Information and Software Technology 58 (2015) 110–122

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Automated events identification in use cases

J. Jurkiewicz *, J. Nawrocki

Poznan University of Technology, Institute of Computing Science, ul. Piotrowo 2, 60-965 Poznań, Poland

ARTICLE INFO

Article history:
Received 22 January 2014
Received in revised form 18 September 2014
Accepted 22 September 2014
Available online 6 October 2014

Keywords:
Requirements engineering
Use cases
Functional requirements
Natural language processing

ABSTRACT

Context: Use cases are a popular method of expressing functional requirements. One contains a main scenario and a set of extensions, each consisting of an event and an alternative sequence of activities. Events omitted in requirements specification can lead to rework. Unfortunately, as it follows from the previous research, manual identification of events is rather ineffective (less than 1/3 of events are identified) and it is slow.

Objective: The goal of this paper is to propose an automatic method of identification of events in use cases and evaluate its quality.

Method: Each step of a main scenario is analyzed by a sequence of NLP tools to identify its performer, activity type and information object. It has been observed that performer, activity type and some attributes of information objects determine types of events that can occur when that activity is performed. That empirical knowledge is represented as a set of axioms and two inference rules have been proposed which allow to identify types of possible events. For each event type an NLG pattern is proposed which allows to generate description of the event type in natural language. The proposed method was compared with two manual approaches to identification of events: *ad hoc* and HAZOP-based. Also a kind of Turing test was performed to evaluate linguistic quality of generated descriptions.

Results: Accuracy of the proposed method is about 80% (for manual approaches it is less than 1/3) and its speed is about 11 steps/minute (ad hoc approach is 4 times slower, and HAZOP-based approach is 20 times slower). Understandability of the generated event descriptions was not worse than understandability of the descriptions written by humans.

Conclusions: The proposed method could be used to enhance contemporary tools for managing use cases.

1. Introduction

Requirements highly influence different aspects of software development [23,24,29,31]. One of the approaches to specifying functional requirements is the concept of use case. It was proposed by Jacobson [20] and further developed by Cockburn [5,12] and other authors [19]. Roughly speaking, a use case is a description of user-system interaction expressed in natural language. The focal element of a use case is the main scenario consisting of a sequence of steps that describe the mentioned user-system interaction. While a step is being executed, some events might happen. Descriptions of those events along with alternative scenarios are also included in a use case.

The question of how to identify a (more or less) complete list of possible events arises. Research shows that non-identified events (some call them "missing logic and condition tests for error handling" [37]) are the most common cause of changes in requirements which result in requirements volatility and creeping scope. The importance of event-completeness has been recognized by many authors (although they do not use this specific term), including Carson [11], Cox et al. [13], Mar [28] and Wiegers [41]. The approach they recommend is based on identification of events performed by experts using brain storming or other "manual" techniques. There is also a method called H4U [22]. It is a form of a HAZOP-based review, which aims at identifying events in use cases. The main disadvantage of manual identification of events is the time and effort required by such a process. Thus, a question arises whether it is possible to identify events automatically and obtain good quality event descriptions in the sense of event-completeness and their readability.

The above question is also interesting from the artificial intelligence point of view. The following saying is attributed to Pablo Picasso: *Computers are useless. They can only give you answers* [2]. Identifying of events that might happen during execution of a

* Corresponding author.
E-mail addresses: jjurkiewicz@cs.put.poznan.pl (J. Jurkiewicz), jnawrocki@cs.put.poznan.pl (J. Nawrocki).

http://dx.doi.org/10.1016/j.infsof.2014.09.011
0950-5849/

use-case step resembles a dialog in which a human is saying a simple sentence (e.g. *This evening I am going to the opera house.*—that corresponds to a use-case step), and a computer is asking a question (e.g. *What if all the tickets are sold?*—that question includes the event *all the tickets are sold*). To ask such event-related questions requires the understanding of a sentence, i.e. some knowledge is necessary (in this example it is knowledge about *opera* and *tickets*), and the ability to infer about events with use of that knowledge.

In this paper a method of automatic identification of events in use cases is presented. First, in Section 2, the model of use-case based requirements specification is outlined. The central part of the method is the inference engine that was used for the generation of event names (event descriptions). It is discussed in Section 3. The mechanism for analysing a use-case step with a Natural Language Processing (NLP) tool is presented in Section 4. The events are generated using Natural Language Generation (NLG) techniques—see Section 5. Empirical evaluation of the proposed approach is discussed in Section 6. Related work is discussed in Section 7.

2. Use-case based requirements specification

In this paper it is assumed that a functional requirements specification document contains the following elements:

- Actors – descriptions of actors interacting with the system being built.
- Information Objects – presentation of domain objects the actors and the system operate on. Every information object can have a set of different properties assigned to it (the possible properties are described in Section 3).
- Use Cases – descriptions of user-system interactions in the form of scenarios performed in order to achieve user's goals. Every use case consists of a name describing a user's goal and a main scenario consisting of a sequence of steps. Steps in the main scenario may be interrupted by events described within extensions. Additionally, alternative scenarios describe how given events should be handled. An exemplary use case is presented in Fig. 1.

Use cases have been used in both research and industry since their introduction in 1985 [20]. Research shows that scenario-based notations (including use cases) are the most popular way of describing customer requirements [31]. Two main factors are responsible for their popularity: firstly, the clear focus on describing the goals of the system; secondly, natural language which the requirements are expressed in. The latter makes it easier for the customer and prospective end-users to understand the requirements document, however, it makes it hard to process the specification in an automatic way.

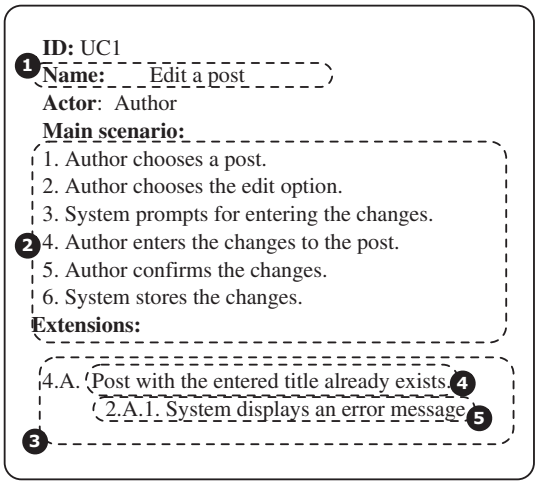

Fig. 1. Example of a textual use case. Main elements: 1 – name, 2 – main scenario section, 3 – extensions section, 4 – an event, and 5 – an alternative scenario.

3. Inference engine for identification of events

3.1. The problem

It is assumed that a functional requirements specification on the input contains descriptions of actors, descriptions of information objects, and use case stubs, i.e. use cases without their extension parts (see Fig. 1). To complete use case stubs with event descriptions the following problem is discussed in this paper:

Events identification problem: Given a step from the main scenario of a use case identify all the events that can happen when the step is executed. Only events which can be detected by the system are of interest (that viewpoint is supported by Adolph et al. [5] in their *Detectable conditions* pattern).

3.2. Abstract model of a use-case step

The process of event identification consists of four phases:

1. Preprocessing of the description of actors and information objects that creates a dictionary of these elements (it resembles the creation of a symbol table by a compiler during parsing).
2. Analysing a step using NLP techniques (a main scenario is analysed step by step).
3. Inferring the possible events for a given step (that results in abstract descriptions of events which are independent of natural language).
4. Generating event descriptions in natural language.

The overview of this process is presented in Fig. 2. The most important part of the presented method is inferring about the possible events.

Our approach to infer about events is based on a model of human–computer interaction, which evolved from our study of available use cases and events associated with them. Our aim was to provide the simplest possible model which would allow to infer about all the events we have found in the considered use cases. The model is twofold and is illustrated in Fig. 3.

Model of the real world (Fig. 3A). Some events are a consequence of a state in which a real object exists (e.g. a store is empty). The model represents real world objects (e.g. a book in a library, an article in a newspaper). What is interesting, for the purpose of inferring events, quite a simple tool proved to be powerful enough: it is the set theory. A fragment of the real world we are interested in can be viewed as individual objects (i.e. items) or collections of objects (i.e. sets). Furthermore, items might be simple or they might be compound and consist of other items. It is important to notice that some activities do not change the real world – they just provide us with information about the world (state) *as-is*. On the other hand, there are some activities which change the world (i.e. its state) and they lead to a new state (*as-to-be*). In our view, without awareness of the last category of activities, identifying some events would be impossible. Moreover, the state of a real world objects can change over time, e.g. a credit card can expire on some day.

Model of IT system (Fig. 3B). Some events are associated with human–computer communication or computer-computer interaction (e.g. an external service is not available). This

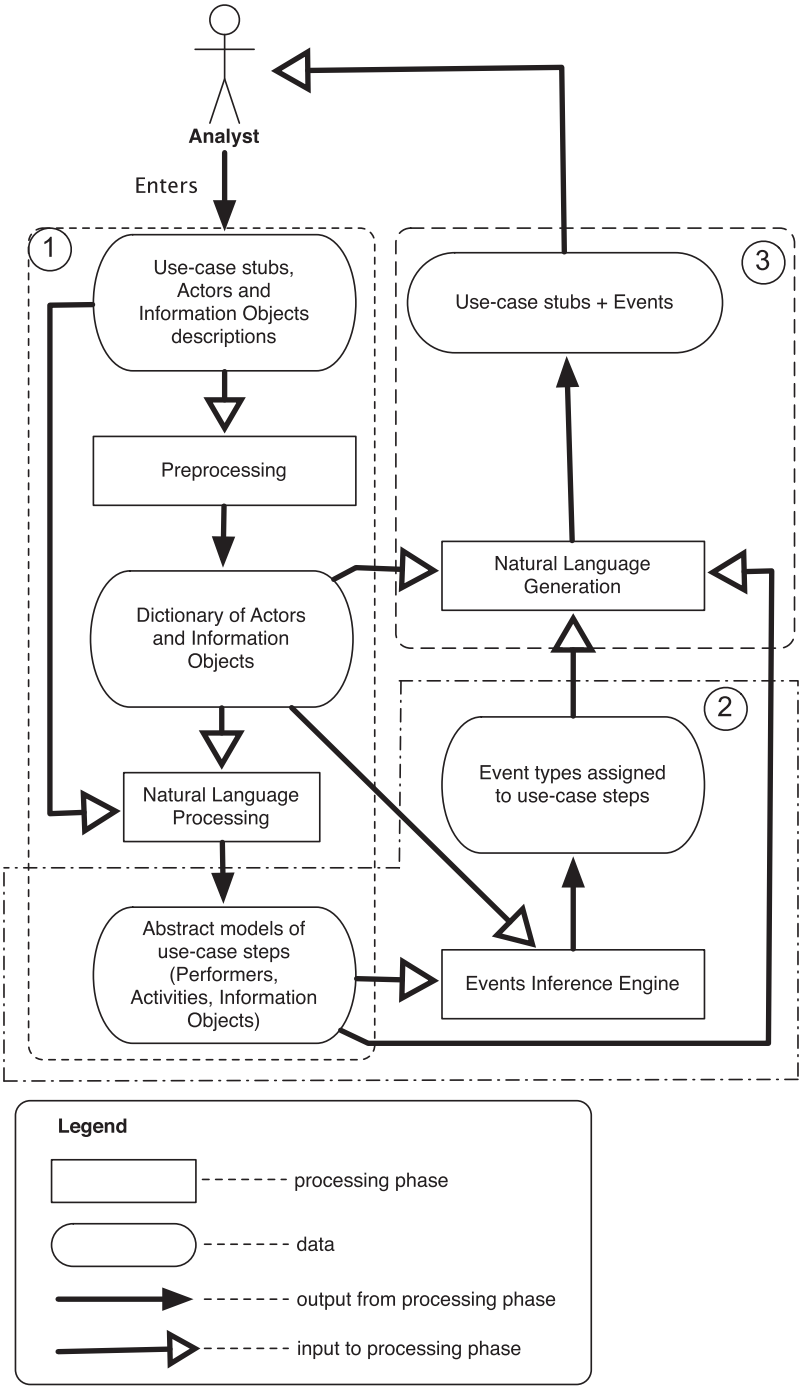

Fig. 2. An overview of use-case processing and events analysis: 1 – initial preprocessing of use case stubs, actors and information objects descriptions (using NLP tools); 2 – generation of abstract classes of events (events inference engine); and 3 – events names generation (using NLG tools).

model assumes that all the interactions between the Actor and the System happen through user interface (UI). The Actor can perform CRUD+SV operations (Create, Read, Update, Delete, Select, Validate) on data (both internal and external to the System). In the case of operations which can be harmful (e.g. deleting an information object), the Actor might need to confirm his operation, so Confirm has been added to our model. Moreover, the System may display data or may inform the Actor about the occurrence of a problem - that is reflected in our model via the Display operation. The behavior of the Actor and the System may depend on time, i.e. performing some operations might take too much time and result in a timeout or it might be too late for some operations to be completed. To some extent, the model of IT system resembles Albrecht's model [6] on which the function points method [7] is based. Unfortunately, Albrecht's model cannot be directly used for inference about events. For instance, some events are strongly associated with the delete operation which is not directly represented in the Albrecht's model. There is a similar problem with the confirm operation.

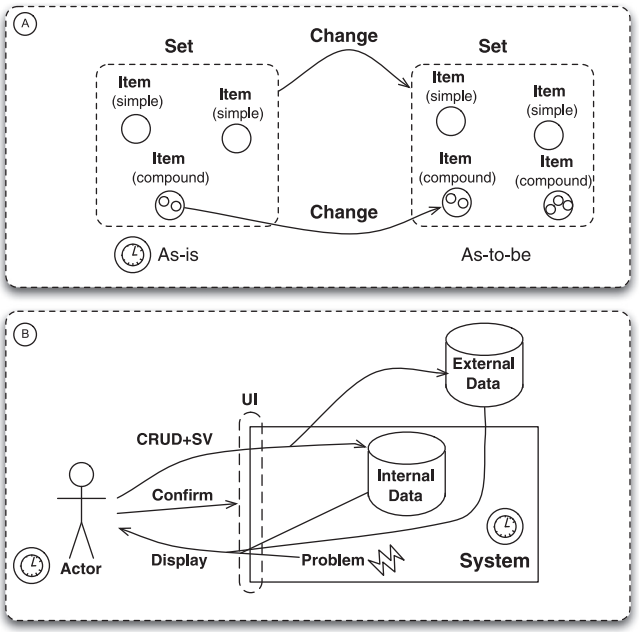

Fig. 3. (A) Model of real world, and (B) model of IT system.

It is assumed that every step of a use case can be represented as a triplet: Performer, Activity, Information Object [5,36]. Those three notions, along with the notion of Event, are described below.

3.2.1. Performer

Performer represents an abstract entity which performs an action in a given step. There are two types of performers: *SYSTEM* – represents the system being built; *USER* – represents one of the end-users of the system. In the example from Fig. 1 *Author* who is a performer in steps 1, 2, 4 and 5 has *USER* type. *System* which is a performer in steps 3 and 6 is of the *SYSTEM* type. Every performer of type *USER* needs to be declared by Analyst as an actor in the requirements specification.

3.2.2. Activity

Activity is a process triggered by Performer and it manipulates Information Objects (Information Objects are described below). To identify events that can appear when a step is executed (i.e. when the activity described by it is performed), we have decided to identify abstract types of activities. The types of activities were derived from the model of IT system presented in Fig. 3B.

Assumption 1. Each step of a use case can be classified as one of the activity types presented in Table 1.

Table 1
Identified types of activities of use-case steps.

Activity type	Exemplary use-case steps
ADD	Author adds new blog post
ENTER	Author enters title of the post
LINK	Author links the comment to the post
READ	Author browses available blog posts
UPDATE	Author changes the title of the post
DELETE	Author deletes the post
SELECT	Author selects a blog post from the available ones
VALIDATE	Author checks the blog post
CONFIRM	Author confirms the changes
DISPLAY	System presents available blog posts

JUSTIFICATION: Part of the proposed types of activities are directly related to the CRUD+SV operations from the model of IT system. Most of the CRUD+SV operations are simple from the point of view of the presented model of real world and each of them is assigned single activity type. For the sake of simplicity we name such activity types after operations corresponding to them. Here are activity types corresponding to simple operations: READ, UPDATE, DELETE, SELECT, VALIDATE.

The Create operation is not simple: it exhibits susceptibility to different events depending on an object it manipulates on. We have identified three activity types that can be assigned to this operation:

- ADD – when a new Information Object is added,
- ENTER – when data concerning part of Information Object is entered,
- LINK – when a connection between Information Objects is established.

Moreover, the confirm operation from the model of IT system is represented by the CONFIRM activity type and the display operation is represented by the DISPLAY activity type. □

3.2.3. Information Object

Information Object represents an object the performer operates on. The use case presented in Fig. 1 operates on a *post* which is an example of Information Object. Information Object is assigned a set of properties that are important from the point of view of event identification. Table 2 presents a set of properties we have found helpful when identifying events. The properties are divided into two sets called Activity-Free Properties (AFP) and Activity-Sensitive Properties (ASP). The former are valid for Information Object no matter which activity is performed on this object, e.g. *Article* always has the *SET* property, no matter what operation is performed on it. These properties correspond to the elements of the models presented in Fig. 3A: Set and Compound Items. Moreover, there might be some cardinality constraints imposed on the object, hence, two additional properties have been added: AT_LEAST (the smallest number or elements) and NO_MORE_THAN (the highest number of elements). Furthermore, some objects can be stored in an external system. This kind of objects should be marked with the REMOTE property.

ASP properties depend on Information Object and an activity performed on it. ASP properties are related to the time aspect of operations of the model of IT system. Assume an invoice can be exported and also some other operation can be performed on it. If only the *export* operation can take too long and can lead to specific events, then an information object *Invoice* will have the *LONG_LASTING* property only for the *export* operation.

3.3. Events

Assumption 2. Each event which can occur in a use case can be classified as one of the event types presented in Table 3.

JUSTIFICATION: To identify event types one can use a HAZOP-like brainstorming session (as described in [22]). As primary keywords one can choose activity types of Table 1. The CRUD+SV activities should be considered in the following six variants:

$$\{\text{Internal data}, \text{External data}\} \times \{\text{Simple}, \text{Compound}, \text{Set}\}$$

The secondary keywords can be the same as proposed in the HAZOP method (NO, MORE, etc. – see [22]). One can go through the list of primary keywords (i.e. all the variants of activity types), one by one, and for each one consider all the secondary keywords

Table 2
Identified properties of information objects together with their descriptions and examples.

Property	Description	Example
Activity-Free Properties (AFP)		
SET	Represents objects which form a set (they have to be unique)	Article
COMPOUND	Represents objects which consist of other objects	Credit Card (consists of holder name, number, expiration date, security code)
AT_LEAST	Represents objects which quantity must be at least X	Participants (number of which must be at least 10)
NO_MORE_THAN	Represents objects which quantity must be no more than X	Participants (number of which cannot be larger than 20)
REMOTE	Represents external objects	Credit Card (in order to process information concerning Credit Card System needs to contact an external payment service)
Activity-Sensitive Properties (ASP)		
LONG_LASTING for activity X	Activity X for this kind of objects can take a significant amount of time	Invoice (export of which may take significant amount of time)
TIMEOUTABLE for activity X	Activity X for this kind of objects is not possible after some deadline	Admission (there might be a deadline for admissions to the university)

Table 3
Identified event types together with their descriptions and examples.

Type of event	Description	Example
WrongData	Provision of wrong/invalid data	ISBN number is invalid
TooLittleLeft	There is not enough of information object	Bank account is empty
Incomplete	Not all data has been provided	ISBN of a book has not been provided
AlreadyExist	Given object or data already exist	Book with the given ISBN already exists
ConnectionProblem	Problem while connecting with a remote system	Could not connect to the credit card operator
TooMuchSelected	Too much data has been provided	Too many subjects selected
NoObjectSelected	No data has been selected from the given options	No subject selected
AlreadySelected	Given data/option has already been selected	The subject has already been selected
TooLittleSelected	Too little items have been selected	Too few subjects selected
DoesNotExist	Given object or data does not exist	Given article does not exist
LackOfConfirmation	Operation has not been confirmed	Author did not confirm
NoDataDefined	There is no information object of a given type defined	There are no subjects defined
WantsToInterrupt	Performer wants to interrupt an operation	Author cancels export operation
TooLateForDoingThis	It is too late to perform a given operation	The deadline for adding new subjects has passed

to identify possible event types. A list of identified event types is maintained through the brainstorming session. After the brainstorming, that list is analyzed, item by item, to:

- remove repeated or undetectable event types,
- clarify their descriptions,
- find examples of a proposed event type in the available real-life use cases.

The results of such a process are presented in Table 3. What is interesting, four HAZOP secondary keywords (AS WELL AS, PART OF, BEFORE, AFTER) did not contribute to any of the identified event types (they seem useless in this process). □

The process of identification of events in use-case steps is based on inference rules formulated after the analysis of real-life use cases. The rules are expressed in a form similar to the propositional logic, i.e.:

$$\frac{premise}{conclusion}$$

By analysing available use cases we have discovered two inference rules called *simple* and *compound*. In order to keep the rules brief and clear the following symbols are used: R for role (i.e. performer type), A for activity type, O for information object, P for property of information object, E for event type.

3.3.1. Simple inference rule of events identification

The following notions are used by the simple reference rule:

- *Observed* is a finite set of axioms. Each axiom has the form of a quadruple $\langle R, A, P, E\rangle$ and it denotes an observation that an activity of type A performed by a performer of type R on an information object with property P can lead to an occurrence of event of type E. The observed set of axioms we have identified so far is presented in Table 4. Obviously, this set can be easily extended by adding new axioms resulting from further research.
- *AFP* (stands for Activity-Free Property) is a function. Its domain is a finite set of information objects described in a requirements specification document (see Section 2). This function returns a finite set of properties of an information object.

It is assumed that every use-case step, originally written in natural language, can be represented by the following triple: performer type (R), activity type (A) and information object (O). It means that each step is a simple sentence in the form of Subject-Verb-Object. This assumption is supported e.g. by Cockburn's guideline [12] which says that every step should be simple and consist of subject, verb and object. Therefore, a use-case step is represented as $S = \langle R, A, O\rangle$. Transformation of an original step to its abstract representation $\langle R, A, O\rangle$ is done with the help of NLP tools (as described in Section 4).

The simple inference rule assumes that properties of information objects depend solely on information object, not on an activity. As events can appear but do not have to, the rule uses the ♢

Table 4
Elements of the *Observed* set; the star means that the row is valid for any value of the properties of the information object.

R (Performer)	*A* (Activity)	*P* (Information Object Property)	*E* (Event type)
USER	ENTER	SET	WrongData
USER	ENTER	COMPOUND	Incomplete
USER	ENTER	SET	AlreadyExists
USER	ENTER	REMOTE	ConnectionProblem
USER	ENTER	AT_LEAST	TooLittleLeft
SYSTEM	DISPLAY	SET	NoDataDefined
USER	SELECT	SET	NoDataDefined
USER	SELECT	SET	AlreadySelected
USER	SELECT	NO_MORE_THAN	TooMuchSelected
USER	SELECT	*	NoObjectSelected
USER	DELETE	*	DoesNotExist
USER	READ	*	DoesNotExist
USER	LINK	*	Incomplete
USER	SELECT	AT_LEAST	TooLittleSelected
USER	CONFIRM	*	LackOfConfirmation
SYSTEM	ADD ENTER LINK READ UPDATE DELETE SELECT VALIDATE CONFIRM DISPLAY	LONG_LASTING	WantsToInterrupt
USER	ADD ENTER LINK READ UPDATE DELETE SELECT VALIDATE CONFIRM DISPLAY	TIMEOUTABLE	TooLateForDoingThis

symbol from modal logic to denote this. The rule is presented below:

$$\frac{S = \langle R, A, O\rangle, P \in AFP(O), \langle R, A, P, E\rangle \in Observed}{S \models \Diamond E} \quad (1)$$

An example of applying this rule is presented in Fig. 4.

3.3.2. Compound inference rule of use-case events

Unfortunately, in some cases the simple inference rule is too weak. Some events depend not only on an information object but also on an activity performed on it. To cope with this, the compound inference rule has been introduced. This rule uses the notion of activity-sensitive properties (*ASP* for short). It is a function whose domain is a finite set of information objects (as in the case of *AFP*). The function returns a finite set of pairs $\langle P, A\rangle$, where P denotes a property of information object O and A denotes an activity type. Property P of object O can exhibit itself only when activity A is performed.

The compound inference rule is presented below:

$$\frac{S = \langle R, A, O\rangle, \langle P, A\rangle \in ASP(O), \langle R, A, P, E\rangle \in Observed}{S \models \Diamond E} \quad (2)$$

To show how the above rule can be used let us extend the example from Fig. 4. Assume that the students have to submit their posts by a certain deadline. In this case, knowing this, Analyst should classify *post* as an information object which is TIMEOUTABLE for the *write in* activity (*write in* corresponds to the ENTER activity type). As a result of parsing the declaration of information objects, the *ASP* function is augmented with the following item:

$\langle Post, ENTER, TIMEOUTABLE\rangle$.

When it comes to parsing the step *Student writes in a post*, it is translated to the following triple

$\langle USER, ENTER, Post\rangle$.

Since the *Observed* set contains the quadruple

$\langle USER, ENTER, TIMEOUTABLE, TooLateForDoingThis\rangle$

and *ASP(Post)* contains

$\langle TIMEOUTABLE, ENTER\rangle$

the following conclusion in inferred: the event *TooLateForDoingThis* can happen when executing the step.

4. Natural language processing of use cases

The presented inference engine works with the elements of abstract model of use-case steps: performer type, activity type and information object. Before the engine can be used every use-case needs to be analysed with NLP tools and the elements of the mentioned model have to be identified.

4.1. Preprocessing

UC Workbench [30] is a tool used for use-case management. With this tool Analyst can enter and manage his/her use case-based requirements specification. The entered descriptions of actors, and information objects are automatically transformed into a dictionary of actors and information objects. (Unfortunately, UC Workbench as described in [30] did not facilitate automatic analysis of use cases supported by NLP techniques.)

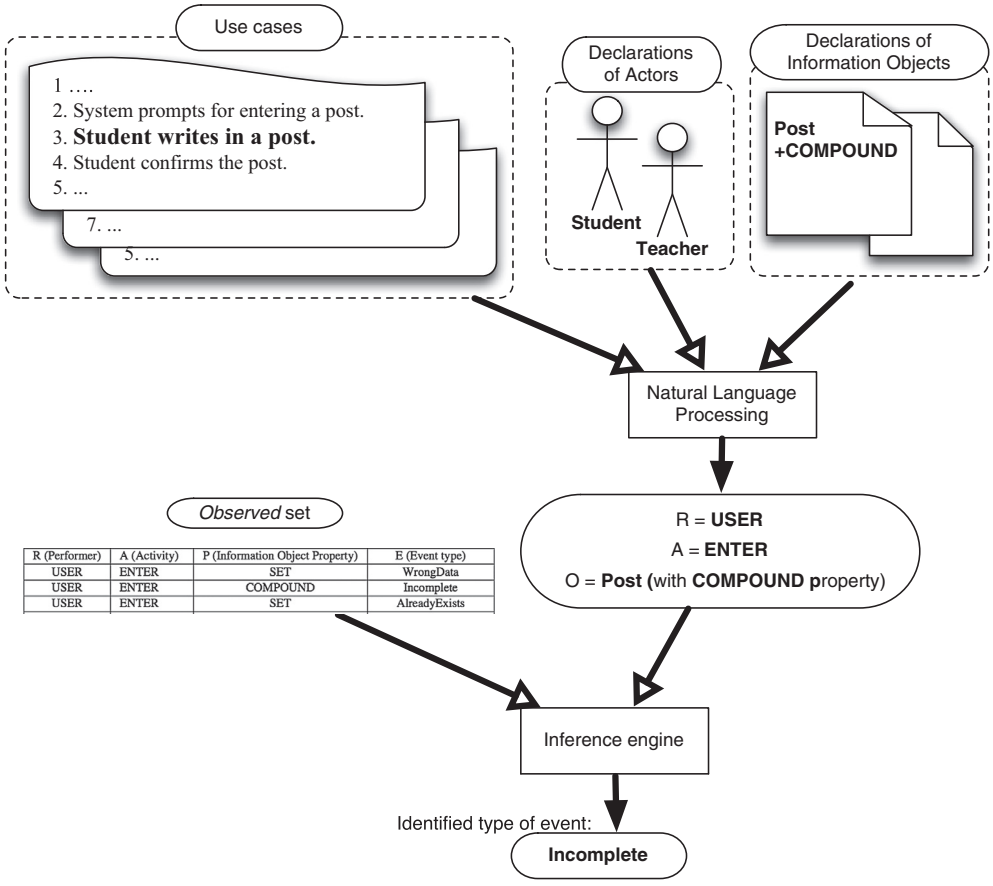

Fig. 4. An example of inferring events for step *Student writes in a post*.

4.2. NLP

Additional plugins for UC Workbench performing natural language analysis were developed in order to analyse the entered use cases. These plugins are organized according to the Chain of Responsibility design pattern [15]. It means that the text of a use-case step goes through all of the tools defined in the chain. Moreover, an output of one tool can be used as an input by another tool. Tools providing the following operations are used in order to analyse a use-case step: sentence segmentation [10], word tagging [4], part-of-speech tagging [3], lemmatization [21], dependency relations tagging [14]. The chain of the used NLP tools is presented in Fig. 5. After the NLP processing, each step is annotated with the elements of the step abstract model: performer type, activity type, information object.

The presented NLP layer is language independent, meaning that this approach could be used for any natural language providing that adequate tools are available for the given language. In our research use cases were written in English and they were analysed with the tools contained in the Stanford Parser kit [14] and Open NLP project [1]. However, any other tool realizing the above mentioned NLP operations could be used, e.g. a French parser [17] or German [35] one.

5. Generation of event descriptions

The inference engine described in Section 3 generates abstract types of events (depicted in Table 3). Those event types (e.g. *TooLittleLeft*) are like row material and need further processing. An analyst needs to obtain a description of event which is expressed in natural language and which is easily understood by the potential readers. Therefore, Natural Language Generation (NLG) was used as the final stage of events identification.

For every abstract event type a template for generation of event description has been proposed. The templates together with examples are presented in Table 5. A template is a sequence of constant strings of characters and items from use-case steps (available after NLP phase), i.e. subject, verb, object. Sometimes those use-case items are modified to present them in appropriate tense (e.g. verb can be presented in the simple past) or grammatical number (e.g. object can be used in plural).

For the task of NLG, a Java library called SimpleNLG [16], which provides a data-to-text mechanism, was used. SimpleNLG offers an API which allows for setting a semantic input for a phrase and direct control over the realization process of phrase generation (by setting various values of phrase features, e.g. tense, voice).

Based on this, a prototype tool combining all three stages of event generation (NLP analysis, inference engine, NLG) was implemented. The tool was built as an additional plugin for UC Workbench. Section 6 presents empirical evaluation of the accuracy of the prototype tool and the understandability of event descriptions generated by the tool.

6. Empirical evaluation

As mentioned in Section 5, a prototype tool combining NLP, inference engine, and NLG was implemented as part of the UC Workbench project. This prototype has been evaluated from the three perspectives: accuracy of events identification (together with precision), speed of analysis of use case steps, and readability of the generated event descriptions.

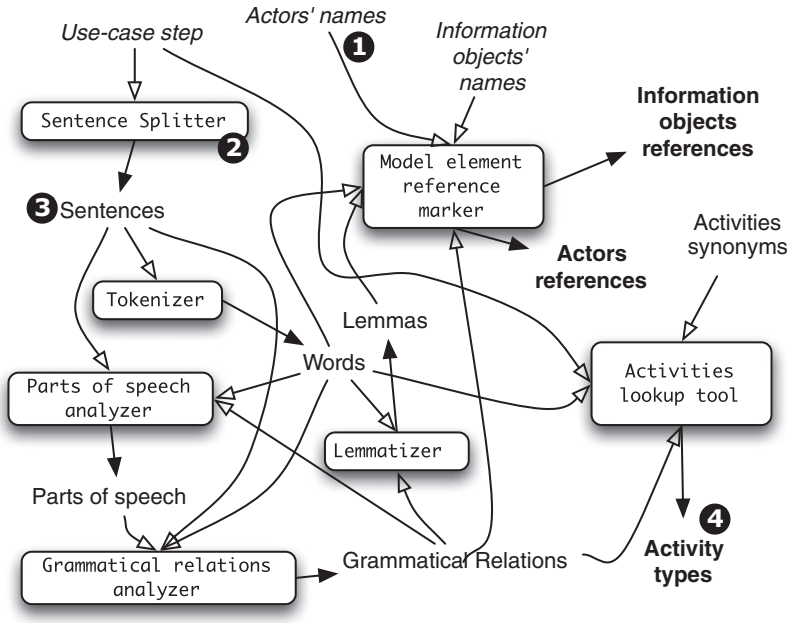

Fig. 5. The chain of the used NLP tools: 1 – input provided by the analyst in a natural language (italic font), 2 – tool, 3 – output from one tool and input for another tool, and 4 – output from the chain (bold font).

Table 5
Information used for generation of descriptions of events. A template for every event type is presented, together with the exemplary data used for the generation of event description.

Event type	Use-case step	Subject	Verb	Object	Event template	Generated description
WrongData	Customer enters credit card details	Customer	Enter	Credit card	<subject> <verb [past tense]> wrong data of <object>	Customer entered wrong data of credit card
TooLittleLeft	Customer enters number of books	Customer	Enter	Book	Not enough <object [plural]> left	Not enough books left
Incomplete	Author writes in an article	Author	Write in	Article	<subject> <verb [past tense, active voice]>, incomplete data of <object [singular]>	Author wrote in incomplete data of article
AlreadyExist	Author writes in an article	Author	Write in	Article	<object>already exists	Article already exists
ConnectionProblem	Customer enters credit card details	Customer	Enter	Credit card	Connection problem occurred	Connection problem occurred
TooMuchSelected	Student selects books	Student	Select	Book	Too many <object> <verb [past tense, passive voice]> by <subject>	Too many books selected by Student
NoObjectSelected	Student chooses a book	Student	Choose	Book	No <object> <verb [past tense, passive voice]> by <subject>	No book chosen by Student
AlreadySelected	Student selects a subject	Student	Select	Subject	<object> has already been <verb [past participle]> by <subject>	Subject has already been selected by Student
TooLittleSelected	Student chooses subjects.	Student	Choose	Subject	Too little <object [plural]> <verb [past tense, passive voice]> by <subject>	Too little subjects chosen by Student
DoesNotExist	Author deletes an article	Author	delete	Article	Given <object> does not exist	Given article does not exist
LackOfConfirmation	Author confirms the operation	Author	Confirm	Operation	<subject> <verb [present perfect tense, negative]>	Author has not confirmed
NoDataDefined	System displays available articles	System	Display	Article	No <object> has been defined	No article has been defined
WantsToInterrupt	System imports invoice data	System	Import	Invoice	Operation canceled	Operation canceled
TooLateForDoingThis	Author enters an article	Author	Enter	Article	Too late for this operation	Too late for this operation

6.1. Automatic vs. manual identification

6.1.1. Experiment definition

The object of the study was automatic events identification. The purpose was to evaluate it with the focus on accuracy, precision and speed. The experiment was conducted to take into account the point of view of the project manager. The context comprised both industrial and academic projects.

6.1.2. Reference set of events

To evaluate the prototype we have used the experimental data concerning accuracy of events identification performed by humans that have been acquired through two earlier experiments [22]: one with 18 students of the Software Engineering Programme at Poznan University of Technology, second with 64 IT professionals (see [22] for more details). The designs of both experiments were the same. The aim of the participants of both experiments was to

identify as many events as possible for the main scenarios from the benchmark use-case-based specification [8]. The identified events in the experiments were analysed by two experts and a set of sensible events for steps from the benchmark specification was distinguished. This set was used as a reference set of events for the evaluation of the prototype tool.

6.1.3. Sensible events

Use-case steps are written in natural language, therefore, their interpretation can differ from reader to reader. From the perspective of use-case events this means that different people might identify different events for a given use-case step. Some of the identified events, although being correct in terms of language correctness, might not make sense in the context of the use case. For example for the step number 3 from Fig. 1 the following event could be identified: *Author answers a phone call*. This event is a correct event from the language perspective, however, it does not make sense for this step. The events which make sense in the context of a given use-case step are desirable and we will call them *sensible*.

6.1.4. Measures

The following measures were used in order to evaluate the prototype:

- *Accuracy* (of events identification) – proportion between the number of generated sensible event descriptions and the total number of events from the reference set of events.[1] It was measured using the following formula:

$$\text{Accuracy} = \frac{\sum_{s=1}^{n} \#events(s)}{\sum_{s=1}^{n} \#Events(s)} \tag{3}$$

  where
  - n is the total number of steps in the benchmark specification;
  - $\#events(s)$ is the number of distinct sensible events identified by the prototype tool for step s;
  - $\#Events(s)$ is the number of distinct sensible events identified by all participants of the experiments for step s.
- *Precision* – proportion of the identified events that were sensible. It was measured using the following formula:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

  The symbols TP, FP are defined by the confusion matrix presented in Fig. 6.
- *Speed* – average number of steps analyzed per minute:

$$Speed(p) = \frac{\#steps}{T}[steps/minute] \tag{5}$$

  where
  - $\#steps$ is the total number of steps analysed by the prototype tool.
  - T is the total time (in minutes) spend by the tool on analysing the steps and generating event descriptions.

		Expert answer	
		y	n
Tool answer	Y	TP	FP
	N	FN	TN

Fig. 6. Confusion matrix. *T (true)* – tool's output is consistent with the expert opinion. *F (false)* – tool's output is inconsistent with the expert opinion. *P (positive)* – positive output of the tool (event was identified). *N (negative)* – negative output of the tool (event was not identified).

6.1.5. Procedure

The prototype tool was run for the whole benchmark specification [8] and the processing time (for the whole benchmark) was measured.[2] After the analysis had been finished the generated event descriptions were evaluated by an expert. For the sake of conformity with the experiments concerning manual event identification [22], the similar procedure was used. The procedure was as follows:

- The generated events were assigned to different abstract classes of events.
- The identified events were divided into sensible and insensible ones and the latter were omitted.
- The measures based on the confusion matrix (presented in Fig. 6) were calculated.

6.1.6. Results

Table 6 presents the results for the prototype tool and for two other methods used for events identification in the experiments described in [22]. H4U was a HAZOP-like method described in [22], and *Ad hoc* was a simple approach for events identification in which the participants of the experiments did not use any particular method for events identification. Precision was not calculated for H4U nor for *ad hoc*, hence, these results are not presented.

The initial accuracy of the prototype tool was equal to 0.73, however, after the analysis of the results it was found that the initial *Observed* set used in the inference engine (see Section 3) could be extended by adding new elements to it. After adding four new elements to the *Observed* set (these elements are presented in Table 7) the accuracy rose to 0.89.

6.1.7. Threats to validity

Threads to internal validity of the performed experiment seem negligible. The observed differences are significant and we do not see any explanation for the observed differences other than the influence of automatization.

From the point of view of external validity two issues seem most important: (1) How representative are the use cases which have been used in the experiment? (2) To what extent do the participants represent the target population (analysts)? As regards representativeness of the use cases, they are part of a benchmark specification which represents a typical requirements specification [8]. The benchmark resulted from analysis of 524 use cases and 16 projects. Thus, one can assume that this threat is negligible. As regards the participants, we have used the data from another experiment [22]. 18 of these participants were students (who had previous experience in software houses) and 62 were IT professionals. Due to such a strong representation of IT professionals and experience of the students one can say that the participants represent the target population, (i.e. analysts) pretty well.

[1] This definition differs from the definition used in the domain of information retrieval. However, we wanted to be consistent with the definitions and terminology used for evaluation of *ad hoc* and HAZOP-based methods [22]. Some people might call it *recall*.

[2] The benchmark was processed 10 times; the presented times are the average. The following hardware was used to perform the measurements: MacBook Air, CPU: 1.8 GHz Intel Core i7, RAM: 4 GB 1333 MHz DDR3, HDD: 256 GB SSD.

Table 6
Results for the prototype tool, the H4U method, and for the *ad hoc* approach.

	The prototype tool	H4U Max	H4U Avg	*Ad hoc* Max	*Ad hoc* Avg
Accuracy	(0.73) 0.89	0.44	0.26	0.28	0.18
Precision	0.93	NA	NA	NA	NA
Speed	10.8	2.32	0.85	2.65	2.58

Table 7
Elements added to the *Observed* set based on analysis of the results of the experiment.

R (Performer)	*A* (Activity)	*P* (Information Object Property)	*E* (Event type)
SYSTEM	FETCH	REMOTE	ConnectionProblem
USER	UPDATE	SET	WrongData
USER	ENTER	COMPOUND	Incomplete
USER	DELETE	AT_LEAST	TooLittleLeft

6.1.8. Interpretation

The presented results for the implemented prototype and two manual approaches to events identification show that the accuracy and speed of events identification can be highly improved by using an automated tool.

The tool was not able to generate all the possible sensible events because some steps did not contain direct references to the information objects (words like *information* or *data* were used instead). Another reason was related to the events concerning the domain of the benchmark specification (i.e. the rules of admission to university). These events could only be identified by domain experts. For example, the step *Student links subjects to the majors* can be interrupted by the event *Too little humanities subjects were chosen*, which is a business rule known only by a domain expert. If those rules were expressed in a formal way (e.g. as a set of criteria defining an acceptable curriculum) then such events could probably be also inferred in an automatic way. However, additional research is required to check the feasibility of such an approach.

We have examined the proposed approach taking into account two criteria: accuracy and precision. Accuracy seems obvious as it describes the ability of the proposed method/tool to identify a possible event (the higher the accuracy the lower the percentage of missed events). Accuracy is of primary importance to the users (Analysts) as the users, when provided with such a tool can rely heavily on it. As a consequence they can be "blind" and unable to identify any additional event. A solution could be a reverse order of activities, i.e. first a human could be asked to identify events and then the system could check the completeness of the given list of events – this would require a mechanism for checking the semantic equivalence of two descriptions of an event.

Accuracy, though very important, is not sufficient. In the case of automatic tool there is a danger that in the pursuit of a high level of accuracy the system will produce such a large number of invalid events that it would damage the usability of the proposed solution (those superfluous events resemble spam and they must be removed by Analyst by hand after careful filtering them out from the correct events – obviously this can be time consuming). Because of this we have complemented accuracy with precision. The latter describes the percentage of superfluous events (the higher the precision the lower the number of superfluous events). From the practical point of view, precision is much less important than accuracy (recall), because it is easy for a human to identify a false positive (i.e. an event description that makes no sense). If the tool misses even a small fraction of events (say, 5%), finding those missed events requires searching the whole set of use cases manually. The proposed method is not able to limit the manual search to particular parts of the input document (manual identification of events has the same drawback). Since identification of some events requires domain knowledge and human intelligence, it seems sensible to complete the work done by a computer with a quality check performed by a human, i.e. the output of the proposed tool should undergo a review (e.g. in a form of inspection, which is quite a popular practice in software companies). If the review was supported by a checklist, one of the items on the checklist could concern event completeness. Obviously, reviewing can only improve accuracy, but it is not able to guarantee that all the possible events will be identified. From the performed experiments it follows that both accuracy and precision of the proposed method are quite high (about 90% – see Table 6), much higher than can be expected from a human.

Speed is another advantage of the tool. It was able to generate events for the whole benchmark specification (consisting of over 150 use-case steps) in a time of under 20 min, while only five participants of the experiments were able to analyse the whole specification in 60 min (more precisely, as it follows from Table 6, the prototype tool needed, on average, about 6 s to analyze a use-case step—if the speed doubled, the proposed method could be used as part of an interactive use-case editor).

6.2. Turing test

Every generated event description was analysed by an expert in order to evaluate if the given event description made sense. However, a question arises if the generated event descriptions are easy to read and understand by people other than the experts involved in the experiment. In order to evaluate the readability of the generated event descriptions an experiment, based on the assumptions of Turing test [39], was conducted.

6.2.1. Experiment definition

The object of the study was automatic events identification. The purpose was to evaluate it with a focus on readability. The experiment was conducted to take into account the point of view of analyst, in the context of academic settings.

6.2.2. Participants

The experiment was conducted with 20 students of the Software Engineering Programme at Poznan University of Technology. The experiment was performed in two parts, first with 12 students and later it was repeated with 8 other students. The students were familiar with the concept of use cases and use-case events. All of the classes of the Software Engineering Programme are conducted in English, hence it was assumed that the students can understand text written in English.

Use-case step:	Candidate chooses a major that he/she wants to pay for.
Event A:	No major was chosen by Candidate.
Event B:	Candidate didn't choose major.

Fig. 7. An *event case* used in the Turing test.

6.2.3. Experimental material

The material used in the experiment consisted of two parts: a sheet of events and a questionnaire. The sheet of events contained a set of 20 'event cases'. Each event case presented two alternative descriptions (A and B) of the same use-case event. The descriptions were written in English. One of the descriptions (e.g. A) was generated by the tool and the other (e.g. B) was provided by a human during the previously mentioned experiment [22]. In the case of a human-provided description, the best (from the linguistic point of view) description was chosen, based on expert evaluation. The order of the event descriptions in the pairs was random, i.e. sometimes the automatically generated description was option A, sometimes B. Fig. 7 presents an example of a pair of event descriptions used in the experiment. The task of each subject was to decide which of the alternative event descriptions is more readable. The participants could choose one of the following options:

- description A is much better than B
- description A is slightly better than B
- it is hard to say which description is better
- description B is slightly better than A
- description B is much better than A

The second part of the experimental material was a questionnaire. It was prepared as a measurement instrument for controlling possible influence factors such as experience with use cases, and general impression about understandability of the presented descriptions.

6.2.4. Procedure

Before the main experiment a preliminary experiment (with 3 participants) was conducted in order to evaluate the proposed experimental material and the time required to perform the task. Some of the comments from the participants of this preliminary experiment were incorporated into the final version of the materials. The results of this preliminary experiment were not taken into account in the final results. The main experiment started with a 5-min introduction. The experimenter demonstrated the sheet of events and described the task. The participants were not informed that half of the event descriptions were generated by the prototype tool. The participants were asked to read the sheet of events and perform the task. They were also asked to note down the start and finish times. The participants were given 30 min to finish the task. After finishing the task the participants were asked to fill in the questionnaire. At the end all the materials were collected by the experimenter. The experiment took, in total, 35 min.

6.2.5. Data validation

After collecting the experimental material, all questionnaires were reviewed. None was rejected due to incompleteness or errors.

6.2.6. Results

For the purpose of evaluation the 5-point Likert scale (see Section 6.2.3) was transformed into 2 values:

- *success*: an event description generated by the prototype tool was considered much better, slightly better or indistinguishable from its counterpart provided by a human;
- *failure*: an event description generated by the prototype tool was considered much worse or slightly worse than its counterpart.

For every pair of event descriptions, the number of *successes* and *failures* were calculated. Descriptive statistics of these results are presented in Table 8.

6.2.7. Hypotheses testing

In order to asses readability of the event descriptions generated by the prototype tool, the following conjecture has been formulated.

Conjecture 1. Automatically generated descriptions are not worse than their manual counterparts, in terms of readability.

Justification. To justify this conjecture one has to compare *successes* against *failures*. In total, there were 243 successes and 157 failures, thus the probability of success was about 60%. To check if that result is significant let us use binomial hypothesis testing. Assume the null hypothesis states that the participants in the experiment randomly assessed the pairs of event descriptions, hence the probability of both *success* and *failure* is 0.5.

$$H_0 : p = 0.5 \tag{6}$$

The alternative hypothesis states that the probability of *success* was greater than 0.5.

$$H_1 : p > 0.5 \tag{7}$$

To test the hypotheses the binomial test was used, with the significance level α set to 0.05. As the result of the testing procedure, the null hypothesis was rejected (p value was equal to 1.996e−05). □

6.2.8. Threats to validity

The threats to internal validity of the performed experiment seem negligible. The observed differences are significant and we

Table 8
Descriptive statistics for the Turing test.

Event-case no	Success	Failure
#1	17 (85%)	3 (15%)
#2	11 (55%)	9 (45%)
#3	9 (45%)	11 (55%)
#4	16 (80%)	4 (20%)
#5	8 (40%)	12 (60%)
#6	5 (25%)	15 (75%)
#7	8 (40%)	12 (60%)
#8	11 (55%)	9 (45%)
#9	15 (75%)	5 (25%)
#10	10 (50%)	10 (50%)
#11	5 (25%)	15 (75%)
#12	19 (95%)	1 (5%)
#13	18 (90%)	2 (10%)
#14	7 (35%)	13 (65%)
#15	13 (65%)	7 (35%)
#16	12 (60%)	8 (40%)
#17	8 (40%)	12 (60%)
#18	18 (90%)	2 (10%)
#19	19 (95%)	1 (5%)
#20	14 (70%)	6 (30%)
N	20	20
Min	5	1
1st Qu.	8	3.75
Median	11.5	8.5
Mean	12.15	7.85
3rd Qu.	16.25	12
Max	19	15

do not see any explanation for the observed differences other than the difference in the source of the event descriptions (the prototype tool or the participants).

There are some threats to external validity that need to be taken into account when interpreting the results of the experiment. First, it needs to be emphasized that only 20 people participated in the experiment, which constitutes a small sample. Second, the event descriptions identified by humans, which have been used in the experiment, were written by students and IT professionals who were not native speakers (their native language was Polish) hence, the proposed event descriptions might not have been fully correct from the linguistic point of view. However, in order to mitigate this threat only the best (from the linguistic point of view) event descriptions werused in the experiment.

6.2.9. Interpretation

Based on the results of the experiment one can conclude that the event descriptions generated by the prototype tool were not worse, in terms of readability, than their counterparts provided by humans. Thus, it is possible to automatically generate event descriptions that might be included into a real life use-case based specification.

7. Related work

A number of authors, including Cox et al. [13], Mar [28], and Wiegers [41], advocate focusing on specification of events and exceptional scenarios. Alexander [9] underlines that "exception cases can often outnumber normal cases and can drive project costs". Experiments conducted by him show that using scenarios helps in identification of exceptions and events, however, he does not propose any particular method for events identification. Cox et al. [13] propose the following question in the checklist for improving requirements quality: "Alternatives and exceptions should make sense and should be complete. Are they?". Nevertheless, they do not present any technique which could help in achieving completeness of events. Similarly, Wiegers [41] suggests the following question in his inspection checklist "Are all known exception conditions documented?"

Letier et al. [26] presented a technique for discovering unforeseen scenarios which might have harmful consequences if they are not handled by a system. Their method is focused on looking for implied scenarios and can help in the elaboration of additional system requirements. The presented cases studies show that the method can give good results, however, it cannot be used with use cases written in a natural language as it assumes Message Sequence Charts are used as a scenario-based specification language. Moreover, the prosed technique is manual and requires vast domain knowledge.

van Lamsweerde and Letier [40] propose an approach for discovering and handling obstacles in requirements. They claim that very often requirements specifications present ideal situations and omit abnormal situations which, if not handled properly, may lead to system failures or unexpected behaviors. Lamsweerde presents techniques which work on requirements defined as users' goals expressed in a formal language called KAOS [25]. The proposed techniques allow for systematic identification of obstacles from goal specifications. Due to the fact that the method is suited for KAOS-based requirements it allows for inference about possible obstacles. However, this strict level of formality makes the requirements hard to write and even harder to read for IT-laymen.

Makino and Ohnishi [27] propose a method for generation of exceptional scenarios. Their approach is based on the analysis of differences between slightly different normal (positive) scenarios. This method assumes that similar normal scenarios should share similar abnormal scenarios. Although this method can help in automatic analysis of exceptional scenarios, it works only for requirements written with a scenario language named SLAF [18], which assumes limited vocabulary and grammar.

Another interesting approach to events identification was taken by Sutcliffe et al. when they were developing their CREWS-SAVRE tool [38]. In general, the procedure was similar to the one used in this paper and it consisted of three phases: (1) analysis of a step, (2) inference, and (3) generation of output. However, each of the phases is different. In CREWS-SAVRE analysis of a step is strongly supported by analyst. He is to determine, in the domain modeler, for each step its Action Type (e.g. Communication, Cognitive, Physical, or System), Role Mappings (e.g. object *telephone* that appear in Step 10 is the same *telephone* in Step 20), Action Link Rules (e.g. THEN or MEANWHILE), etc. Annotating steps in that way is time consuming. In our approach use-case steps are analyzed in an automatic way. As regards inference, in both cases, i.e. CREWS-SAVRE and the tool described in this paper, it is automatic, but the results of inference are different. CREWS-SAVRE aims at identifying so-called generic requirements (e.g. *The system shall check for data entry mistakes* or *The system shall restrict possible data entry*). Those requirements are assigned to generic exceptions (e.g. *What if information suspect lie?*) and in CREWS-SAVRE the inference engine is to identify such exceptions. In our approach the inference engine is to generate events that would be included into the use case (e.g. *Credit card expired*). As a consequence, in each of the compared tools generation of output is different. For CREWS-SAVRE it is enough to select text of generic requirements from a database. In the case of automatic augmenting use-case scenarios with exceptions one needs NLG tools to translate abstract description of an identified event to a readable sentence in a natural language. Thus, for the approach described in this paper it makes sense to conduct Turing test. In the case of CREWS-SAVRE such a test would be superfluous.

The approach presented in this paper uses activity types (see Table 1). Those activity types resemble types of actions identified by Ochodek et al. [32] in their research that aimed at identifying transactions for the purpose of use-case-based effort estimation [33]. Five of the activity types (Update, Delete, Select, Validate, and Confirm) have their counterparts in Ochodek's types of actions (e.g. Update corresponds to Modification, Delete to Removal, etc.). However, the other five do not directly map onto Ochodekís types (e.g. Add, Enter, and Create correspond to Ochodekís Provision; similarly Read and Display map into Presentation). Moreover, three of Ochodekís types of actions (Transfer, Store, and Question) do not have their counterparts in the activity types of Table 1. This difference between activity types and Ochodekís types of actions is quite natural as they have been created for different purposes. Nevertheless, the question arises whether there is a unified theory of use-case actions that would serve the both purposes.

8. Conclusions

Event completeness is one of the aspects of quality of software requirements. *Ad hoc* approach to events identification provides poor results (see [22]). The H4U method, proposed in [22] and aimed at manual identification of events, was able to improve effectiveness of the process. However, the results concerning H4U were still not satisfactionary. Therefore, an automated method for event identification has been proposed in this paper. The method has been evaluated and compared to H4U. The following conclusions can be drawn from the results presented in this paper:

- The proposed method of automated identification of events can achieve higher accuracy than manual approaches. The final accuracy was at the level of 89%, compared to the average accu-

racy of the H4U method which was about 26%. The proposed method also achieved high precision (93%).
- The automated method is better than the H4U method in terms of speed. On average it can analyse around 10 steps per minute, while people using H4U are able to analyse about 1 step per minute.
- The event descriptions generated by the prototype tool could not be differentiated, in the sense of readability, from the descriptions proposed by humans. This was investigated in an experiment resembling Turing test, where participants (students of the Software Engineering Programme) had to compare pairs of the proposed descriptions of events. This makes the proposed method interesting from the practical point of view.

Probably both the results of the Turing test and the speed of events identification could be significantly improved if better NLP and NLG techniques were used. Even more important issue concerns the inference rules: we have discovered them using the induction principle and there is no guarantee that the proposed set is the best possible. Maybe another set of inference rules would lead to better results.

Identification of events in use cases is aimed mainly at supporting Analyst in elicitation of functional requirements. However, the proposed technique can be used in other contexts. For instance Paydar and Kahani investigated possibility of semi-automatic conversion of use-case diagrams into a corresponding activity diagrams [34]. One of the components of their method is a use case similarity metric which is used to retrieve from the model repository the most similar use cases. Their similarity metric takes into account subject of a use case, its behavior, concepts and actors. An alternative approach could be based on a sequence of possible events identified according to the approach presented in this paper.

Acknowledgments

This Project was funded by Polish National Science Center project UMO-2011/01/N/ST6/06794 conducted at Poznan University of Technology.

The authors are thankful to Dr. Miroslaw Ochodek for his help and comments concerning statistical analysis of the experiments' results. Thanks go also to the anonymous reviewers - their remarks helped to improve the paper.

References

[1] OpenNLP homepage. <http://opennlp.apache.org> (checked 04.06.13).
[2] Pablo Picasso's quote. <http://www.brainyquote.com/quotes/quotes/p/pablopicas102018.html> (checked 01.08.13).
[3] Stanford Log-linear Part-Of-Speech Tagger. <http://nlp.stanford.edu/software/tagger.shtml> (checked 04.06.13).
[4] Stanford Tokenizer. <http://nlp.stanford.edu/software/tokenizer.shtml> (checked 04.06.13).
[5] S. Adolph, P. Bramble, A. Cockburn, A. Pols, Patterns for Effective Use Cases, Addison-Wesley, 2002.
[6] A.J. Albrecht, Measuring application development productivity, in: Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium, October 1979, pp. 83–92.
[7] A.J. Albrecht, AD/M Productivity Measurement and Estimate Validation, in: IBM Corporate Information Systems and Administration, 1984.
[8] B. Alchimowicz, J. Jurkiewicz, M. Ochodek, J. Nawrocki, Building benchmarks for use cases, Comput. Inf. 29 (1) (2010) 27–44.
[9] I. Alexander, Scenario-driven search finds more exceptions, in: Proceedings of the 11th International Workshop on Database and Expert Systems Applications, DEXA '00, IEEE Computer Society, 2000, p. 991.
[10] Leo Breiman, Jerome Friedman, Charles J. Stone, R.A. Olshen, Classification and Regression Trees, first ed., Chapman and Hall/CRC, 1984.
[11] Ronald S. Carson, Requirements completeness: a deterministic approach, in: Proceedings of 8th Annual International INCOSE Symposium, 1998.
[12] A. Cockburn, Writing Effective Use Cases, Addison-Wesley, Boston, 2001.
[13] Karl Cox, Aybüke Aurum, D. Ross Jeffery, An experiment in inspecting the quality of use case descriptions, J. Res. Pract. Inf. Technol. 36 (4) (2004).
[14] Marie-Catherine de Marneffe, Bill MacCartney, Christopher D. Manning, Generating typed dependency parses from phrase structure parses, in: LREC 2006, 2006. <http://dx.doi.org/10.1.1.74.3875>.
[15] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Longman Publishing Co., Inc, 1995.
[16] Albert Gatt, Ehud Reiter, Simplenlg: a realisation engine for practical applications, in: Proceedings of the 12th European Workshop on Natural Language Generation, ENLG '09, Association for Computational Linguistics, 2009, pp. 90–93.
[17] Spence Green, Marie-Catherine de Marneffe, John Bauer, Christopher D. Manning, Multiword expression identification with tree substitution grammars: a parsing tour de force with french, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, EMNLP '11, Association for Computational Linguistics, 2011, pp. 725–735.
[18] Zhang Hong Hui, Atsushi Ohnishi, Integration and evolution method of scenarios from different viewpoints, in: Proceedings of the 6th International Workshop on Principles of Software Evolution, IWPSE '03, IEEE Computer Society, 2003, p. 183.
[19] R. Hurlbut, A Survey of Approaches for Describing and Formalizing Use Cases, Expertech, Ltd, 1997.
[20] I. Jacobson, Concepts for modeling large real time systems, Royal Institute of Technology, Dept. of Telecommunication Systems-Computer Systems, 1985.
[21] Daniel Jurafsky, James H. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, first ed., Prentice Hall PTR, 2000.
[22] J. Jurkiewicz, J. Nawrocki, M. Ochodek, T. Glowacki, HAZOP-based identification of events in use cases. An empirical study, Empirical Software Eng. doi: http://dx.doi.org/10.1007/s10664-013-9277-5.
[23] Mayumi Itakura Kamata, Tetsuo Tamai, How does requirements quality relate to project success or failure? In Requirements Engineering Conference, 2007, RE'07, 15th IEEE International, 2007, pp. 69–78.
[24] Dean Leffingwell, Calculating the return on investment from more effective requirements management, Am. Programmer 10 (1997) 13–16.
[25] Emmanuel Letier, Reasoning about Agents in Goal-Oriented Requirements Engineering, 2001.
[26] Emmanuel Letier, Jeff Kramer, Jeff Magee, Sebastián Uchitel, Monitoring and control in scenario-based requirements analysis, in: ICSE, 2005, pp. 382–391.
[27] Masayuki Makino, Atsushi Ohnishi, A method of scenario generation with differential scenario, in: RE, 2008, pp. 337–338.
[28] Brian W. Mar, Requirements for development of software requirements, in: Proceedings of NCOSE, 1994.
[29] Kjetil Molkken, Magne Jrgensen, A review of surveys on software effort estimation, in: Proceedings of the 2003 International Symposium on Empirical Software Engineering, ISESE'03, IEEE Computer Society, 2003, p. 223.
[30] Jerzy Nawrocki, Łukasz Olek, UC workbench – a tool for writing use cases and generating mockups, in: Extreme Programming and Agile Processes in Software Engineering, LNCS, vol. 3556, Springer, 2005, pp. 230–234.
[31] C.J. Neill, P.A. Laplante, Requirements engineering: the state of the practice, Softw. IEEE 20 (6) (2003) 40–45.
[32] M. Ochodek, B. Alchimowicz, J. Jurkiewicz, J. Nawrocki, Improving the reliability of transaction identification in use cases, Inf. Softw. Technol. 53 (8) (2011) 885–897.
[33] M. Ochodek, J. Nawrocki, K. Kwarciak, Simplifying effort estimation based on Use Case Points, Inf. Softw. Technol. 53 (3) (2011) 200–213.
[34] Samad Paydar, Mohsen Kahani, A semi-automated approach to adapt activity diagrams for new use cases, Inf. Softw. Technol. (2014).
[35] Anna N. Rafferty, Christopher D. Manning, Parsing three german treebanks: lexicalized and unlexicalized baselines, in: Proceedings of the Workshop on Parsing German, PaGe'08, Association for Computational Linguistics, 2008, pp. 40–46.
[36] M. Śmiałek, Software Development with Reusable Requirements-based Cases, Prace Naukowe – Politechnika Warszawska: Elektryka, Oficyna Wydawnicza Politechniki Warszawskiej, 2007. <http://books.google.pl/books?id=9jafMwAACAAJ>.
[37] George E. Stark, Paul W. Oman, Alan Skillicorn, Alan Ameele, An examination of the effects of requirements changes on software maintenance releases, J. Softw. Maint. 11 (5) (1999) 293–309.
[38] Alistair G. Sutcliffe, Neil A.M. Maiden, Shailey Minocha, Darrel Manuel, Supporting scenario-based requirements engineering, IEEE Trans. Softw. Eng. 24 (12) (1998).
[39] Alan M. Turing, Computing machinery and intelligence, Mind LIX (1950) 433–460.
[40] Axel van Lamsweerde, Emmanuel Letier, Handling obstacles in goal-oriented requirements engineering, IEEE Trans. Softw. Eng. 26 (10) (2000) 978–1005.
[41] Karl E. Wiegers, Inspection checklist for use case documents, in: Proceedings of 9th IEEE International Software Metrics Symposium, IEEE, 2003.