noun phrases and provides them to the term management system.

The term management system is an interactive system which displays the extracted terms to the software engineer. The software engineer filters the terms and classifies the remaining terms as a function, an entity or an attribute. The system stores the classified terms in a knowledge base. Class conflicts can be identified by the system and the knowledge base can be used for automatic generation of relevant design artifacts such as object models, data models, etc.

Some open issues remains - if the linguistic parser provides a multiple parse tree, the system is unable to identify the best one so it selects the first one. The system is also unable to do compound noun analysis and proper noun processing.

To compare this work with ours, we focus on processing of use case specification but authors of [7] allow to process any requirement specification. Because a format of a general requirement specification is not so strict as the format of use cases, they cannot retrieve so precise information. Moreover we solve some of their opened issues such as identifying the best parse tree.

7.3. Natural Language Requirements Analysis and Class Model Generation Using UCDA

The work [11, 12] presents a methodology to automate natural language requirements' analysis and class model generation based on Rational Unified Process [21]. A CASE tool named Use-case driven Development Assistant (UCDA) is implemented to support this methodology. UCDA uses natural language processing to help with the use case and class model creation. The presented results seem very good except their work does not solve possible problems as if a parser returns an incorrect parse tree.

The use case specification is created from the stakeholder requests. The requests are parsed and use case names and actors are identified. The candidate actors are derived from nouns, especially those that are subjects of the statements. The candidate use cases are derived from verb phrases acting as actor's predicates.

The class model is derived from the use case specification. First, the use case steps are divided to simple statements according to simple rules which respect the use case step structure. The examples of such structure patterns are: *Basic* - the whole sentence is a simple statement or *if-then* - the if-clause can consist of one or more condition statements and the then-clause contains a flow of event statement. Secondly, a parse tree of every statement is acquired and the objects and messages (analogy to our actions) are obtained from the parse tree using one of the 17 defined rules. The rules represent the relationships between the statement structures (structure of the parse tree) and the behavior types (actors, and types of analytic classes based on Rational Unified Process [21] - entity, boundary, control).

This work is very similar to our work. The presented rules are close to our rules except if we do not distinguish types of analytical objects and their employed parser uses different format and provides slightly different information than our employed linguistic parsers. Moreover we partially solve problems with linguistic analysis which are not mentioned in their work such as if a single parser returns an incorrect parse tree.

Chapter 8

Conclusion and Future Work

Our first goal was to propose a metric evaluating the quality of a parse tree. The solution of this task is proposed in Sect. 3.5. where we not only define the metric but we integrate it into our algorithm of use case to pro-case conversion. The results obtained on testing data show (chap. 5) that the metric identifies the best part trees correctly in the most cases so we can avoid many violations caused by an incorrect parse tree returned by a single parser.

The second goal was to propose an algorithm which would be better than the algorithm proposed in [14]. We may suppose we have fulfilled the task as our algorithm (sect. 3.4) accepts more types of use cases than the previous one and it identifies the direct objects more precisely. Moreover we can identify violations in use cases caused by use case writers.

The thesis are supported by a proof-of-the-concept implementation of the proposed algorithm showing that our ideas are realizable.

The requirement specifications are often incomplete and the state-of-the-art linguistic tools do not allow to process all sentences correctly. Thus, we can see that the attempts of a fully automatic use cases to pro-cases conversion cannot be successful nowadays but there is a strong potential in using our algorithm in an interactive tool for such conversion.

The proposed algorithm has the limitations described in Sect. 3.6. A possible topic for our future work is the elimination of some of these limitations. The way we identify indirect objects is probably the most serious limitation of our algorithm. To solve this issue we want to try using one of the state-of-the-art linguistic approaches discussed in Chapter. 6. We intend identifying more violations caused by use case writers, as well.

Another area of our interest can be cooperation with the UCDA project [11, 12] because the project uses similar methods to solve similar problems and it also yields good results.

References

[1] Bies, A., Ferguson, M., Katz, K., MacIntyre, R.: *Bracketing Guidelines for Treebank II Style Penn Treebank Project,* Computer and Information Science Department, University of Pennsylvania, http://www.cis.upenn.edu/~treebank/

[2] Bikel, D. M. : *Design of a Multi-lingual, Parallel-processing Statistical Parsing Engine*, in Proceedings of HLT 2002, http://www.cis.upenn.edu/~dbikel/software.html#stat-parser

[3] Blaheta D., Charniak, E., *Assigning Function Tags to Parsed Text*, Proceedings of the 1st Annual Meeting of the North American Chapter of the Association for Computational Linguistic, pp. 234-240, May, 2000, Seattle

[4] Charniak, E.: *Statistical Techniques for Natural Language Parsing*, AI Magazine 18(4): 33-44 (1997)

[5] Cockburn, A.: *Writing Effective Use Cases*, Addison-Wesley Pub Co, ISBN: 0201702258, 1st edition, Jan 2000

[6] Collins, M.: *A New Statistical Parser Based on Bigram Lexical Dependencies*, ACL 1996: 184-191. 34th Annual Meeting of the Association for Computational Linguistics, 24-27 June 1996, University of California, Santa Cruz, California, USA, Proceedings. Morgan Kaufmann Publishers

[7] Fantechi, A., Gnesi, S., Lami, G., Maccari, A.: *Application of Linguistic Techniques for Use Case Analysis*, in Proceedings of RE 2002, pp. 157-164, Sep 9-13, 2002, Essen, Germany. IEEE Computer Society 2002

[8] Henderson, J. C., Brill, E.: *Exploiting diversity in natural language processing: Combining parsers*, in Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing (EMNLP-99), pp. 187-194, June, 1999, College Park, Maryland, USA

[9] Larman, C.: *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall PTR, ISBN: 0130925691, 2nd ed, 2001

[10] Lin, D.: *MINIPAR*, http://www.cs.ualberta.ca/~lindek/

[11] Liu, D., Subramaniam, K., Eberlein, A., Far, B. H.,: *Automating Transition from Use-Cases to Class Model*, IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2003), May, 2003

[12] Liu, D., Subramaniam, K., Eberlein, A., Far, B. H.,: *Natural Language Requirements Analysis and Class Model Generation Using UCDA*, Lecture Notes in Computer Science Volume 3029/2004, Innovations in Applied Artificial Intelligence: 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE, pp. 295-304, May, 2004, Springer

[13] MacDonell, S. G., Min, K., Connor, A. M., *Autonomous Requirements Specification Processing using Natural Language Processing*, Proceedings of the 14th International Conference on Adaptive Systems and Software Engineering (IASSE05), 2005

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

[15] Minnen, G., Carroll J., Pearce, D.: *Applied morphological processing of English*, Natural Language Engineering, 7(3), pp. 207-223, (2001), http://www.cogs.susx.ac.uk/lab/nlp/carroll/morph.html

[16] Object Management Group (OMG): Unified Modeling Language (UML), version 1.4, formal/2001-09-67, http://www.omg.org/uml/

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

[18] Plasil, F., Mencl, V.: *Getting "Whole Picture" Behavior in a Use Case Model*, in Proceedings of IDPT 2003, Austin, Texas, U.S.A., Dec 2003, ISSN 1090-9389

[19] Plasil, F., Mencl, V.: *Use Cases: Assembling "Whole Picture Behavior"*, TR 02/11, 2002, Dept. of Computer Science, University of New Hampshire, Durham

[20] Plasil F., Visnovsky, S.: *Behavior Protocols for Software Components*. Transactions on Software Engineering, IEEE, vol 28, no 11 (2002)

[21] Rational Software Corporation (IBM), *Rational Unified Process*, version 2003.06.01.06, 2003, http://www-130.ibm.com/developerworks/rational/products/rup/

[22] Ratnaparkhi, A.: *A Maximum Entropy Part-Of-Speech Tagger*, in Proceedings of the Empirical Methods in Natural Language Processing Conference, May 17-18, 1996. University of Pennsylvania, http://www.cis.upenn.edu/~adwait/statnlp.html

[23] Schneider, G.: Extracting and Using Trace-Free Functional Dependencies from the Penn Treebank to Reduce Parsing Complexity, in Proceedings of Treebanks and Linguistic Theories (TLT) 2003,pp. 153-164, Växjö, Sweden

[24] Stevenson, M.: *Extracting Syntactic Relations using Heuristics*, ESSLLI98 - Workshop on Automated Acquisition of Syntax and Parsing, pp. 248-256, 1998.

Appendix A: Market Place

In this appendix, we demonstrate the conversion of use cases used in [14] (The full text of these use cases can be found in[19]). Followings section in this appendix contains an output of a proof-of-the-concept prototype developed to support ideas proposed in this thesis.

A.1. Generated Actions

This section publishes actions generated from the processed use cases by the algorithm proposed in this thesis. The format of the created actions is described in Sect. 3.3.

```
M1-1.   {score: 6, actions: [?SL.submitItemDescription]}
M1-2.   {score: 5, actions: [#validateDescription]}
M1-3.   {score: 12, actions: [?SL.adjustPrice, ?SL.enterContactBillingInformation]}
M1-4.   {score: 6, actions: [#validateContactInformation]}
M1-5.   {score: 5, actions: [#verifyHistory]}
M1-6.   {score: 9, actions: [!TC.validateWholeOffer]}
M1-7.   {score: 5, actions: [#listOffer]}
M1-8.   {score: 6, actions: [#respondAuthorizationNumber]}
M1-2a1  {score: 5, actions: [%ABORT]}
M1-5a1  {score: 5, actions: [%ABORT]}
M1-6a1  {score: 5, actions: [%ABORT]}
M1-2b1  {score: 8, actions: [!SL.providePriceAssessment]}
M2-1.   {score: 7, actions: [?BU.enterBasicSearchCriterion]}
M2-2.   {score: 5, actions: [#respondList]}
M2-3.   {score: 6, actions: [?BU.requestCompleteListing]}
M2-4.   {score: 5, actions: [#respondInformation]}
M2-2a1  {score: 5, actions: [%ABORT]}
M2-2b1  {score: 6, actions: [?BU.narrowSearchResult]}
M2-2b2  {score: 6, actions: [%GOTO 2]}
M3-1.   {score: 5, actions: [?BU.acceptOffer]}
```

M3-2. {score: 5, actions: [#validateOffer]}

M3-3. {score: 18, actions: [?BU.enterBillingInformation, ?BU.selectPaymentMethod, ?BU.providePaymentDetail]}

M3-4. {score: 9, actions: [!CRA.validateInformation]}

M3-5. {score: 5, actions: [#performSale]}

M3-6. {score: 13, actions: [!SL.informOffer, #provideShippingInformation]}

M3-7. {score: 5, actions: [#transferPayment]}

M3-8. {score: 8, actions: [!BU.respondAuthorizationNumber]}

M3-2a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

M4-1. {score: 5, actions: [?SL.locateOffer]}

M4-2. {score: 5, actions: [?SL.cancelSystem]}

M4-3. {score: 7, actions: [!SL.respondRequest]}

M4-4. {score: 6, actions: [?SL.respondAuthorizationNumber]}

M4-5. {score: 6, actions: [#validateRequestIdentity]}

M4-6. {score: 5, actions: [#removeOffer]}

M4-4a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

M4-5a1 {score: 6, actions: [%GOTO 3]}

M5-1. {score: 5, actions: [?SL.locateOffer]}

M5-2. {score: 5, actions: [?SL.provideSystem]}

M5-3. {score: 7, actions: [!SL.respondRequest]}

M5-4. {score: 6, actions: [?SL.respondAuthorizationNumber]}

M5-5. {score: 6, actions: [#validateRequestIdentity]}

M5-6. {score: 5, actions: [#returnStatus]}

M5-4a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

M5-5a1 {score: 6, actions: [%GOTO 3]}

M6-1. {score: 5, actions: [?SL.locateOffer]}

M6-2. {score: 5, actions: [?SL.updateSystem]}

M6-3. {score: 7, actions: [!SL.respondRequest]}

M6-4. {score: 6, actions: [?SL.respondAuthorizationNumber]}

M6-5. {score: 6, actions: [#validateRequestIdentity]}

M6-6. {score: 5, actions: [#updateOffer]}

M6-4a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

M6-5a1 {score: 6, actions: [%GOTO 3]}

M7-1. {score: 5, actions: [?BU.searchOffer]}

M7-2. {score: 5, actions: [?BU.buyItem]}

M7-1a1 {score: 5, actions: [%ABORT]}

M7-1b1 {score: 5, actions: [%END]}

CS1-1. {score: 5, actions: [?CL.submitInformation]}

CS1-2. {score: 5, actions: [#validateDescription]}

CS1-3. {score: 12, actions: [?CL.adjustPrice, ?CL.enterContactBillingInformation]}

CS1-4. {score: 6, actions: [#validateContactInformation]}

CS1-5. {score: 7, actions: [!SU.validateSeller]}

CS1-6. {score: -100, actions: [Actor to actor communication: SU-->SL]}

CS1-7. {score: 9, actions: [!TC.validateWholeOffer]}

CS1-8. {score: 5, actions: [#listOffer]}

CS1-9. {score: 5, actions: [#respondAcknowledgment]}

CS1-2a1 {score: 5, actions: [%ABORT]}

CS1-2b1 {score: 8, actions: [!SL.providePriceAssessment]}

CS1-7a1 {score: 5, actions: [%ABORT]}

CS2-1. {score: 7, actions: [?BU.enterBasicSearchCriterion]}

CS2-2. {score: 5, actions: [#respondList]}

CS2-3. {score: 6, actions: [?BU.requestCompleteListing]}

CS2-4. {score: 5, actions: [#respondInformation]}

CS2-2a1 {score: 5, actions: [%ABORT]}

CS2-2b1 {score: 6, actions: [?BU.narrowSearchResult]}

CS2-2b2 {score: 6, actions: [%GOTO 2]}

CS3-1. {score: -100, actions: [AUX verb: "be" was used]}

CS3-2. {score: 5, actions: [#validateOffer]}

CS3-3. {score: -1000, actions: [No "terminate" verb found]}

CS3-4. {score: 18, actions: [?CL.enterBillingInformation, ?CL.selectPaymentMethod, ?CL.provideNecessaryDetail]}

CS3-5. {score: 9, actions: [!CRA.validateInformation]}

CS3-6. {score: 5, actions: [#performTrade]}

CS3-7. {score: 13, actions: [!SL.informOffer, #provideShippingInformation]}

CS3-8. {score: 5, actions: [#transferPayment]}

CS3-9. {score: 8, actions: [!BU.respondAuthorizationNumber]}

CS3-2a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

CS4-1. {score: 5, actions: [?SL.locateOffer]}

CS4-2. {score: 5, actions: [?SL.cancelSystem]}

CS4-3. {score: 7, actions: [!SL.respondRequest]}

CS4-4. {score: 6, actions: [?SL.respondAuthorizationNumber]}

CS4-5. {score: 6, actions: [#validateRequestIdentity]}

CS4-6. {score: 5, actions: [#removeOffer]}

CS4-4a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

CS4-5a1 {score: 6, actions: [%GOTO 3]}

CS5-1. {score: 5, actions: [?SL.locateOffer]}

CS5-2. {score: 5, actions: [?SL.provideSystem]}

CS5-3. {score: 7, actions: [!SL.respondRequest]}

CS5-4. {score: 6, actions: [?SL.respondAuthorizationNumber]}

CS5-5. {score: 6, actions: [#validateRequestIdentity]}

CS5-6. {score: 5, actions: [#returnStatus]}

CS5-4a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

CS5-5a1 {score: 6, actions: [%GOTO 3]}

CS6-1. {score: 5, actions: [?SL.locateOffer]}

CS6-2. {score: 5, actions: [?SL.updateSystem]}

CS6-3. {score: 7, actions: [!SL.respondRequest]}

CS6-4. {score: 6, actions: [?SL.respondAuthorizationNumber]}

CS6-5. {score: 6, actions: [#validateRequestIdentity]}

CS6-6. {score: 5, actions: [#updateOffer]}

CS6-4a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

CS6-5a1 {score: 6, actions: [%GOTO 3]}

CS7-1. {score: 5, actions: [?BU.searchOffer]}

CS7-2. {score: -1000, actions: [No "terminate" verb found]}

CS7-1a1 {score: 5, actions: [%ABORT]}

CS7-1b1 {score: 5, actions: [%END]}

CS8-1. {score: 5, actions: [?SU.searchDatabase]}

CS8-2. {score: 5, actions: [?SU.displayDescription]}

CS8-3. {score: 5, actions: [?SU.removeItem]}

CS8-1a1 {score: 5, actions: [%TERMINATE]}

CS8-2a1 {score: 5, actions: [%TERMINATE]}

CS8-2b1 {score: 6, actions: [%GOTO 2]}

CL1-1. {score: 6, actions: [?SL.submitItemDescription]}

CL1-2. {score: 5, actions: [#submitDescription]}

CL1-3. {score: 8, actions: [!SL.reportSystemResponse]}

CL1-4. {score: 8, actions: [?SL.submitPriceBillingContactInformation]}

CL1-5. {score: 8, actions: [#enterPriceBillingContactInformation]}

CL1-6. {score: 8, actions: [!SL.reportSystemResponse]}

CL1-2a1 {score: 5, actions: [%ABORT]}

CL2-1. {score: 5, actions: [?BU.submitReference]}

CL2-2. {score: 5, actions: [#submitReference]}

CL2-3. {score: 8, actions: [!SL.reportSystemResponse]}

CL2-4. {score: 12, actions:
[?BU.submitBillingShippingInformation_PaymentMethodPaymentDetail]}

CL2-5. {score: 13, actions:
[#enterBillingShippingInformation_PaymentMethod_PaymentDetail]}

CL2-6. {score: 8, actions: [!BU.reportSystemResponse]}

CL2-3a1 {score: 5, actions: [%ABORT]}

SU1-1. {score: 5, actions: [?CS.decideSeller]}

SU1-2. {score: 10, actions: [#validateSeller, #signalSystem]}

SU2-1. {score: 8, actions: [!CS.searchDatabase]}

SU2-2. {score: -1000, actions: [No "terminate" verb found]}

SU2-3. {score: 8, actions: [!CS.removeItem]}

SU2-1a1 {score: 5, actions: [%TERMINATE]}

SU2-2a1 {score: 5, actions: [%TERMINATE]}

SU2-2b1 {score: 6, actions: [%GOTO 2]}

Appendix B: Process Sale

In this appendix, we demonstrate the conversion of a use case *UC1 Process Sale* described in [9]. Following sections in this appendix contain contents of files used when testing our approach with a proof-of-the-concept prototype developed for this task and an output of the prototype.

B.1. Parse Trees

This section contains parse trees of steps of input use cases which we have obtained when employing linguistic tools. Following subsections describes the trees created by single parsers which are converted to a uniform format. We hide the identity of the parsers as mentioned in Sect. 3.2. Before processing steps of use cases, we have modified them in an automated way to meet the premisses 1 and the additional requirement (sect. 3.1). The format of leaves of published trees is (POS original-word lemma) where POS means *Part of Speech* tag.

B.1.1. Parse trees generated by parser 1

UC1-1. (S (NPB (NN Customer Customer)) (VP (VBZ arrives arrive) (PP (IN at at) (NP (NPB (NNP POS PO) (NN checkout checkout)) (PP (IN with with) (NP (NPB (NNS goods goods) (CC and and) (NNS / /)) (CC or or) (NPB (NNS services service)))))) (S (VP (TO to to) (VP (VB purchase purchase) (PUNC. . .))))))
UC1-2. (S (NPB (NNP Cashier Cashier)) (VP (VBZ starts start) (NPB (DT a a) (JJ new new) (NN sale sale) (PUNC. . .))))
UC1-3. (S (NPB (NNP Cashier Cashier)) (VP (VBZ enters enter) (NPB (NN item item) (NN identifier identifier) (PUNC. . .))))
UC1-4. (S (NPB (NNP System System)) (VP (VBZ records record) (NP (NPB (NN sale sale) (NN line line) (NN item item)) (CC and and) (NPB (NNS presents present) (NN item item) (NN description description) (PUNC, , ,)) (NPB (NN price price) (PUNC, , ,)) (CC and and) (NPB (VBG running run) (NN total total) (PUNC. . .)))))

UC1-5. (S (NPB (NNP System System)) (VP (VBZ presents present) (ADJP (JJ total total) (PP (IN with with) (NP (NPB (NNS taxes tax)) (VP (VBN calculated calculate) (PUNC. . .)))))))
UC1-6. (S (S (NPB (NNP Cashier Cashier)) (VP (VBZ tells tell) (NPB (NN Customer Customer)) (NPB (DT the the) (NN total total) (PUNC, , ,)))) (CC and and) (S (VP (VBZ asks ask) (PP (IN for for) (NPB (NN payment payment) (PUNC. . .))))))
UC1-7. (S (S (NPB (NN Customer Customer)) (VP (VBZ pays pay))) (CC and and) (S (NPB (NNP System System)) (VP (VBZ handles handle) (NPB (NN payment payment) (PUNC. . .)))))
UC1-8. (S (NPB (NNP System System)) (VP (VP (VBZ logs log) (VP (VBN completed complete) (NPB (NN sale sale)))) (CC and and) (VP (VBZ sends send) (NPB (NN sale sale) (CC and and) (NN payment payment) (NN information information)) (PP (TO to to) (NP (NPB (DT the the) (JJ external external) (NNP Accounting Accounting) (NN system system)) (CC and and) (NPB (JJ Inventory Inventory) (NN system system) (PUNC. . .)))))))
UC1-9. (S (NPB (NNP System System)) (VP (VBZ presents present) (NPB (NN receipt receipt) (PUNC. . .))))
UC1-10. (S (NPB (NN Customer Customer)) (VP (VBZ leaves leave) (PP (IN with with) (NPB (NN receipt receipt) (CC and and) (NNS goods goods) (PUNC. . .)))))
UC1-*a (S (PP (IN At At) (NPB (DT any any) (NN time time) (PUNC, , ,))) (NPB (NNP Systems System)) (VP (VBZ fails fail)))
UC1-*a1 (S (NPB (NNP Cashier Cashier)) (VP (VBZ restarts restart) (NPB (NNP System System) (PUNC, , ,)) (NP (NP (NPB (NNS logs log)) (PP (IN in in) (PUNC, , ,))) (CC and and) (NP (NPB (NNS requests request) (NN recovery recovery)) (PP (IN of of) (NPB (JJ prior prior) (NN state state) (PUNC. . .)))))))
UC1-*a2 (S (NPB (NNP System System) (NNS reconstructs reconstruct)) (VP (ADVP (RB prior prior)) (VBP state state) (PUNC: : :)))
UC1-*a2a (S (NPB (NNP System System)) (VP (VBZ detects detect) (NPB (NNS anomalies anomaly)) (VP (VBG preventing prevent) (NPB (NN recovery recovery) (PUNC: : :)))))
UC1-*a2a1 (S (NPB (NNP System System)) (VP (VP (VBZ signals signal) (NPB (NN error error)) (PP (TO to to) (NPB (DT the the) (NNP Cashier Cashier) (PUNC, , ,)))) (VP (VBZ records record) (NPB (DT the the) (NN error error) (PUNC, , ,))) (CC and and) (VP (VBZ enters enter) (NPB (DT a a) (JJ clean clean) (NN state state) (PUNC. . .)))))

UC1-*a2a2 (S (NPB (NNP Cashier Cashier)) (VP (VBZ starts start) (NPB (DT a a) (JJ new new) (NN sale sale) (PUNC. . .))))
UC1-3a (NPB (NNP Invalid Invalid) (NN identifier identifier) (PUNC: : :))
UC1-3a1 (S (NPB (NNP System System)) (VP (VP (VBZ signals signal) (NPB (NN error error))) (CC and and) (VP (VBZ rejects reject) (NPB (NN entry entry) (PUNC. . .)))))
UC1-3b (S (NPB (EX There There)) (VP (VBP are be) (ADJP (JJ multiple multiple) (PP (IN of of) (NPB (JJ same same) (NN item item) (NN category category))) (CC and and)) (VP (VBG tracking track) (SBAR (S (NPB (JJ unique unique) (NN item item) (NN identity identity)) (VP (VBZ is be) (ADJP (RB not not) (JJ important important))))))))
UC1-3b1 (S (NPB (NNP Cashier Cashier)) (VP (MD can can) (VP (VB enter enter) (NP (NPB (NN item item) (NN category category) (NN identifier identifier)) (CC and and) (NPB (DT the the) (NN quantity quantity) (PUNC. . .))))))
UC1-3-6a (S (NPB (NNP Customer Customer)) (VP (VBZ asks ask) (ADJP (JJR Cashier Cashier)) (S (VP (TO to to) (VP (VB remove remove) (NPB (DT an an) (NN item item)) (PP (IN from from) (NPB (DT the the) (NN purchase purchase) (PUNC: : :))))))))
UC1-3-6a1 (S (NPB (NNP Cashier Cashier)) (VP (VBZ enters enter) (NP (NPB (NN item item) (NN identifier identifier)) (PP (IN for for) (NP (NPB (NN removal removal)) (PP (IN from from) (NPB (NN sale sale) (PUNC. . .))))))))
UC1-3-6a2 (S (NPB (NNP System System)) (VP (VBZ displays display) (VP (VBN updated update) (NPB (VBG running run) (NN total total) (PUNC. . .)))))
UC1-3-6b (S (NPB (NN Customer Customer)) (VP (VBZ tells tell) (ADJP (JJR Cashier Cashier)) (S (VP (TO to to) (VP (VB cancel cancel) (NPB (NN sale sale) (PUNC: : :)))))))
UC1-3-6b1 (S (NPB (NNP Cashier Cashier)) (VP (VBZ cancels cancel) (NPB (NN sale sale)) (PP (IN on on) (NPB (NNP System System) (PUNC. . .)))))
UC1-3-6c (S (NPB (NNP Cashier Cashier)) (VP (VBZ suspends suspend) (NPB (DT the the) (NN sale sale) (PUNC: : :))))
UC1-3-6c1 (S (NPB (NNP System System)) (VP (VBZ records record) (NPB (NN sale sale)) (SBAR (RB so so) (IN that that) (S (NPB (PRP it it)) (VP (VBZ is be) (ADJP (JJ available available) (PP (IN for for) (NP (NPB (NN retrieval retrieval)) (PP (IN on on) (NPB (DT any any) (NNP POS PO) (NN terminal terminal) (PUNC. . .)))))))))))
UC1-4a (S (NP (NPB (DT The The) (NN system system)) (VP (VBN generated generate) (NPB (NN item item) (NN price price)))) (VP (VBZ is be) (RB not not) (VP (VBN wanted want) (PUNC: : :))))

UC1-4a1 (S (NPB (NNP Cashier Cashier)) (VP (VBZ enters enter) (VP (VB override override) (NPB (NN price price) (PUNC. . .)))))
UC1-4a2 (S (NPB (NNP System System)) (VP (VBZ presents present) (NPB (JJ new new) (NN price price) (PUNC. . .))))
UC1-5a (S (NPB (NNP System System)) (VP (VBZ detects detect) (NP (NPB (NN failure failure)) (S (VP (TO to to) (VP (VB communicate communicate) (PP (IN with with) (NPB (JJ external external) (NN tax tax) (NN calculation calculation) (NN system system) (NN service service) (PUNC: : :)))))))))
UC1-5a1 (S (S (NPB (NNP System System)) (VP (VBZ restarts restart) (NP (NPB (DT the the) (NN service service)) (PP (IN on on) (NPB (DT the the) (NNP POS PO) (NN node node) (PUNC, , ,)))))) (CC and and) (S (VP (VBZ continues continue) (PUNC. . .))))
UC1-5a1a (S (NPB (NNP System System)) (VP (VBZ detects detect) (SBAR (IN that that) (S (NPB (DT the the) (NN service service)) (VP (VBZ does do) (RB not not) (VP (VB restart restart) (PUNC. . .)))))))
UC1-5a1a1 (S (NPB (NNP System System)) (VP (VBZ signals signal) (NPB (NN error error) (PUNC. . .))))
UC1-5a1a2 (S (NPB (NNP Cashier Cashier)) (VP (MD may may) (ADVP (RB manually manually)) (VP (VP (VP (VB calculate calculate)) (CC and and) (VP (VB enter enter) (NPB (DT the the) (NN tax tax) (PUNC, , ,))) (CC or or) (VP (VB cancel cancel) (NPB (DT the the) (NN sale sale) (PUNC. . .))))))
UC1-5b (S (NPB (NNP Customer Customer)) (VP (VBZ says say) (SBAR (S (NPB (PRP they they)) (VP (VBP are be) (ADJP (JJ eligible eligible) (PP (IN for for) (NPB (DT a a) (NN discount discount) (PUNC: : :)))))))))
UC1-5b1 (S (NPB (NNP Cashier Cashier)) (VP (VBZ signals signal) (NPB (NN discount discount) (NN request request) (PUNC. . .))))
UC1-5b2 (S (NPB (NNP Cashier Cashier)) (VP (VBZ enters enter) (NPB (NN Customer Customer) (NN identification identification) (PUNC. . .))))
UC1-5b3 (S (NPB (NNP System System)) (VP (VBZ presents present) (NPB (NN discount discount) (NN total total) (PUNC, , ,)) (PP (VBN based base) (PP (IN on on) (NPB (NN discount discount) (NNS rules rule) (PUNC. . .))))))
UC1-5c (S (NPB (NN Customer Customer)) (VP (VBZ says say) (SBAR (S (NPB (PRP they they)) (VP (VBP have have) (NPB (NN credit credit)) (PP (IN in in) (NPB (PRP$ their their) (NN account account) (PUNC, , ,))) (S (VP (TO to to) (VP (VB apply apply) (PP (TO to to) (NPB (DT the the) (NN sale sale) (PUNC: : :)))))))))))

UC1-5c1 (S (NPB (NNP Cashier Cashier)) (VP (VBZ signals signal) (NPB (NN credit credit) (NN request request) (PUNC. . .))))
UC1-5c2 (S (NPB (NNP Cashier Cashier)) (VP (VBZ enters enter) (NPB (NN Customer Customer) (NN identification identification) (PUNC. . .))))
UC1-5c3 (S (S (NPB (NNP System System)) (VP (VBZ applies apply) (NPB (NN credit credit)) (PP (IN up up) (PP (TO to to) (NPB (NN price price)))) (PP (IN = =) (NPB (CD 0 0) (PUNC, , ,))))) (CC and and) (S (VP (VBZ reduces reduce) (NPB (DT the the) (VBG remaining remain) (NN credit credit) (PUNC. . .)))))
UC1-5d (S (NPB (DT Some Some) (NNS items item)) (VP (VBZ remainds remaind)))
UC1-5d1 (S (NPB (NNP Use Use) (NN case case)) (VP (VB resume resume) (PP (IN with with) (NPB (NN step step) (CD 3 3) (PUNC. . .)))))
UC1-6a (S (NPB (NNP Customer Customer)) (VP (VBZ says say) (SBAR (S (NPB (PRP they they)) (VP (VBD intended intend) (S (VP (TO to to) (VP (VB pay pay) (PP (IN by by) (NP (NPB (NN cash cash)) (CC but but) (NPB (NPB (NN don don) (POS ' ')) (NNS t t)))))) (VP (VB have have) (NPB (JJ enough enough) (NN cash cash) (PUNC: : :)))))))))
UC1-6a1a (S (NPB (NNP Customer Customer)) (VP (VBZ uses use) (NPB (DT an an) (JJ alternate alternate) (NN payment payment) (NN method method) (PUNC. . .))))
UC1-6a1b (S (NPB (NN Customer Customer)) (VP (VBZ tells tell) (ADJP (JJR Cashier Cashier)) (S (VP (TO to to) (VP (VB cancel cancel) (NPB (NN sale sale) (PUNC. . .)))))))
UC1-6a1b1 (S (NPB (NNP Cashier Cashier)) (VP (VBZ cancels cancel) (NPB (NN sale sale)) (PP (IN on on) (NPB (NNP System System) (PUNC. . .)))))
UC1-7a (S (VP (VBG Paying Pay) (PP (IN by by) (NPB (NN cash cash) (PUNC: : :)))))
UC1-7a1 (S (NPB (NNP Cashier Cashier)) (VP (VBZ enters enter) (NPB (DT the the) (NN cash cash) (NN amount amount)) (VP (VBN tendered tender) (PUNC. . .))))
UC1-7a2 (S (S (NPB (NNP System System)) (VP (VBZ presents present) (NP (NPB (DT the the) (NN balance balance)) (ADJP (JJ due due) (PUNC, , ,))))) (CC and and) (S (VP (VBZ releases release) (NPB (DT the the) (NN cash cash) (NN drawer drawer) (PUNC. . .)))))
UC1-7a3 (S (NPB (NN Cashier Cashier)) (VP (VP (VBZ deposits deposit) (NPB (NN cash cash)) (VP (VBN tendered tender))) (CC and and) (VP (VBZ returns return) (NP (NPB (NN balance balance)) (PP (IN in in) (NPB (NN cash cash)))) (PP (TO to to) (NPB (NN Customer Customer) (PUNC. . .))))))
UC1-7a4 (S (NPB (NNP System System)) (VP (VBZ records record) (NPB (DT the the) (NN cash cash) (NN payment payment) (PUNC. . .))))

UC1-7b (S (VP (VBG Paying Pay) (PP (IN by by) (NPB (NN credit credit) (PUNC: : :)))))
UC1-7b1 (S (NPB (NN Customer Customer)) (VP (VBZ enters enter) (NPB (PRP$ their their) (NN credit credit) (NN account account) (NN information information) (PUNC. . .))))
UC1-7b2 (S (NPB (NNP System System)) (VP (VBZ sends send) (NPB (NN payment payment) (NN authorization authorization) (NN request request)) (PP (TO to to) (NPB (DT an an) (JJ external external) (NN Payment Payment) (NNP Authorization Authorization) (NNP Service Service) (NNP System System) (PUNC, , ,) (CC and and) (NNS requests request) (NN payment payment) (NN approval approval) (PUNC. . .)))))
UC1-7b2a (S (NPB (NNP System System)) (VP (VBZ detects detect) (NP (NPB (NN failure failure)) (S (VP (TO to to) (VP (VB collaborate collaborate) (PP (IN with with) (NPB (JJ external external) (NN system system)))))))))
UC1-7b2a1 (S (NPB (NNP System System)) (VP (VBZ signals signal) (NPB (NN error error)) (PP (TO to to) (NPB (NNP Cashier Cashier) (PUNC. . .)))))
UC1-7b2a2 (S (NPB (NNP Cashier Cashier)) (VP (VBZ asks ask) (NPB (NN Customer Customer)) (PP (IN for for) (NPB (JJ alternate alternate) (NN payment payment) (PUNC. . .)))))
UC1-7b3 (S (NPB (NNP System System)) (VP (VP (VBZ receives receive) (NPB (NN payment payment) (NN approval approval))) (CC and and) (VP (VBZ signals signal) (NPB (NN approval approval)) (PP (TO to to) (NPB (NNP Cashier Cashier) (PUNC. . .))))))
UC1-7b3a (S (NPB (NNP System System)) (VP (VBZ receives receive) (NPB (NN payment payment) (NN denial denial) (PUNC: : :))))
UC1-7b3a1 (S (NPB (NNP System System)) (VP (VBZ signals signal) (ADJP (JJ denial denial) (PP (TO to to) (NPB (NNP Cashier Cashier) (PUNC. . .))))))
UC1-7b3a2 (S (NPB (NNP Cashier Cashier)) (VP (VBZ asks ask) (NPB (NN Customer Customer)) (PP (IN for for) (NPB (JJ alternate alternate) (NN payment payment) (PUNC. . .)))))
UC1-7b4 (S (NPB (NNP System System)) (VP (VBZ records record) (NP (NPB (DT the the) (NN credit credit) (NN payment payment) (PUNC, , ,)) (SBAR (WHNP (WDT which which)) (S (VP (VBZ includes include) (NPB (DT the the) (NN payment payment) (NN approval approval) (PUNC. . .))))))))
UC1-7b5 (S (NPB (NNP System System)) (VP (VBZ presents present) (NPB (NN credit credit) (NN payment payment) (NN signature signature) (NN input input) (NN mechanism mechanism) (PUNC. . .))))

UC1-7b6 (S (NPB (NNP Cashier Cashier)) (VP (VBZ asks ask) (NPB (NN Customer Customer)) (PP (IN for for) (NPB (DT a a) (NN credit credit) (NN payment payment) (NN signature signature) (PUNC. . .)))))
UC1-7b7 (S (NPB (NNP Customer Customer)) (VP (VBZ enters enter) (NPB (NN signature signature) (PUNC. . .))))
UC1-7c (S (VP (VBG Paying Pay) (PP (IN by by) (NPB (NN check check)))))
UC1-7d (S (VP (VBG Paying Pay) (PP (IN by by) (NPB (JJ debit debit)))))
UC1-7e (S (NPB (NNP Customer Customer)) (VP (VBZ presents present) (NPB (NNS coupons coupon) (PUNC: : :))))
UC1-7e1 (S (PP (IN Before Before) (S (VP (VBG handling handle) (NPB (NN payment payment) (PUNC, , ,))))) (NP (NPB (NNP Cashier Cashier) (NNS records record)) (SBAR (S (NPB (DT each each) (NN coupon coupon) (CC and and) (NNP System System)) (VP (VBZ reduces reduce))))) (VP (VB price price) (ADJP (RB as as) (JJ appropriate appropriate) (PUNC. . .))))
UC1-7e2 (S (NPB (NNP System System)) (VP (VBZ records record) (NPB (DT the the) (JJ used used) (NNS coupons coupon)) (PP (IN for for) (NPB (NN accounting accounting) (NNS reasons reason) (PUNC. . .)))))
UC1-7e1a (S (NPB (NNP Coupon Coupon)) (VP (VBD entered enter) (SBAR (S (VP (VBZ is be) (RB not not) (PP (IN for for) (NPB (DT any any) (VBN purchased purchase) (NN item item) (PUNC: : :))))))))
UC1-7e1a1 (S (NPB (NNP System System)) (VP (VBZ signals signal) (NPB (NN error error)) (PP (TO to to) (NPB (NNP Cashier Cashier) (PUNC. . .)))))
UC1-9a (S (NPB (EX There There)) (VP (VBP are be) (NPB (NN product product) (NNS rebates rebate) (PUNC: : :))))
UC1-9a1 (S (NPB (NNP System System)) (VP (VBZ presents present) (NPB (DT the the) (NN rebate rebate) (NNS forms form) (CC and and) (NN rebate rebate) (NNS receipts receipt)) (PP (IN for for) (NPB (DT each each) (NN item item))) (PP (IN with with) (NPB (DT a a) (NN rebate rebate) (PUNC. . .)))))
UC1-9b (NPB (NNP Customer Customer) (NNS requests request) (NN gift gift) (NN receipt receipt) (PUNC: : :))
UC1-9b1 (S (NPB (NNP Cashier Cashier) (NNS requests request) (NN gift gift) (NN receipt receipt) (CC and and) (NNP System System)) (VP (VBZ presents present) (NPB (PRP it it) (PUNC. . .))))

B.1.2. Parse trees generated by parser 2

UC1-1. (S (NP (NN Customer customer)) (VP (VBZ arrives arrive) (PP (IN at at) (NP (NNP POS po) (NN checkout checkout))) (PP (IN with with) (NP (NNS goods goods) (CC and and) (NNS / /) (CC or or) (NNS services service))) (S (VP (TO to to) (VP (VB purchase purchase))))) (. . .))
UC1-2. (S (NP (NNP Cashier cashier)) (VP (VBZ starts start) (NP (DT a a) (JJ new new) (NN sale sale))) (. . .))
UC1-3. (S (NP (NNP Cashier cashier)) (VP (VBZ enters enter) (NP (NN item item) (NN identifier identifier))) (. . .))
UC1-4. (S (NP (NNP System system)) (VP (VP (VBZ records record) (NP (NN sale sale) (NN line line) (NN item item))) (CC and and) (VP (VBZ presents present) (NP (NP (NN item item) (NN description description)) (, , ,) (NP (NN price price)) (, , ,) (CC and and) (NP (VBG running runn) (NN total total))))) (. . .))
UC1-5. (S (NP (NNP System system)) (VP (VBZ presents present) (ADJP (JJ total total) (PP (IN with with) (NP (NP (NNS taxes tax)) (VP (VBN calculated calculate)))))) (. . .))
UC1-6. (S (NP (NNP Cashier cashier)) (VP (VP (VBZ tells tell) (NP (NN Customer customer)) (NP (DT the the) (NN total total))) (, , ,) (CC and and) (VP (VBZ asks ask) (PP (IN for for) (NP (NN payment payment))))) (. . .))
UC1-7. (S (S (NP (NN Customer customer)) (VP (VBZ pays pay))) (CC and and) (S (NP (NNP System system)) (VP (VBZ handles handle) (NP (NN payment payment)))) (. . .))
UC1-8. (S (NP (NNP System system)) (VP (VP (VBZ logs log) (VP (VBN completed complete) (NP (NN sale sale)))) (CC and and) (VP (VBZ sends send) (NP (NN sale sale) (CC and and) (NN payment payment) (NN information information)) (PP (TO to to) (NP (DT the the) (JJ external external) (NN Accounting accounting) (NN system system) (CC and and) (JJ Inventory inventory) (NN system system))))) (. . .))
UC1-9. (S (NP (NNP System system)) (VP (VBZ presents present) (NP (NN receipt receipt))) (. . .))
UC1-10. (S (NP (NN Customer customer)) (VP (VBZ leaves leave) (PP (IN with with) (NP (NN receipt receipt) (CC and and) (NNS goods goods)))) (. . .))
UC1-*a (S (PP (IN At at) (NP (DT any any) (NN time time))) (, , ,) (NP (NNP Systems system)) (VP (VBZ fails fail)))
UC1-*a1 (S (NP (NNP Cashier cashier)) (VP (VP (VBZ restarts restart) (SBAR (S (NP (NNP System system)) (, , ,) (VP (VBZ logs log) (PRT (RP in in)))))) (, , ,) (CC and and)

(VP (NNS requests request) (NP (NP (NN recovery recovery)) (PP (IN of of) (NP (JJ prior prior) (NN state state)))))) (. . .))
UC1-*a2 (S (NP (NNP System system)) (VP (NNS reconstructs reconstruct) (NP (JJ prior prior) (NN state state))) (: : :))
UC1-*a2a (S (NP (NNP System system)) (VP (VBZ detects detect) (NP (NP (NNS anomalies anomaly)) (VP (VBG preventing prevent) (NP (NN recovery recovery))))) (: : :))
UC1-*a2a1 (S (NP (NNP System system)) (VP (VP (VBZ signals signal) (NP (NN error error)) (PP (TO to to) (NP (DT the the) (NNP Cashier cashier)))) (, , ,) (VP (VBZ records record) (NP (DT the the) (NN error error))) (, , ,) (CC and and) (VP (VBZ enters enter) (NP (DT a a) (JJ clean clean) (NN state state)))) (. . .))
UC1-*a2a2 (S (NP (NNP Cashier cashier)) (VP (VBZ starts start) (NP (DT a a) (JJ new new) (NN sale sale))) (. . .))
UC1-3a (NP (NP (NNP Invalid invalid) (NN identifier identifier)) (: : :))
UC1-3a1 (S (NP (NNP System system)) (VP (VP (VBZ signals signal) (NP (NN error error))) (CC and and) (VP (VBZ rejects reject) (NP (NN entry entry)))) (. . .))
UC1-3b (S (S (NP (EX There there)) (VP (VBP are be) (ADJP (JJ multiple multiple) (PP (IN of of) (NP (JJ same same) (NN item item) (NN category category)))))) (CC and and) (S (NP (VBG tracking track) (JJ unique unique) (NN item item) (NN identity identity)) (VP (VBZ is be) (ADJP (RB not not) (JJ important important)))))
UC1-3b1 (S (NP (NNP Cashier cashier)) (VP (MD can can) (VP (VB enter enter) (NP (NP (NN item item) (NN category category) (NN identifier identifier)) (CC and and) (NP (DT the the) (NN quantity quantity))))) (. . .))
UC1-3-6a (S (NP (NNP Customer customer)) (VP (VBZ asks ask) (NP (JJR Cashier cashier)) (S (VP (TO to to) (VP (VB remove remove) (NP (DT an an) (NN item item)) (PP (IN from from) (NP (DT the the) (NN purchase purchase))))))) (: : :))
UC1-3-6a1 (S (NP (NNP Cashier cashier)) (VP (VBZ enters enter) (NP (NP (NN item item) (NN identifier identifier)) (PP (IN for for) (NP (NP (NN removal removal)) (PP (IN from from) (NP (NN sale sale))))))) (. . .))
UC1-3-6a2 (S (NP (NNP System system)) (VP (VBZ displays display) (VP (VBN updated update) (NP (VBG running runn) (NN total total)))) (. . .))
UC1-3-6b (S (NP (NN Customer customer)) (VP (VBZ tells tell) (NP (JJR Cashier cashier)) (S (VP (TO to to) (VP (VB cancel cancel) (NP (NN sale sale)))))) (: : :))
UC1-3-6b1 (S (NP (NNP Cashier cashier)) (VP (VBZ cancels cancel) (NP (NN sale sale)) (PP (IN on on) (NP (NNP System system)))) (. . .))

UC1-3-6c (S (NP (NNP Cashier cashier)) (VP (VBZ suspends suspend) (NP (DT the the) (NN sale sale))) (: : :))
UC1-3-6c1 (S (NP (NNP System system)) (VP (VBZ records record) (NP (NN sale sale)) (SBAR (IN so so) (IN that that) (S (NP (PRP it it)) (VP (VBZ is be) (ADJP (JJ available available) (PP (IN for for) (NP (NP (NN retrieval retrieval)) (PP (IN on on) (NP (DT any any) (NNP POS po) (NN terminal terminal)))))))))) (. . .))
UC1-4a (S (NP (NP (DT The the) (NN system system)) (VP (VBN generated generate) (NP (NN item item) (NN price price)))) (VP (VBZ is be) (RB not not) (VP (VBN wanted want))) (: : :))
UC1-4a1 (S (NP (NNP Cashier cashier)) (VP (VBZ enters enter) (NP (NN override override) (NN price price))) (. . .))
UC1-4a2 (S (NP (NNP System system)) (VP (VBZ presents present) (NP (JJ new new) (NN price price))) (. . .))
UC1-5a (S (NP (NNP System system)) (VP (VBZ detects detect) (NP (NP (NN failure failure)) (S (VP (TO to to) (VP (VB communicate communicate) (PP (IN with with) (NP (JJ external external) (NN tax tax) (NN calculation calculation) (NN system system) (NN service service)))))))) (: : :))
UC1-5a1 (S (NP (NNP System system)) (VP (VP (VBZ restarts restart) (NP (NP (DT the the) (NN service service)) (PP (IN on on) (NP (DT the the) (NNP POS po) (NN node node))))) (, , ,) (CC and and) (VP (VBZ continues continue))) (. . .))
UC1-5a1a (S (NP (NNP System system)) (VP (VBZ detects detect) (SBAR (IN that that) (S (NP (DT the the) (NN service service)) (VP (VBZ does do) (RB not not) (VP (VB restart restart)))))) (. . .))
UC1-5a1a1 (S (NP (NNP System system)) (VP (VBZ signals signal) (NP (NN error error))) (. . .))
UC1-5a1a2 (S (NP (NNP Cashier cashier)) (VP (MD may may) (ADVP (RB manually manually)) (VP (VP (VB calculate calculate)) (CC and and) (VP (VB enter enter) (NP (DT the the) (NN tax tax))) (, , ,) (CC or or) (VP (VB cancel cancel) (NP (DT the the) (NN sale sale))))) (. . .))
UC1-5b (S (NP (NNP Customer customer)) (VP (VBZ says say) (SBAR (S (NP (PRP they they)) (VP (VBP are be) (ADJP (JJ eligible eligible) (PP (IN for for) (NP (DT a a) (NN discount discount)))))))) (: : :))
UC1-5b1 (S (NP (NNP Cashier cashier)) (VP (VBZ signals signal) (NP (NN discount discount) (NN request request))) (. . .))

UC1-5b2 (S (NP (NNP Cashier cashier)) (VP (VBZ enters enter) (NP (NN Customer customer) (NN identification identification))) (. . .))
UC1-5b3 (S (NP (NNP System system)) (VP (VBZ presents present) (NP (NP (NN discount discount) (NN total total)) (, , ,) (PP (VBN based base) (PP (IN on on) (NP (NN discount discount) (NNS rules rule)))))) (. . .))
UC1-5c (S (NP (NN Customer customer)) (VP (VBZ says say) (SBAR (S (NP (PRP they they)) (VP (VBP have have) (NP (NN credit credit)) (PP (IN in in) (NP (PRP$ their their) (NN account account))) (, , ,) (S (VP (TO to to) (VP (VB apply apply) (PP (TO to to) (NP (DT the the) (NN sale sale)))))))))) (: : :))
UC1-5c1 (S (NP (NNP Cashier cashier)) (VP (VBZ signals signal) (NP (NN credit credit) (NN request request))) (. . .))
UC1-5c2 (S (NP (NNP Cashier cashier)) (VP (VBZ enters enter) (NP (NN Customer customer) (NN identification identification))) (. . .))
UC1-5c3 (S (NP (NNP System system)) (VP (VP (VBZ applies apply) (NP (NN credit credit)) (PP (IN up up) (PP (TO to to) (NP (NN price price)))) (PP (IN = =) (NP (CD 0 0)))) (, , ,) (CC and and) (VP (VBZ reduces reduce) (NP (DT the the) (VBG remaining remain) (NN credit credit)))) (. . .))
UC1-5d (S (NP (DT Some some) (NNS items item)) (VP (VBZ remainds remaind)))
UC1-5d1 (S (NP (NNP Use use) (NN case case)) (VP (VB resume resume) (PP (IN with with) (NP (NN step step) (CD 3 3)))) (. . .))
UC1-6a (S (S (NP (NNP Customer customer)) (VP (VBZ says say) (SBAR (S (NP (PRP they they)) (VP (VBD intended intend) (S (VP (TO to to) (VP (VB pay pay) (PP (IN by by) (NP (NN cash cash))))))))))) (CC but but) (S (NP (NN don don) (" ' ') (NNS t t)) (VP (VBP have have) (NP (JJ enough enough) (NN cash cash)))) (: : :))
UC1-6a1a (S (NP (NNP Customer customer)) (VP (VBZ uses use) (NP (DT an an) (JJ alternate alternate) (NN payment payment) (NN method method))) (. . .))
UC1-6a1b (S (NP (NN Customer customer)) (VP (VBZ tells tell) (NP (JJR Cashier cashier)) (S (VP (TO to to) (VP (VB cancel cancel) (NP (NN sale sale)))))) (. . .))
UC1-6a1b1 (S (NP (NNP Cashier cashier)) (VP (VBZ cancels cancel) (NP (NN sale sale)) (PP (IN on on) (NP (NNP System system)))) (. . .))
UC1-7a (S (VP (VBG Paying pay) (PP (IN by by) (NP (NN cash cash)))) (: : :))
UC1-7a1 (S (NP (NNP Cashier cashier)) (VP (VBZ enters enter) (NP (NP (DT the the) (NN cash cash) (NN amount amount)) (VP (VBN tendered tender)))) (. . .))

UC1-7a2 (S (NP (NNP System system)) (VP (VP (VBZ presents present) (NP (NP (DT the the) (NN balance balance)) (ADJP (JJ due due)))) (, , ,) (CC and and) (VP (VBZ releases release) (NP (DT the the) (NN cash cash) (NN drawer drawer)))) (. . .))
UC1-7a3 (S (NP (NN Cashier cashier) (NNS deposits deposit)) (VP (NN cash cash) (VP (VBN tendered tender)) (CC and and) (VP (VBZ returns return) (NP (NP (NN balance balance)) (PP (IN in in) (NP (NN cash cash)))) (PP (TO to to) (NP (NN Customer customer))))) (. . .))
UC1-7a4 (S (NP (NNP System system)) (VP (VBZ records record) (NP (DT the the) (NN cash cash) (NN payment payment))) (. . .))
UC1-7b (S (VP (VBG Paying pay) (PP (IN by by) (NP (NN credit credit)))) (: : :))
UC1-7b1 (S (NP (NN Customer customer)) (VP (VBZ enters enter) (NP (PRP$ their their) (NN credit credit) (NN account account) (NN information information))) (. . .))
UC1-7b2 (S (NP (NNP System system)) (VP (VP (VBZ sends send) (NP (NN payment payment) (NN authorization authorization) (NN request request)) (PP (TO to to) (NP (DT an an) (JJ external external) (NN Payment payment) (NNP Authorization authorization) (NNP Service service) (NNP System system)))) (, , ,) (CC and and) (VP (NNS requests request) (NP (NN payment payment) (NN approval approval)))) (. . .))
UC1-7b2a (S (NP (NNP System system)) (VP (VBZ detects detect) (NP (NP (NN failure failure)) (S (VP (TO to to) (VP (VB collaborate collaborate) (PP (IN with with) (NP (JJ external external) (NN system system)))))))))
UC1-7b2a1 (S (NP (NNP System system)) (VP (VBZ signals signal) (NP (NN error error)) (PP (TO to to) (NP (NNP Cashier cashier)))) (. . .))
UC1-7b2a2 (S (NP (NNP Cashier cashier)) (VP (VBZ asks ask) (NP (NN Customer customer)) (PP (IN for for) (NP (JJ alternate alternate) (NN payment payment)))) (. . .))
UC1-7b3 (S (NP (NNP System system)) (VP (VP (VBZ receives receive) (NP (NN payment payment) (NN approval approval))) (CC and and) (VP (VBZ signals signal) (NP (NN approval approval)) (PP (TO to to) (NP (NNP Cashier cashier))))) (. . .))
UC1-7b3a (S (NP (NNP System system)) (VP (VBZ receives receive) (NP (NN payment payment) (NN denial denial))) (: : :))
UC1-7b3a1 (S (NP (NNP System system)) (VP (VBZ signals signal) (ADJP (JJ denial denial) (PP (TO to to) (NP (NNP Cashier cashier))))) (. . .))
UC1-7b3a2 (S (NP (NNP Cashier cashier)) (VP (VBZ asks ask) (NP (NN Customer customer)) (PP (IN for for) (NP (JJ alternate alternate) (NN payment payment)))) (. . .))

UC1-7b4 (S (NP (NNP System system)) (VP (VBZ records record) (NP (NP (DT the the) (NN credit credit) (NN payment payment)) (, , ,) (SBAR (WHNP (WDT which which)) (S (VP (VBZ includes include) (NP (DT the the) (NN payment payment) (NN approval approval))))))) (. . .))
UC1-7b5 (S (NP (NNP System system)) (VP (VBZ presents present) (NP (NN credit credit) (NN payment payment) (NN signature signature) (NN input input) (NN mechanism mechanism))) (. . .))
UC1-7b6 (S (NP (NNP Cashier cashier)) (VP (VBZ asks ask) (NP (NN Customer customer)) (PP (IN for for) (NP (DT a a) (NN credit credit) (NN payment payment) (NN signature signature)))) (. . .))
UC1-7b7 (S (NP (NNP Customer customer)) (VP (VBZ enters enter) (NP (NN signature signature))) (. . .))
UC1-7c (S (VP (VBG Paying pay) (PP (IN by by) (NP (NN check check)))))
UC1-7d (S (VP (VBG Paying pay) (PP (IN by by) (NP (JJ debit debit)))))
UC1-7e (S (NP (NNP Customer customer)) (VP (VBZ presents present) (NP (NNS coupons coupon))) (: : :))
UC1-7e1 (S (PP (IN Before before) (S (VP (VBG handling handle) (NP (NN payment payment))))) (, , ,) (S (NP (NNP Cashier cashier)) (VP (VBZ records record) (NP (DT each each) (NN coupon coupon)))) (CC and and) (S (NP (NNP System system)) (VP (VBZ reduces reduce) (NP (NN price price)) (PP (IN as as) (ADJP (JJ appropriate appropriate))))) (. . .))
UC1-7e2 (S (NP (NNP System system)) (VP (VBZ records record) (NP (DT the the) (JJ used used) (NNS coupons coupon)) (PP (IN for for) (NP (NN accounting accounting) (NNS reasons reason)))) (. . .))
UC1-7e1a (S (NP (NP (NN Coupon coupon)) (VP (VBN entered enter))) (VP (VBZ is be) (RB not not) (PP (IN for for) (NP (DT any any) (VBN purchased purchase) (NN item item)))) (: : :))
UC1-7e1a1 (S (NP (NNP System system)) (VP (VBZ signals signal) (NP (NN error error)) (PP (TO to to) (NP (NNP Cashier cashier)))) (. . .))
UC1-9a (S (NP (EX There there)) (VP (VBP are be) (NP (NN product product) (NNS rebates rebate))) (: : :))
UC1-9a1 (S (NP (NNP System system)) (VP (VBZ presents present) (NP (DT the the) (NN rebate rebate) (NNS forms form) (CC and and) (NN rebate rebate) (NNS receipts receipt))

(PP (IN for for) (NP (DT each each) (NN item item))) (PP (IN with with) (NP (DT a a) (NN rebate rebate)))) (. . .))
UC1-9b (NP (NP (NNP Customer customer) (NNS requests request) (NN gift gift) (NN receipt receipt)) (: : :))
UC1-9b1 (S (NP (NNP Cashier cashier) (NNS requests request) (NN gift gift) (NN receipt receipt) (CC and and) (NNP System system)) (VP (VBZ presents present) (NP (PRP it it))) (. . .))

B.1.3. Parse trees generated by parser 3

UC1-1. (S (NP (NN Customer customer)) (VP (VBZ arrives arrive) (PP (IN at at) (NP (NNP POS po) (NN checkout checkout))) (PP (IN with with) (NP (NNS goods goods) (CC and and) (NNS / /) (CC or or) (NNS services service))) (S (VP (TO to to) (VP (VB purchase purchase))))) (. . .))
UC1-2. (S (NP (NN Cashier cashier)) (VP (VBZ starts start) (NP (DT a a) (JJ new new) (NN sale sale))) (. . .))
UC1-3. (S (NP (NN Cashier cashier)) (VP (VBZ enters enter) (NP (NN item item) (NN identifier identifier))) (. . .))
UC1-4. (S (NP (NN System system)) (VP (VP (VBZ records record) (NP (NN sale sale) (NN line line) (NN item item))) (CC and and) (VP (VBZ presents present) (NP (NP (NN item item) (NN description description)) (, , ,) (NP (NN price price)) (, , ,) (CC and and) (NP (VBG running runn) (NN total total))))) (. . .))
UC1-5. (S (NP (NN System system) (NNS presents present)) (VP (VBP total total) (PP (IN with with) (NP (NP (NNS taxes tax)) (VP (VBN calculated calculate))))) (. . .))
UC1-6. (S (NP (NN Cashier cashier)) (VP (VP (VBZ tells tell) (S (VP (NN Customer customer) (NP (DT the the) (NN total total))))) (, , ,) (CC and and) (VP (VBZ asks ask) (PP (IN for for) (NP (NN payment payment))))) (. . .))
UC1-7. (S (S (NP (NN Customer customer)) (VP (VBZ pays pay))) (CC and and) (S (NP (NNP System system)) (VP (VBZ handles handle) (NP (NN payment payment)))) (. . .))
UC1-8. (S (NP (NN System system)) (VP (VP (VBZ logs log) (NP (VBN completed complete) (NN sale sale))) (CC and and) (VP (VBZ sends send) (NP (NN sale sale) (CC and and) (NN payment payment) (NN information information)) (PP (TO to to) (NP (DT

the the) (NX (NX (JJ external external) (NNP Accounting accounting) (NN system system)) (CC and and) (NX (NN Inventory inventory) (NN system system))))))) (. . .))
UC1-9. (S (NP (NN System system)) (VP (VBZ presents present) (NP (NN receipt receipt))) (. . .))
UC1-10. (S (NP (NN Customer customer)) (VP (VBZ leaves leave) (PP (IN with with) (NP (NN receipt receipt) (CC and and) (NNS goods goods)))) (. . .))
UC1-*a (S (PP (IN At at) (NP (DT any any) (NN time time))) (, , ,) (NP (NNP Systems system)) (VP (VBZ fails fail)))
UC1-*a1 (S (NP (NN Cashier cashier)) (VP (VP (VBZ restarts restart) (NP (NNP System system))) (, , ,) (VP (VBZ logs log) (PRT (RP in in))) (, , ,) (CC and and) (VP (NNS requests request) (NP (NP (NN recovery recovery)) (PP (IN of of) (NP (JJ prior prior) (NN state state)))))) (. . .))
UC1-*a2 (S (NP (NN System system)) (VP (VBZ reconstructs reconstruct) (NP (JJ prior prior) (NN state state))) (: : :))
UC1-*a2a (S (NP (NN System system)) (VP (VBZ detects detect) (NP (NNS anomalies anomaly) (VBG preventing prevent) (NN recovery recovery))) (: : :))
UC1-*a2a1 (S (NP (NN System system)) (VP (VP (VBZ signals signal) (NP (NN error error)) (PP (TO to to) (NP (DT the the) (NN Cashier cashier)))) (, , ,) (VP (VBZ records record) (NP (DT the the) (NN error error))) (, , ,) (CC and and) (VP (VBZ enters enter) (NP (DT a a) (JJ clean clean) (NN state state)))) (. . .))
UC1-*a2a2 (S (NP (NN Cashier cashier)) (VP (VBZ starts start) (NP (DT a a) (JJ new new) (NN sale sale))) (. . .))
UC1-3a (NP (JJ Invalid invalid) (NN identifier identifier) (: : :))
UC1-3a1 (S (NP (NN System system)) (VP (VP (VBZ signals signal) (NP (NN error error))) (CC and and) (VP (VBZ rejects reject) (NP (NN entry entry)))) (. . .))
UC1-3b (S (S (NP (EX There there)) (VP (AUX are are) (ADJP (JJ multiple multiple) (PP (IN of of) (NP (NP (JJ same same) (NN item item) (NN category category)) (CC and and) (NP (VBG tracking track) (JJ unique unique) (NN item item) (NN identity identity))))))) (VP (AUX is is) (RB not not) (ADJP (JJ important important))))
UC1-3b1 (S (NP (NN Cashier cashier)) (VP (MD can can) (VP (VB enter enter) (NP (NP (NN item item) (NN category category) (NN identifier identifier)) (CC and and) (NP (DT the the) (NN quantity quantity))))) (. . .))

UC1-3-6a (S (NP (NN Customer customer)) (VP (VBZ asks ask) (NP (NN Cashier cashier)) (S (VP (TO to to) (VP (VB remove remove) (NP (NP (DT an an) (NN item item)) (PP (IN from from) (NP (DT the the) (NN purchase purchase)))))))) (: : :))
UC1-3-6a1 (S (NP (NN Cashier cashier)) (VP (VBZ enters enter) (NP (NP (NN item item) (NN identifier identifier)) (PP (IN for for) (NP (NP (NN removal removal)) (PP (IN from from) (NP (NN sale sale))))))) (. . .))
UC1-3-6a2 (S (NP (NN System system)) (VP (VBZ displays display) (NP (VBN updated update) (VBG running runn) (NN total total))) (. . .))
UC1-3-6b (S (NP (NN Customer customer)) (VP (VBZ tells tell) (NP (NN Cashier cashier)) (S (VP (TO to to) (VP (VB cancel cancel) (NP (NN sale sale)))))) (: : :))
UC1-3-6b1 (S (NP (NN Cashier cashier)) (VP (VBZ cancels cancel) (NP (NP (NN sale sale)) (PP (IN on on) (NP (NNP System system))))) (. . .))
UC1-3-6c (S (NP (NN Cashier cashier)) (VP (VBZ suspends suspend) (NP (DT the the) (NN sale sale))) (: : :))
UC1-3-6c1 (S (NP (NN System system)) (VP (VBZ records record) (NP (NN sale sale)) (SBAR (IN so so) (IN that that) (S (NP (PRP it it)) (VP (AUX is is) (ADJP (JJ available available) (PP (IN for for) (NP (NN retrieval retrieval)))) (PP (IN on on) (NP (DT any any) (NNP POS po) (NN terminal terminal))))))) (. . .))
UC1-4a (S (NP (DT The the) (NN system system)) (VP (VBD generated generate) (SBAR (S (NP (NN item item) (NN price price)) (VP (AUX is is) (RB not not) (VP (VBN wanted want)))))) (: : :))
UC1-4a1 (S (NP (NN Cashier cashier)) (VP (VBZ enters enter) (NP (NN override override) (NN price price))) (. . .))
UC1-4a2 (S (NP (NN System system)) (VP (VBZ presents present) (NP (JJ new new) (NN price price))) (. . .))
UC1-5a (S (NP (NN System system)) (VP (VBZ detects detect) (S (NP (NN failure failure) (S (VP (TO to to) (VP (VB communicate communicate) (PP (IN with with) (NP (JJ external external) (NN tax tax) (NN calculation calculation) (NN system system) (NN service service))))))))) (: : :))
UC1-5a1 (S (NP (NN System system)) (VP (VP (VBZ restarts restart) (NP (NP (DT the the) (NN service service)) (PP (IN on on) (NP (DT the the) (NNP POS po) (NN node node))))) (, , ,) (CC and and) (VP (VBZ continues continue))) (. . .))

UC1-5a1a (S (NP (NN System system)) (VP (VBZ detects detect) (SBAR (IN that that) (S (NP (DT the the) (NN service service)) (VP (AUX does does) (RB not not) (VP (VB restart restart)))))) (. . .))
UC1-5a1a1 (S (NP (NN System system)) (VP (VBZ signals signal) (NP (NN error error))) (. . .))
UC1-5a1a2 (S (NP (NN Cashier cashier)) (VP (MD may may) (ADVP (RB manually manually)) (VP (VP (VB calculate calculate) (CC and and) (VB enter enter) (NP (DT the the) (NN tax tax))) (, , ,) (CC or or) (VP (VB cancel cancel) (NP (DT the the) (NN sale sale))))) (. . .))
UC1-5b (S (NP (NN Customer customer)) (VP (VBZ says say) (SBAR (S (NP (PRP they they)) (VP (AUX are are) (ADJP (JJ eligible eligible) (PP (IN for for) (NP (DT a a) (NN discount discount)))))))) (: : :))
UC1-5b1 (S (NP (NN Cashier cashier)) (VP (VBZ signals signal) (NP (NN discount discount) (NN request request))) (. . .))
UC1-5b2 (S (NP (NN Cashier cashier)) (VP (VBZ enters enter) (NP (NN Customer customer) (NN identification identification))) (. . .))
UC1-5b3 (S (NP (NN System system)) (VP (VBZ presents present) (NP (NP (NN discount discount) (NN total total)) (, , ,) (VP (VBN based base) (PP (IN on on) (NP (NN discount discount) (NNS rules rule)))))) (. . .))
UC1-5c (S (NP (NN Customer customer)) (VP (VBZ says say) (SBAR (S (NP (PRP they they)) (VP (AUX have have) (NP (NN credit credit)) (PP (IN in in) (NP (NP (PRP$ their their) (NN account account)) (, , ,) (SBAR (S (VP (TO to to) (VP (VB apply apply) (PP (TO to to) (NP (DT the the) (NN sale sale))))))))))))) (: : :))
UC1-5c1 (S (NP (NN Cashier cashier)) (VP (VBZ signals signal) (NP (NN credit credit) (NN request request))) (. . .))
UC1-5c2 (S (NP (NN Cashier cashier)) (VP (VBZ enters enter) (NP (NN Customer customer) (NN identification identification))) (. . .))
UC1-5c3 (S (NP (NN System system)) (VP (VP (VBZ applies apply) (NP (NN credit credit)) (PRT (RP up up)) (PP (TO to to) (NP (NN price price))) (S (VP (SYM = =) (NP (CD 0 0))))) (, , ,) (CC and and) (VP (VBZ reduces reduce) (NP (DT the the) (VBG remaining remain) (NN credit credit)))) (. . .))
UC1-5d (NP (DT Some some) (NNS items item) (NNS remainds remaind))
UC1-5d1 (S (NP (NN Use use) (NN case case)) (VP (VB resume resume) (PP (IN with with) (NP (NN step step) (CD 3 3)))) (. . .))

UC1-6a (S (NP (NN Customer customer)) (VP (VBZ says say) (SBAR (S (S (NP (PRP they they)) (VP (VBD intended intend) (S (VP (TO to to) (VP (VB pay pay) (PP (IN by by) (NP (NN cash cash)))))))) (CC but but) (S (NP (NP (NNP don don) (POS ' ')) (NNS t t)) (VP (AUX have have) (NP (JJ enough enough) (NN cash cash))))))) (: : :))
UC1-6a1a (S (NP (NN Customer customer)) (VP (VBZ uses use) (NP (DT an an) (JJ alternate alternate) (NN payment payment) (NN method method))) (. . .))
UC1-6a1b (S (NP (NN Customer customer)) (VP (VBZ tells tell) (NP (NN Cashier cashier)) (S (VP (TO to to) (VP (VB cancel cancel) (NP (NN sale sale)))))) (. . .))
UC1-6a1b1 (S (NP (NN Cashier cashier)) (VP (VBZ cancels cancel) (NP (NP (NN sale sale)) (PP (IN on on) (NP (NNP System system))))) (. . .))
UC1-7a (S (VP (VBG Paying pay) (PP (IN by by) (NP (NN cash cash)))) (: : :))
UC1-7a1 (S (NP (NN Cashier cashier)) (VP (VBZ enters enter) (NP (NP (DT the the) (NN cash cash) (NN amount amount)) (VP (VBN tendered tender)))) (. . .))
UC1-7a2 (S (NP (NN System system)) (VP (VP (VBZ presents present) (NP (NP (DT the the) (NN balance balance)) (ADJP (JJ due due)))) (, , ,) (CC and and) (VP (VBZ releases release) (NP (DT the the) (NN cash cash) (NN drawer drawer)))) (. . .))
UC1-7a3 (S (NP (NN Cashier cashier)) (VP (VP (VBZ deposits deposit) (ADJP (NN cash cash) (VBN tendered tender))) (CC and and) (VP (VBZ returns return) (NP (NP (NN balance balance)) (PP (IN in in) (NP (NN cash cash)))) (PP (TO to to) (NP (NN Customer customer))))) (. . .))
UC1-7a4 (S (NP (NN System system)) (VP (VBZ records record) (NP (DT the the) (NN cash cash) (NN payment payment))) (. . .))
UC1-7b (S (VP (VBG Paying pay) (PP (IN by by) (NP (NN credit credit)))) (: : :))
UC1-7b1 (S (NP (NN Customer customer)) (VP (VBZ enters enter) (NP (PRP$ their their) (NN credit credit) (NN account account) (NN information information))) (. . .))
UC1-7b2 (S (NP (NN System system)) (VP (VP (VBZ sends send) (NP (NN payment payment) (NN authorization authorization) (NN request request)) (PP (TO to to) (NP (DT an an) (JJ external external) (NN Payment payment) (NN Authorization authorization) (NNP Service service) (NNP System system)))) (, , ,) (CC and and) (VP (NNS requests request) (NP (NN payment payment) (NN approval approval)))) (. . .))
UC1-7b2a (S (NP (NN System system)) (VP (VBZ detects detect) (S (NP (NN failure failure) (S (VP (TO to to) (VP (VB collaborate collaborate) (PP (IN with with) (NP (JJ external external) (NN system system))))))))))

UC1-7b2a1 (S (NP (NN System system)) (VP (VBZ signals signal) (NP (NP (NN error error)) (PP (TO to to) (NP (NN Cashier cashier))))) (. . .))
UC1-7b2a2 (S (NP (NN Cashier cashier)) (VP (VBZ asks ask) (NP (NN Customer customer)) (PP (IN for for) (NP (JJ alternate alternate) (NN payment payment)))) (. . .))
UC1-7b3 (S (NP (NN System system)) (VP (VP (VBZ receives receive) (NP (NN payment payment) (NN approval approval))) (CC and and) (VP (VBZ signals signal) (NP (NN approval approval)) (PP (TO to to) (NP (NN Cashier cashier))))) (. . .))
UC1-7b3a (S (NP (NN System system)) (VP (VBZ receives receive) (NP (NN payment payment) (NN denial denial))) (: : :))
UC1-7b3a1 (S (NP (NN System system)) (VP (VBZ signals signal) (NP (NP (NN denial denial)) (PP (TO to to) (NP (NN Cashier cashier))))) (. . .))
UC1-7b3a2 (S (NP (NN Cashier cashier)) (VP (VBZ asks ask) (NP (NN Customer customer)) (PP (IN for for) (NP (JJ alternate alternate) (NN payment payment)))) (. . .))
UC1-7b4 (S (NP (NN System system)) (VP (VBZ records record) (NP (NP (DT the the) (NN credit credit) (NN payment payment)) (, , ,) (SBAR (WHNP (WDT which which)) (S (VP (VBZ includes include) (NP (DT the the) (NN payment payment) (NN approval approval))))))) (. . .))
UC1-7b5 (S (NP (NN System system)) (VP (VBZ presents present) (NP (NN credit credit) (NN payment payment) (NN signature signature) (NN input input) (NN mechanism mechanism))) (. . .))
UC1-7b6 (S (NP (NN Cashier cashier)) (VP (VBZ asks ask) (NP (NN Customer customer)) (PP (IN for for) (NP (DT a a) (NN credit credit) (NN payment payment) (NN signature signature)))) (. . .))
UC1-7b7 (S (NP (NN Customer customer)) (VP (VBZ enters enter) (NP (NN signature signature))) (. . .))
UC1-7c (S (VP (VBG Paying pay) (PP (IN by by) (NP (NN check check)))))
UC1-7d (S (VP (VBG Paying pay) (PP (IN by by) (NP (NN debit debit)))))
UC1-7e (S (NP (NN Customer customer)) (VP (VBZ presents present) (NP (NNS coupons coupon))) (: : :))
UC1-7e1 (S (PP (IN Before before) (S (VP (VBG handling handle) (NP (NN payment payment))))) (, , ,) (S (NP (NN Cashier cashier)) (VP (VBZ records record) (NP (DT each each) (NN coupon coupon)))) (CC and and) (S (NP (NNP System system)) (VP (VBZ reduces reduce) (NP (NN price price)) (PP (IN as as) (ADJP (JJ appropriate appropriate))))) (. . .))

UC1-7e2 (S (NP (NN System system)) (VP (VBZ records record) (NP (DT the the) (JJ used used) (NNS coupons coupon)) (PP (IN for for) (NP (NN accounting accounting) (NNS reasons reason)))) (. . .))
UC1-7e1a (S (NP (NP (NN Coupon coupon)) (VP (VBN entered enter))) (VP (AUX is is) (RB not not) (PP (IN for for) (NP (DT any any) (JJ purchased purchased) (NN item item)))) (: : :))
UC1-7e1a1 (S (NP (NN System system)) (VP (VBZ signals signal) (NP (NP (NN error error)) (PP (TO to to) (NP (NN Cashier cashier))))) (. . .))
UC1-9a (SINV (ADVP (RB There there)) (VP (AUX are are)) (NP (NN product product) (NNS rebates rebate)) (: : :))
UC1-9a1 (S (NP (NN System system)) (VP (VBZ presents present) (NP (DT the the) (NX (NX (NN rebate rebate) (NNS forms form)) (CC and and) (NX (NN rebate rebate) (NNS receipts receipt)))) (PP (IN for for) (NP (NP (DT each each) (NN item item)) (PP (IN with with) (NP (DT a a) (NN rebate rebate)))))) (. . .))
UC1-9b (NP (NP (NP (NN Customer customer) (NNS requests request)) (NP (NN gift gift) (NN receipt receipt))) (: : :))
UC1-9b1 (S (NP (NP (NN Cashier cashier) (NNS requests request) (NN gift gift) (NN receipt receipt)) (CC and and) (NP (NNP System system))) (VP (VBZ presents present) (NP (PRP it it))) (. . .))

B.2. Generated Actions

This section publishes actions generated from the processed use cases by the algorithm proposed in this thesis. The format of the created actions is described in Sect. 3.3.

UC1-1. {score: 6, actions: [?CUS.arrivePoCheckout]}
UC1-2. {score: 6, actions: [?CAS.startNewSale]}
UC1-3. {score: 6, actions: [?CAS.enterItemIdentifier]}
UC1-4. {score: 17, actions: [#recordSaleLineItem, #presentItemDescription_Price_Total]}
UC1-5. {score: 5, actions: [#presentTax]}
UC1-6. {score: 10, actions: [?CAS.tellTotal, ?CAS.askPayment]}
UC1-7. {score: 8, actions: [?CUS.pay, #handlePayment]}

UC1-8. {score: 25, actions: [#logSale, !AS.sendSalePaymentInformation, !IS.sendSalePaymentInformation]}
UC1-9. {score: 5, actions: [#presentReceipt]}
UC1-10. {score: 6, actions: [?CUS.leaveReceiptGoods]}
UC1-*a1 {score: 13, actions: [?CAS.restartSystem, ?CAS.log, ?CAS.requestRecovery]}
UC1-*a2 {score: 6, actions: [#reconstructPriorState]}
UC1-*a2a1 {score: 18, actions: [!CAS.signalError, #recordError, #enterCleanState]}
UC1-*a2a2 {score: 6, actions: [?CAS.startNewSale]}
UC1-3a1 {score: 10, actions: [#signalError, #rejectEntry]}
UC1-3b1 {score: -100, actions: [AUX verb: "can" was used]}
UC1-3-6a1 {score: 6, actions: [?CAS.enterItemIdentifier]}
UC1-3-6a2 {score: 5, actions: [#displayTotal]}
UC1-3-6b1 {score: 5, actions: [?CAS.cancelSale]}
UC1-3-6c1 {score: 5, actions: [#recordSale]}
UC1-4a1 {score: 6, actions: [?CAS.enterOverridePrice]}
UC1-4a2 {score: 6, actions: [#presentNewPrice]}
UC1-5a1 {score: 8, actions: [#restartService, #continue]}
UC1-5a1a1 {score: 5, actions: [#signalError]}
UC1-5a1a2 {score: -100, actions: [AUX verb: "may" was used]}
UC1-5b1 {score: 6, actions: [?CAS.signalDiscountRequest]}
UC1-5b2 {score: -100, actions: [Actor to actor communication: CAS-->CUS]}
UC1-5b3 {score: 6, actions: [#presentDiscountTotal]}
UC1-5c1 {score: 6, actions: [?CAS.signalCreditRequest]}
UC1-5c2 {score: -100, actions: [Actor to actor communication: CAS-->CUS]}
UC1-5c3 {score: 10, actions: [#applyCredit, #reduceCredit]}
UC1-5d1 {score: 6, actions: [%GOTO 3]}
UC1-6a1b1 {score: 5, actions: [?CAS.cancelSale]}
UC1-7a1 {score: 6, actions: [?CAS.enterCashAmount]}
UC1-7a2 {score: 11, actions: [#presentBalance, #releaseCashDrawer]}
UC1-7a3 {score: -95, actions: [?CAS.depositCash, Actor to actor communication: CAS-->CUS]}
UC1-7a4 {score: 6, actions: [#recordCashPayment]}
UC1-7b1 {score: 7, actions: [?CUS.enterCreditAccountInformation]}

```
UC1-7b2    {score: 18, actions: [!PASS.sendPaymentAuthorizationRequest,
#requestPaymentApproval]}
UC1-7b2a1     {score: 7, actions: [!CAS.signalError]}
UC1-7b2a2     {score: -100, actions: [Actor to actor communication: CAS-->CUS]}
UC1-7b3    {score: 13, actions: [#receivePaymentApproval, !CAS.signalApproval]}
UC1-7b3a1     {score: 7, actions: [!CAS.signalDenial]}
UC1-7b3a2     {score: -100, actions: [Actor to actor communication: CAS-->CUS]}
UC1-7b4    {score: 6, actions: [#recordCreditPayment]}
UC1-7b5    {score: 9, actions: [#presentCreditPaymentSignatureInputMechanism]}
UC1-7b6    {score: -100, actions: [Actor to actor communication: CAS-->CUS]}
UC1-7b7    {score: 5, actions: [?CUS.enterSignature]}
UC1-7e1    {score: -1000, actions: [No "terminate" verb found]}
UC1-7e2    {score: 6, actions: [#recordUsedCoupon]}
UC1-7e1a1     {score: 7, actions: [!CAS.signalError]}
UC1-9a1    {score: 9, actions: [#presentRebateForm_RebateReceipt]}
UC1-9b1    {score: -1000, actions: [No "terminate" verb found]}
```

Appendix C: CSPS

In this appendix, we show the generated actions to use case set of CSPS described in Rational Unified Process [21]. We do not publish these use cases' text or the parse trees because RUP [21] is a commercial product and we have no permission to publish it.

C.1. Generated Actions

This section publishes actions generated from the processed use cases by the algorithm proposed in this thesis. The format of the created actions is described in Sect. 3.3. The original use cases are not numbered so we have numbered them in order to be published in RUP [21].

UC1-1.	{score: 5, actions: [#placeStory]}
UC1-2.	{score: 5, actions: [?ED.viewStory]}
UC1-3.	{score: 5, actions: [?ED.categorizeStory]}
UC1-4.	{score: 5, actions: [?ED.markStory]}
UC1-5.	{score: 5, actions: [#includeStory]}
UC1-6.	{score: 5, actions: [#triggerInitiation]}
UC1-2a1	{score: 6, actions: [?ED.selectModifyStory]}
UC1-2a2	{score: 5, actions: [#displayTitle]}
UC1-2a3	{score: 6, actions: [?ED.selectSpecificTitle]}
UC1-2a4	{score: 5, actions: [#displayCharacteristic]}
UC1-2a5	{score: 5, actions: [?ED.updateCharacteristic]}
UC1-2a6	{score: 5, actions: [?ED.selectSave]}
UC1-2a7	{score: 5, actions: [#repostStory]}
UC1-2a8	{score: 5, actions: [#triggerInitiation]}
UC1-2b1	{score: 6, actions: [?ED.viewAdvertisingContent]}
UC1-2b2	{score: 3, actions: [?ED.mark]}
UC1-2b3	{score: 6, actions: [#includeAdvertisingContent]}
UC1-2b4	{score: 9, actions: [!AD.createPreliminaryBillingRecord]}
UC1-2b5	{score: 7, actions: [#markPreliminaryBillingRecord]}
UC1-2c1	{score: 6, actions: [?ED.viewAdvertisingContent]}

UC1-2c2 {score: 8, actions: [?ED.mark, ?ED.provideReason]}
UC1-2c3 {score: 7, actions: [!AD.notifyRejection]}
UC1-2c4 {score: 5, actions: [#deleteContent]}
UC1-2d1 {score: 6, actions: [(SBAR (IN If if) (S (NP (NN story story)) (VP (VP (AUX has has) (VP (AUX been been) (VP (VBN deleted delete) (PP (IN by by) (NP (NN editor editor)))))) (CC and and) (VP (AUX is is) (RB not not) (ADJP (RB currently currently) (JJ viewable viewable)))))), %TERMINATE]}
UC1-3a1 {score: 10, actions: [?ED.markStory, ?ED.describeReason]}
UC1-3a2 {score: 12, actions: [#notifyOriginator, !ED.includeReason]}
UC1-3a3 {score: 5, actions: [#deleteStory]}
UC2-1. {score: 6, actions: [?ED.selectEditProfile]}
UC2-2. {score: 5, actions: [#displayCategory]}
UC2-3. {score: 5, actions: [?ED.selectCategory]}
UC2-4. {score: -99, actions: [(SBAR (IN If if) (S (NP (QP (JJR more more) (IN than than) (CD one one)) (NN profile profile)) (VP (VBZ exists exist) (PP (IN for for) (NP (NN category category) (PUNC, , ,)))))), Illegal sentence found - do not use AUX verbs: be]}
UC2-5. {score: 6, actions: [?ED.selectSpecificProfile]}
UC2-6. {score: 5, actions: [#displayDetail]}
UC2-7. {score: 5, actions: [?ED.updateDetail]}
UC2-8. {score: 11, actions: [(SBAR (IN If if) (S (NP (NP (NN information information)) (VP (VBN changed change))) (VP (VBZ includes include) (NP (NN credit credit) (NN card card) (NN information information) (PUNC, , ,))))), !CCS.validateNewInformation]}
UC2-9. {score: 6, actions: [#updateSubscriberProfile]}
UC2-8a1 {score: -200, actions: [AUX verb: "be" was used, Illegal sentence found - do not use AUX verbs: be]}
UC3-1. {score: 6, actions: [?SUB.selectPayFee]}
UC3-2. {score: 8, actions: [!SUB.checkCurrentSubscriber]}
UC3-3. {score: 7, actions: [!SUB.checkFile]}
UC3-4. {score: 15, actions: [(SBAR (IN If if) (S (NP (NN user user)) (VP (VBZ declines decline) (NP (NP (JJ current current) (NN card card) (NN information information)) (PP (IN on on) (NP (NN file file))))))), !SUB.promptCreditCardNumber_ExpirationDate_Pin]}
UC3-5. {score: 6, actions: [#verifyExpirationDate]}
UC3-6. {score: 7, actions: [#sendCreditCardInfo]}
UC3-7. {score: 9, actions: [!SUB.indicateNewExpirationDate]}

UC3-2a1 {score: 8, actions: [#displayCurrentCreditCardInformation]}

UC3-2a2 {score: 8, actions: [?SUB.acceptInformation, ?SUB.update]}

UC3-2a3 {score: 7, actions: [#sendCreditCardInfo]}

UC3-2a4 {score: 7, actions: [!SUB.indicateNewExpirationDate]}

UC3-7a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: will]}

UC4-1. {score: 6, actions: [?AD.selectPrintReport]}

UC4-2. {score: 8, actions: [!AD.displayAdvertisingContent]}

UC4-3. {score: 5, actions: [?AD.selectPiece]}

UC4-4. {score: 7, actions: [!AD.displayList]}

UC4-5. {score: 5, actions: [?AD.selectReport]}

UC4-6. {score: 5, actions: [?AD.selectFormat]}

UC4-7. {score: 11, actions: [#createFirstReport, !AD.prompt]}

UC4-8. {score: -99, actions: [(SBAR (IN If if) (S (NP (NN advertiser advertiser)) (VP (VBZ selects select) (NP (NNP Save save) (PUNC, , ,))))), Illegal sentence found - do not use AUX verbs: be]}

UC4-9. {score: -99, actions: [(SBAR (IN If if) (S (NP (NN advertiser advertiser)) (VP (VBZ selects select) (NP (NNP View view) (PUNC, , ,))))), Illegal sentence found - do not use AUX verbs: be]}

UC4-10. {score: 12, actions: [(SBAR (IN If if) (S (NP (JJR more more) (NNS reports report)) (VP (VBP are be) (ADJP (JJ available available) (PP (IN in in) (NP (NN session session))))))), ?AD.selectReport, ?AD.terminateViewingSession]}

UC4-11. {score: 11, actions: [#createNextReport, !AD.prompt]}

UC4-12. {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

UC5-1. {score: 6, actions: [?AD.selectPostContent]}

UC5-2. {score: 7, actions: [#validateAccountBillingInformation]}

UC5-3. {score: 5, actions: [#promptName]}

UC5-4. {score: 6, actions: [?AD.uploadGifFormat]}

UC5-5. {score: 11, actions: [#storeGif, #recordLocationName]}

UC5-6. {score: 6, actions: [#acknowledgeSuccessfulSaving]}

UC5-7. {score: 6, actions: [#displayPotentialCategory]}

UC5-8. {score: 5, actions: [?AD.selectCategory]}

UC5-9. {score: 7, actions: [#displayPotentialFrequencyPrice]}

UC5-10. {score: 5, actions: [?AD.selectFrequency]}

UC5-11. {score: 7, actions: [#createPreliminaryBillingRecord]}

UC5-12. {score: 5, actions: [#placeContent]}

UC5-3a1 {score: -100, actions: [AUX verb: "be" was used]}

UC5-3a2 {score: 7, actions: [#presentCentralPhoneNumber]}

UC6-1. {score: 6, actions: [?USER.selectProvideFeedback]}

UC6-2. {score: 8, actions: [!USER.lookPhoneNumber]}

UC6-3. {score: 13, actions: [#displayPhoneNumber, !USER.giveOption]}

UC6-4. {score: 6, actions: [?USER.selectEmailOption]}

UC6-5. {score: 11, actions: [#lookEmailAddress, #passBrowser]}

UC6-6. {score: 7, actions: [#launchEmailBrowserClient]}

UC6-7. {score: 5, actions: [?USER.enterMessage]}

UC6-8. {score: -1000, actions: [No "terminate" verb found]}

UC7-1. {score: 6, actions: [#scanArchivedList]}

UC7-2. {score: 6, actions: [(SBAR (WHADVP (WRB When when)) (S (NP (NN story story)) (VP (VBZ is be) (ADVP (RB no no) (RB longer longer)) (VP (VBN referenced reference) (PP (IN by by) (NP (ADJP (JJ unread unread) (CC or or) (JJ archived archived)) (NN category category) (PUNC, , ,))))))), #deleteStory]}

UC7-3. {score: 13, actions: [#displayBannerAd_GeneralContentCategory_SpecificStory]}

UC7-4. {score: 5, actions: [?SUB.viewStory]}

UC7-5. {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

UC7-4a1 {score: 7, actions: [!SUB.provideOption]}

UC7-4a2 {score: 6, actions: [%GOTO 4]}

UC8-1. {score: 5, actions: [?CE.placeContent]}

UC8-2. {score: 6, actions: [#checkReferenceInformation]}

UC8-3. {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

UC8-4. {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

UC8-2a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

UC9-1. {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

UC9-2. {score: 5, actions: [#checkCategory]}

UC9-3. {score: 7, actions: [!SUB.checkSubscriber]}

UC9-4. {score: 6, actions: [#generateTextMessage]}

UC9-5. {score: 5, actions: [#constructSeries]}

UC9-6. {score: 8, actions: [!SUB.sendEmailMessage]}

UC9-4a1 {score: -100, actions: [Illegal sentence found - do not use AUX verbs: be]}

UC10-1. {score: 6, actions: [?PS.selectSubscribeOption]}

UC10-2.	{score: 13, actions: [#lookCurrentContractTerm_AvailableServiceOption]}
UC10-3.	{score: 9, actions: [#displayContractTerm_ServiceOption]}
UC10-4.	{score: 11, actions: [?PS.acknowledgeTerm, ?PS.selectServiceOption]}
UC10-5.	{score: 6, actions: [#recordServiceOption]}
UC10-6.	{score: 5, actions: [#displayCategory]}
UC10-7.	{score: 5, actions: [?PS.selectCategory]}
UC10-8.	{score: 5, actions: [#displayDetail]}
UC10-9.	{score: 5, actions: [?PS.updateDetail]}
UC10-10.	{score: 11, actions: [#validateDatum, #updateSubscriberProfile]}
UC10-4a1	{score: 5, actions: [%TERMINATE]}