sciendo RIGA TECHNICAL UNIVERSITY
Applied Computer Systems
ISSN 2255-8691 (online)
ISSN 2255-8683 (print)
December 2019, vol. 24, no. 2, pp. 94–103
https://doi.org/10.2478/acss-2019-0012
https://content.sciendo.com

Extracting TFM Core Elements From Use Case Scenarios by Processing Structure and Text in Natural Language

Erika Nazaruka[1*], Jānis Osis[2], Viktorija Gribermane[3]
[1, 2]*Department of Applied Computer Science, Riga Technical University, Riga, Latvia*
[3]*Institute of Applied Computer Systems, Riga Technical University, Riga, Latvia*

***Abstract* – Extracting core elements of Topological Functioning Model (TFM) from use case scenarios requires processing of both structure and natural language constructs in use case step descriptions. The processing steps are discussed in the present paper. Analysis of natural language constructs is based on outcomes provided by Stanford CoreNLP. Stanford CoreNLP is the Natural Language Processing pipeline that allows analysing text at paragraph, sentence and word levels. The proposed technique allows extracting actions, objects, results, preconditions, post-conditions and executors of the functional features, as well as cause-effect relations between them. However, accuracy of it is dependent on the used language constructs and accuracy of specification of event flows. The analysis of the results allows concluding that even use case specifications require the use of rigor, or even uniform, structure of paths and sentences as well as awareness of the possible parsing errors.**

***Keywords* – Computation independent model, functional feature, natural language processing, Stanford CoreNLP, topological functioning model, use case.**

I. Introduction

Model-driven and model-based software development approaches propose using transformations between models from different viewpoints in order to get source code. One of the brightest representatives of the approaches is Model Driven Architecture (MDA). MDA suggests using a chain of model transformations, namely, from a Computation Independent Model (CIM) to a Platform Independent Model (PIM), then to a Platform Specific Model (PSM) and to source code [1]. Transformation of requirements into analytical and design models as well as into test cases is one of important questions in the field of the system analysis and design.

In our vision of topological functioning model driven software development (Fig. 1), the starting point is exactly text fragments in a natural language. These fragments represent either information about the implemented functional characteristics of the already operating system (both manual and automated), or desired functional characteristics of the system to be built. They represent knowledge about the systems from a computation independent viewpoint. A Topological Functioning Model (TFM) formalises representation of the knowledge from the same viewpoint. Moreover, the two types of knowledge are to be in conformity, i.e., they will have the common core that consists of functional characteristics that already exist or are obvious and will be implemented in the desired system. Generation and refinement of the TFM is an iterative process. Transition to the platform independent (or specific) viewpoint is possible using transformation of the TFM to an analytical or design model, for example, to the following Topological UML (that is an UML, i.e., Unified Modelling Language, extension) diagrams: topological class diagrams, topological use case diagram, communication diagrams, and object diagrams; state diagrams; component and deployment diagrams [2]. The final step of the transformations is generation of source code for selected platforms.

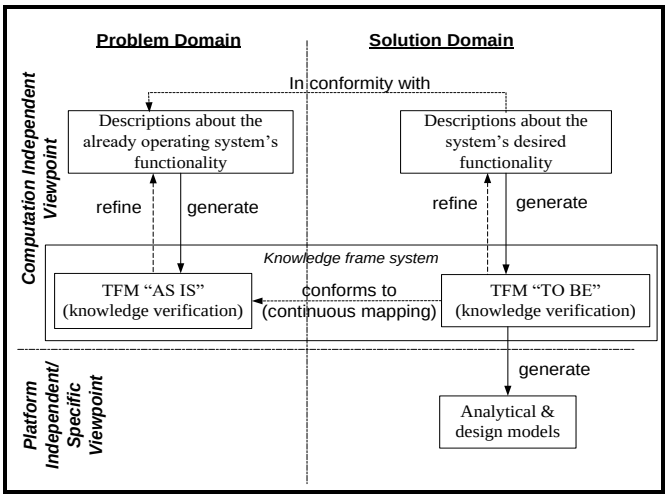

Fig. 1. TFM driven software transformation.

The TFM is embedded into a knowledge frame system [3], [4] for keeping and managing results of the text processing. The aim is to gain from an inferring mechanism and flexibility of the knowledge base as well as to discover conflicts in knowledge, manage synonyms, and infer new knowledge from the existing one.

Preparation of textual descriptions and manual knowledge acquisition from them is resource-consuming [5]. In practice, there are two ways how to deal with this issue. The wide-used

* Corresponding author's e-mail: erika.nazaruka@rtu.lv

one is to skip the step of preparation of descriptions/specifications and to start from human analysis of the available information. The second one is to automate or semi-automate this process. We are on the automation way.

Automation of transformation of requirements to models has one important issue, i.e., a natural language. Requirements can be expressed using a variety of formats, for example, as a use case scenario, a user story, a structured text, an explaining scheme, an explaining case, etc. The more rigor structure a format has, the easier its automated processing is. However, all of the formats have one difficulty inherited from a natural language, i.e., the use of natural language suggests using multiple possible language constructs for expressing the same phenomena.

The goal of the research is to get knowledge from use case scenarios according to all the core elements of the TFM. Since a use case scenario is a semi-structured specification, it requires parsing not only the structure but also the text using Natural Language Processing (NLP) capabilities.

The paper is organised as follows. Section II describes the core elements of the TFM that need to be extracted and the main principles of the (NLP. Section III presents the algorithm for extracting values for the TFM core elements. Section IV demonstrates the work of the algorithm on an example. The paper is concluded with discussion on main findings and further research areas.

II. TFM And Use Case Scenarios

A. Core Elements of Topological Functioning Model

The TFM is a formal mathematical model proposed by Janis Osis at Riga Technical University in 1969 for modelling and analysing functionality of mechanical systems [6]. However, this model can be applied to systems from business, software, biological, mechanical or other domains. The TFM represents modelled functionality as a digraph (X, Θ), where X is a set of inner functional characteristics (called functional features) of the system, and Θ is a topology set on these characteristics in a form of a set of cause-and-effect relations [6].

TFM models can be compared for similarities using a continuous mapping mechanism [7]. This mechanism is used for making consistent solution and problem domains (Fig. 1). Since the 1990s, the TFM has been elaborated for software development [8] starting from principles of object-oriented system analysis and design and ending with principles of the MDA.

The TFM has topological and functioning properties [6]. The topological properties take their origin in topological algebra. They are connectedness, neighbourhood, closure and continuous mapping. Connectedness ensures that all functional characteristics of the system are dependent of each other work in a direct or an indirect way. Neighbourhoods are sets, where each set is a functional characteristic of the system together with all its direct (with the step equal to 1) predecessors and followers. A mathematical operation of union of all neighbourhoods of the system inner functional characteristics is called “closure”. The closure is used to define the border of the system in a mathematical way. Since any TFM is a topological space, they can be compared for similarity or either refined or simplified. Thanks to continuous mapping between topological spaces, the initial structure of the topological models is preserved during modifications.

The functioning properties take their origin in the system theory. They are cause-effect relations, cycle structures, inputs and outputs. The cause-effect relations are those dependencies between functional characteristics of the system that allow the system to function. The end of execution of one functional characteristic triggers initiation of other depending functional characteristics. Since we talk about the system that is running (or functioning), these dependencies form a cycle (or cycles) of functionality. Behaviour of the system depends on input signals from the external environment as well as on output signals of the system (reaction) to the external environment.

The composition of the TFM is presented in [6]. Rules of composition and derivation of the TFM from the textual system description within TFM4MDA (TFM for Model Driven Architecture) are provided by examples and described in detail in several publications [9]–[11]. The TFM can be manually created in the TFM Editor or can also be generated automatically from the business use case descriptions in the IDM toolset [12].

The main TFM concept is a functional feature (FF_i) that represents a system functional characteristic, e.g., a business process, a task, an action, or an activity [6]. It can be specified by a unique tuple (1).

$$FF_i = <A, \boldsymbol{R}, \boldsymbol{O}, \boldsymbol{PrCond}, \boldsymbol{PostCond}, \boldsymbol{Pr}, \boldsymbol{Ex}>, \quad (1)$$

where [6]:

- A is an object action;
- ***R*** is a set of results of the object action (it is an optional element);
- ***O*** is an object that gets the result of the action or a set of objects that are used in this action;
- ***PrCond*** is a set of preconditions or atomic business rules;
- ***PostCond*** is a set of post-conditions or atomic business rules;
- ***Pr*** is a set of providers of the feature, i.e., entities (systems or sub-systems) that provide or suggest an action with a set of certain objects;
- ***Ex*** is a set of executors (direct performers) of the functional feature, i.e., a set of entities (systems or sub-systems) that enact a concrete action.

The second TFM concept is a *cause-effect relation* between functional features. It defines a cause from which triggering of an effect occurs.

Formal definitions of cause-effect relations and their combinations are given in [2], [13]. The main definition states that a cause-effect relation is a binary relation that links a cause functional feature to an effect functional feature. In fact, this relation indicates control flow transition in the system.

Cause-effect relations (and their combinations) may be joined by logical operators, namely, conjunction (AND), disjunction (OR), or exclusive disjunction (XOR). The logic of the combination of cause-effect relations denotes system

behaviour and execution (e.g., decision making, parallel or sequential actions).

Thus, the elements of the functional features and the cause-effect relations between them must be identified in the text.

B. Structure of Use Case Scenarios

Use cases can be expressed in both informal and structured manners. The informal one is just a sequence of steps written in a free manner in a natural language. More formal specifications are the ones that have a flow or flows of numbered steps, steps of interaction between an actor and a system in a table form, or both supported by an UML sequence or activity diagrams.

In the research, we deal with the numbered step format. Use case scenario structures that have a flow or flows of numbered steps may have different forms. For example, one of more completed forms is presented by Winters and Schneider [14], where it has the following parts:

- a use case name in the form "Do [preposition] What";
- a brief description usually a paragraph or less that may include the priority and status of the use case;
- a context diagram that is part of the entire use case diagram;
- preconditions of a use case that must be true before the use case starts;
- a flow of events (basic and alternatives);
- post-conditions of a use case that must be true when the use case ends independently of the flow executed;
- a subordinate use case diagram;
- subordinate use cases with their own flows of events;
- an activity diagram for the flow of events;
- a view of participating classes in a form of a class diagram;
- sequence diagrams;
- a user interface;
- business rules in the form of a list that this use case implements;
- special requirements that pertain to this certain use case, e.g., timing, sizing, or usability;
- other artefacts such as references to the subsystems, analysis models, etc.;
- outstanding issues, i.e., a list of questions that need to be answered.

A context diagram, a subordinate use case diagram, an activity diagram, a view of participating classes, sequence diagrams, and a user interface contain graphical schemes.

The numbered steps format of a use case scenario usually has predefined *keywords*, for example:

- "the use case begins when" is used for an entry point into the use case;
- "the use case ends" is used for an exit point form the use case;
- "for each… end loop" is used for iterations;
- "basic path" is used as a title of the section of the normal flow of steps;
- "alternative paths" is used as a title for the section of branches in the execution logic;
- "Alternative <*number*>: <*explanation*>" is used as a title of a branch in the execution logic;
- "special requirements" is used as a title of the section for non-functional requirements; etc.

Used keywords are not specified in the format itself. They are discussable and conventional within a certain developer team or a project.

C. Existing Mappings Between TFM Elements and Use Cases

Application of NLP to use case scenarios is partially implemented in the IDM (Integrated Domain Modelling) toolset, where processing of a use case scenario is performed using the Stanford Parser Java Library for identifying the executors ***Ex*** and describing the functional feature *D* that is the *verb phrase VP* from the text of a step in a scenario [15], [16].

The prerequisite for parsing is that sentences of use case steps must be in the simple form to answer the question "Who does what?", e.g., "Librarian checks out the book". This structure of a sentence is a recommended one for use case steps [14], [17].

Parsing in the IDM is performed according to these steps:

- Identify *coordinating conjunctions* to split a sentence into several clauses, and, thus, several functional features;
- Identify the *verb phrase* in a clause that is considered a union of action *A*, object *O* and result *R* (if it is indicated) and constitutes the so-called *description* of the functional feature *D*;
- Identify the *noun phrase* that is marked as executor Ex_i if it meets the same noun in the list of actors for the use case;
- Pre-conditions and post-conditions are taken directly from the corresponding preceding step in the use case (if they are specified);
- Topological relations are equal to the sequence of use case steps.

As a result, the elements *A*, ***R***, ***O*** are *implicitly* located in the description of functional feature *D*, and each step has a single executor Ex_i.

III. Extracting TFM Core Elements

A. Tags and Denotations Used in the Algorithm

Before discussion of the algorithm, the notation and abbreviations used must be explained. The algorithm uses the Stanford CoreNLP toolkit [18] that contains components that deal with the following NLP tasks: tokenization, sentence splitting, POS tagging, morphological analysis (identification of base forms), NER, syntactical parsing, co-reference resolution and other annotations such as gender and sentiment analysis. The syntactical parsing uses constituent and dependency representations. Discovering basic dependencies can help in identification of actions and corresponding objects, results, modes (that can serve for identification of causal dependencies) and initiators.

In the present research, we use Stanford CoreNLP version 3.9.2 that for POS tagging uses tags listed in Penn Treebank II [19]. In the research, the following tags are used: S – simple declarative clause, NN – noun, single, NNS – noun, plural, NP – noun phrase, PRP – preposition, VBZ – verb, 3rd person

singular present, VBP – verb, non-3rd person singular present, VBD – verb, past tense, VBG – verb, gerund or present participle, VBN – verb, past participle, VB – base form, VP – verb phrase, IN – preposition or subordinating conjunction, RP – particle.

The following denotations are used for dependencies: *dobj* – a direct object, *compound* – a complex noun that consists of several words, *prt* – a particle, *nsubj* – a nominal subject.

A direction of a dependency is denoted by the arrow symbol "→". Alternative tags in a pattern are separated with the symbol "|". Optional elements are located in the square brackets "[]".

B. Processing Structure and Text in Natural Language

Thus, analysis of a use case starts from parsing the structure using the keywords predefined for section titles and then analysing the corresponding description of execution paths (Fig. 2).

The use case itself forms one functional feature (*ff*). This functional feature can be expanded into a set of specialised functional features *specialized_ff*. The set is obtained during the analysis of paths in a use case itself and in subordinated use cases. This set is linked with the functional feature *ff*.

The cause-effect relations from all the scenarios are kept in the collection *cause_effect_rels*.

Therefore, by analysing a use case and its scenarios it is possible to extract a functional feature at the level two of abstraction. If a scenario invokes an included use case or an extending use case, it is not expanded but forms a separate specialised functional feature. However, further, after analysis of its own specification, it will have its own sets of specialised functional features. Hence, the multiple layers of abstraction could be formed. It means that the TFM preserves its abilities to be simplified and refined, while keeping its structure.

Text in the use case specification is analysed in several blocks. The reason is that each sentence can contain one or several actions and, therefore, potential functional features. NLP tasks are executed for a collection of sentences from the blocks. Thus, the text for NL analysis is formed from a use case name, a basic path and alternative paths, subordinate use case names, subordinate use case basic paths and alternative paths. The text is sent to NLP tool as a whole in order to reduce time of processing. The obtained result is then processed using the following patterns [20] and rules.

Rule 1. Identify action *A*. Action *A* is presented by the infinitive of verb v_i that fits the pattern S(VP(VBZ | VBP | VBD | VBN | VBG | VB v_i [→ *compound:prt* → RP *particle*]) → *dobj* → NP(NN| NNS| PRP n_1)).

Rule 2. Identify domain objects *R* and *O*. If v_i is found, then get *n1* from the same structure that matches the pattern.

Rule 2.1. If VP(v_i) is not linked by *nmod*: but *nmod:agent* with another noun NN|NNS|PRP n_j, then the following is true:

a) If NP(n_1 → *compound* → n_2) AND VP(NP(n_1) → *nmod:poss* | *of* | *to* | *into* | *from* | *for* → NP(n_2)), then the object O_i is equal to n_1 and the result R_i is left empty. Otherwise, if one of such links does exist, the object *Oi* is equal to n_2.

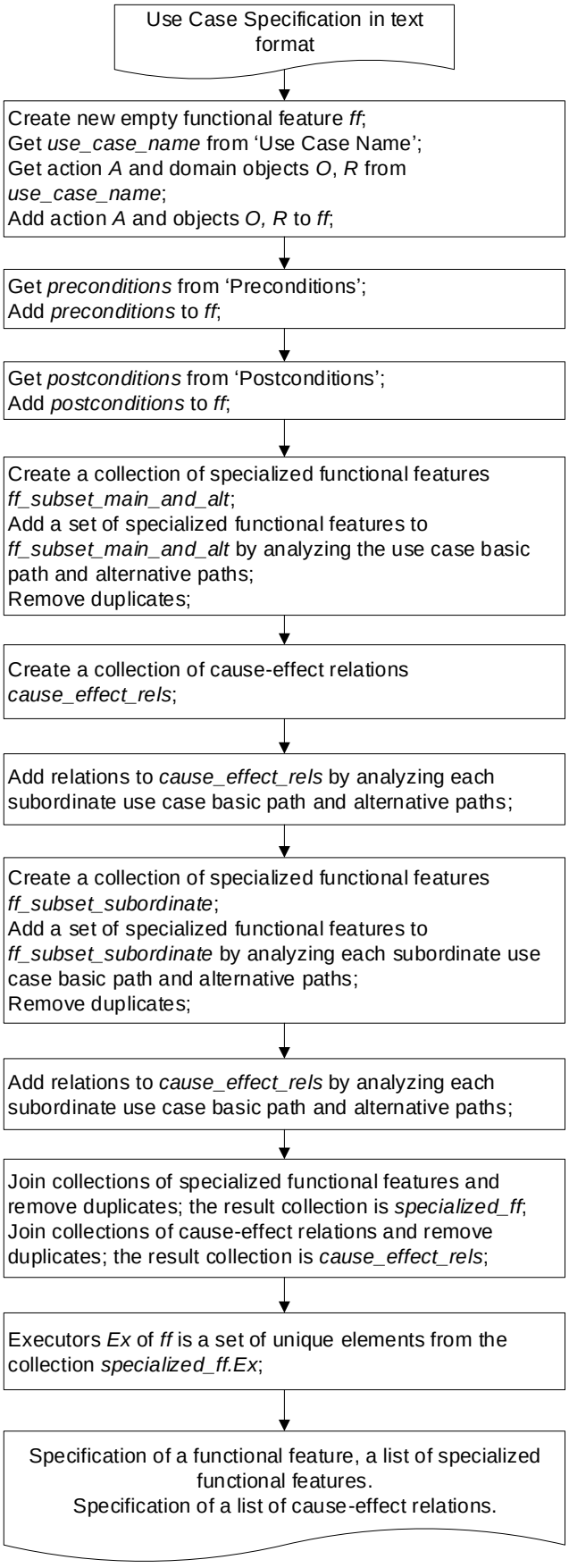

Fig. 2. The steps of processing the structure and scenarios in a use case.

b) If NP(n_1 → *compound* → n_2), then R_i is equal to the NP(n_1)+" of". The object O_i is equal to n_2.
c) If VP(NP(n_1) → *nmod:poss* | *of* | *to* | *into* | *from* | *for* → NP(n_2)), then R_i is equal to the NP(n_1) → [NP(n_2) →] *case* → IN *preposition*. The object O_i is equal to n_2.

Rule 2.2. If VP(v_i) is linked with another NN | NNS | PRP n_2 by *nmod*: but *nmod:agent* , then the following is true:
a) The object O_i is equal to n_2 from PP(NP(n_2)).
b) The result R_i is equal to NP(n_1) + IN *preposition*, where *preposition* is in the PP(NP(n_2)).

Rule 3. Identify executors *Ex*. A noun n_i that matches the pattern S(VP(VBZ | VBP | VBD | VBN | VBG | VB v_i [+ → *compound:prt* → RP *particle*]) → *nsubj* → NP(NN | NNS | PRP n_i)).

Rule 4. Identify providers *Pr*. Nevertheless, providers ***Pr*** fit the same pattern as the executors, they must be acquired by a developer.

Rule 5. Identify pre-conditions *PrCond*. Search for patterns "IF/WHEN<*condition*> THEN: step(s)", "*For each <element> … end loop*", "*While <action|event|condition>… end loop*". The <condition>, "For each <element>", "While<*action|event|condition*>" are a precondition to the defined specialised functional feature that contains the first action in the sentence. Sometimes, the text may contain additional functional characteristic and form a separate functional feature. If the pattern follows the construct "The use case begins", then the precondition is related to the first functional feature of this use case.

Rule 6. Identify post-conditions *PrCond*. At this point of the research, post-conditions can be identified only if they are explicitly marked as post-conditions.

Rule 7. Identify a cause-effect relation *T* in the simple form. The simple form of a cause-effect relation is presented as a tuple <*cause-ff*, *effect-ff*>. The topology is determined according to the sequences of sentences in the block. The following sub-rules are applied:

- **Rule 7.1. Search for a sequence of steps.** Successful termination of each preceding step initiates its direct subsequent step. Thus, *T* = <a functional feature with the previous step action, a functional feature with the current step action>;
- **Rule 7.2. Search for redirection to the indicated step.** The redirection may be expressed either using some pre-defined phrases, e.g., "the use case continues at <flow> step <number>", or another phrase with similar meaning. These phrases must be known. *T* = <a functional feature with the current action, a functional feature with the first action from the redirected sentence>;
- **Rule 7.3. Search for redirection to an alternative flow.** If an alternative flow takes only a few sentences, it can be located directly within the flow-of-events section. In this case, Rule 7.1 is applied. Otherwise, Rule 7.2 is applied;
- **Rule 7.4. Search for redirection from an alternative flow.** Sometimes, a basic flow contains only the "typical" sequence of steps without any redirection to alternative flows. Then a point, where the alternative starts, is indicated in the alternative itself using phrases like "In step <*number*>, <*precondition*>, <*step/event*>". In this case, *T* = <the functional feature with the first action from the indicated sentence, the functional feature with the first action of the current sentence>;
- **Rule 7.5. Search for redirection to a subordinated use case.** A subordinate use case has its own name that is used as a marker with a keyword "subordinate use cases" to express a transition to it;
- **Rule 7.6. Search for invocation of an included use case.** An included use case is invoked using the keyword "Include" and a name of the use case. Thus, *T* = <the functional feature with the current action, the functional feature formed by the name of the included use case>;
- **Rule 7.7. Search for invocation of an extending use case.** An extending use case is invoked using the keyword "Extends" and a name of the use case. An extending use case is invoked using its name as a marker at the certain extension point either in a flow of events or in a special section of extension points. A precondition must also be indicated. Thus, *T* = <the functional feature with the current action, the functional feature formed by the name of the extending use case>;
- **Rule 7.8. Cycle constructs.** In order to indicate iterative sequences, constructs *For each <element> … end loop* and *While <action/event>… end loop* are used. Here, *T* = "functional feature of the last action in the block, functional feature of the first action in the block>;
- **Rule 7.9. Search for the functional feature in the pre-condition.** If a pre-condition contains a functional feature, then *T* = <the functional feature from the pre-condition, the functional feature with the current action>.

Analysis of cause-effect relations in a use case specification is easier thanks to causality explicitly indicated in most cases. However, steps may contain short discourses that must be analysed in the same way as text in prose.

IV. Illustrative Example

A. Use Case Scenario

Let us consider a use case "Search for borrowed book" from a library system. The use case scenario uses the template presented in Section II but skips parts with graphical information as well as use case description, business rules and special requirements.

Use Case Name: Search for borrowed book.

Brief Description: This use case describes the process for finding a particular borrowed book in the system.

Preconditions: The user is logged in.

Flow of Events:

Basic Path:

1. The use case begins when the Find Book screen is displayed.
2. The user enters a borrowed book ID or client ID.
3. The user selects Search.
4. The system requires the database to get the borrowed books requested (subordinate use case: Get Borrowed Book List, Get Borrowed Book).

5. If the user enters a client ID, then
   a. the system displays a list of borrowed books for that client, including at least a borrowed book ID and a date of borrowing a book.
   b. the user selects one borrowed book.

   End if
6. The system returns the selected borrowed book and the use case ends.

Alternative Paths:

- In Step 4, borrowed book ID is not found in system, the system displays an error message.
- In Step 4, there is no such a client, the system displays an error message.
- In Step 4, a database is not available, the system displays an error message.

Post-conditions: none.

Subordinate Use Cases:

Subordinate Use Case Name: *Get Borrowed Book List*

Basic Path:

1. The use case begins when a request for a borrowed book list is received.
2. The system sends the client ID to the borrowed book database with a query for all borrowed books for this client.
3. The database returns a list of all records found that match the client ID submitted. The list must include at least a borrowed book ID and the date the book was borrowed for each borrowed book in the list.
4. The user case ends.

Subordinate Use Case Name: *Get Borrowed Book*

Basic Path:

1. The use case begins when a request for a borrowed book is received.
2. The system sends the borrowed book ID to the borrowed book database with a query for the borrowed book matching the identifier.
3. The database returns zero or one borrowed book matching the identifier.
4. The use case ends.

B. Processing Use Case Scenarios

According to the scheme presented in Fig. 2, at the beginning the first functional feature linked with the use case itself is created.

1. Create an empty functional feature *ff* =<*A*, *R*, *O*, *PrCond*, *PostCond*, *Ex*, *Pr*>.

2. The found *use_case_name* = "Search for borrowed book". Action *a* = "search", objects *O* and *R* fits the pattern from Rule 2.1-a. It means *O* = "book", result *R* is empty. Thus, *ff* =<*A* = "search", *R* = "", *O* = "book", *PrCond*, *PostCond*, *Ex*, *Pr*>.

3. Pre-conditions *ff.PrCond* = "The user is logged in".

4. Post-conditions *ff.PostCond* = "".

5. Initialisation of specialised functional feature collection *ff_subset_main_and_alt*. After the analysis, the resulting functional features are presented in Table I.

TABLE I

THE COLLECTION *FF_SUBSET_MAIN_AND_ALT*

ID	***A***	***R***	***O***	***Precond***	***Ex***
The use case begins when the Find Book screen is displayed (skipped)					
The user enters a borrowed book ID or client ID					
sff_1	enter	ID of	book	*when the Find Book screen is displayed*	the user
sff_2	enter	ID of	client	*when the Find Book screen is displayed*	the user
The user selects Search					
sff_3	select		Search		the user
The system requires the database to get the borrowed books requested (subordinate use case: Get Borrowed Book List, Get Borrowed Book).					
sff_4	require		database		the system
sff_5	get	list of	book		
sff_6	get		book		

If the user enters a client ID, then
a. the system displays a list of borrowed books for that client, including at least a borrowed book ID and date of borrowing a book.
b. the user selects one borrowed book.
End if					

sff_7	display	list of	books	the user enters a client ID (sff_2)	the system
sff_8	select		book		the user
The system returns the selected borrowed book and the use case ends.					
sff_9	return		book		the system
• In Step 4, borrowed book ID is not found in the system, the system displays an error message. • In Step 4, there is no such a client, the system displays an error message. • In Step 4, a database is not available, the system displays an error message.					
sff_10	display	message of	error	borrowed book ID is not found in the system	the system
sff_11	display	message of	error	no such a client	the system
sff_12	display	message of	error	database is not available	the system

6. Initialisation of cause-effect relation collection *cause_effect_rels.*

7. The collection *cause_effect_rels* contains the results presented in Table II.

8. Initialisation of specialised functional feature collection *ff_subset_subordinate*. After the analysis, the resulting functional features are presented in Table III. The sentence "*The database returns a list of all records found that match the client ID submitted*" did not match the verb pattern, since the noun "a list" was not recognised as a direct object (Fig. 3).

TABLE II

THE COLLECTION *CAUSE_EFFECT_RELS* IN STEP 7

Cause ID	Effect ID	Rule No.
sff_1	sff_3	Rule 7.1
sff_2	sff_3	Rule 7.1
sff_3	sff_4	Rule 7.1
sff_4	sff_5	Rule 7.5
sff_4	sff_6	Rule 7.5
sff_5	sff_10	Rule 7.1
sff_5	sff_11	Rule 7.1
sff_5	sff_12	Rule 7.1
sff_6	sff_10	Rule 7.1
sff_6	sff_11	Rule 7.1
sff_6	sff_12	Rule 7.1
sff_2	sff_7	Rule 7.9
sff_7	sff_8	Rule 7.1
sff_8	sff_9	Rule 7.1

TABLE III

THE COLLECTION *FF_SUBSET_SUBORDINATE*

ID	***A***	***R***	***O***	***Precond***	***Ex***
Specialised functional features for *sff_5 Get List of Book*					
The use case begins when a request for a borrowed book list is received.					
The system sends the client ID to the borrowed book database with a query for all borrowed books for this client.					
sff_13	send	ID of	client	*when a request for a borrowed book list is received*	the system
The database returns a list of all records found that match the client ID submitted. The list must include at least a borrowed book ID and the date the book was borrowed for each borrowed book in the list.					
sff_14	include	ID of	book		the list
sff_15	include		date		the list
The user case ends. (skipped)					
Specialised functional features for *sff_6 Get Book*					
The use case begins when a request for a borrowed book is received.					
The system sends the borrowed book ID to the borrowed book database with a query for the borrowed book matching the identifier.					
sff_16	send	ID of	book	*when a request for a borrowed book is received*	the system
The database returns zero or one borrowed book matching the identifier.					
sff_17	return		book		the database
The use case ends. (skipped)					

9. The set of cause-effect relations presented in Table IV is added to the *cause_effect_rels*.

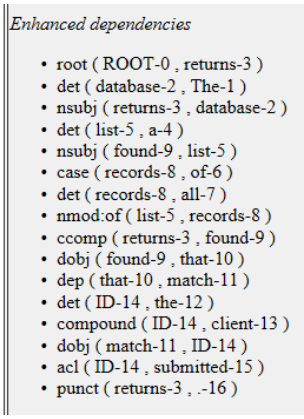

Fig. 3. Enhanced dependencies of the sentence "The database returns a list of all records found that match the client ID submitted".

TABLE IV

ADDITION TO THE COLLECTION *CAUSE_EFFECT_REL*

Cause ID	Effect ID	Rule No.	Expands
sff_4	sff_13	Rule 7.5	sff_5
sff_13	sff_14	Rule 7.1	sff_5
sff_13	sff_15	Rule 7.1	sff_5
sff_4	sff_16	Rule 7.5	sff_6
sff_16	sff_17	Rule 7.1	sff_6

10. Initialisation of the collection *specialized_ff*. The result collection that is a union of *ff_subset_main_and_alt* and *ff_subset_subordinate*.

11. Executors of the functional feature *ff*, i.e., *ff.Ex* = *specialized_ff.Ex()* -> *getUnique()* = {"the user", "the system", "the database", "the list"}.

C. Discovered Issues and Their Causes

The proposed technique suggests using processing of structure and text in use case specifications. Text processing is based on matching the patterns for sentences in the active voice. The main objective of it is to obtain a set of functional characteristics (features) of the system.

The suggested technique has met several issues:

1. Sentence expressions lack either the initiator of the step, or a direct object. In the former case, a reason is the use of the passive voice in the sentence without indicating by whom the action is done. In the latter case, a reason is that a direct object is expressed using clauses that play a role of adjectives or long explanatory phrases.

2. Pre-conditions with the same meaning are expressed using different phrases. Therefore, their analysis can lead to duplications in functional features or different and redundant cause-effect relations.

3. The indicated paths in event flows may have conditional sequences that are related to subordinated use cases, but the type of the relation can be not clear, namely, where they start and end as well what step is next. It is demonstrated in this example, in Steps 4 and 5 of the main use case.

4. The use case scenario may have different levels of detalisation. It requires a structure that keeps abstraction/detalisation relations.

5. In some cases, a statement in the active voice used for some additional explanation is ambiguous. For example, specialised functional features sff_14 and sff_15 have "the list" as an executor. However, "the list" is a domain object. In this case, two concerns are mixed. The first one, where a use case specification must be, is a viewpoint on the interaction between an actor and a system. The second one, which is indicated in the sentence, is a viewpoint on the application inner logic.

6. Stanford CoreNLP does not take into account that some titles (or names of the domain objects) may be expressed using verb phrases. It leads to the incorrect determination of dependencies between parts of the sentence.

The enumerated issues lead to the understanding that processing of the use case specifications, which allows using not only simple sentences in the form "Who does what", is the same complex as processing of the text in the formal written style.

V. Related Work

Reducing time for analysis using knowledge extraction from different types of input data and representing them as a model is also discussed in other authors' work.

The first type of input data is a textual document. The target is models and UML diagrams, for instance, creation of use case diagrams [21] and UML Activity Diagrams using identification of simple verbal sentences [22] from textual requirements in Arabic. UML class diagrams from textual requirements [23], and from use case descriptions [24] are created in English. Researchers also analyse textual user requirements in natural language and requirements used by engineering diagrams to create the Use Case Path model, the Hybrid Activity Diagram model and the Domain model [25]. In research [25], the authors applied syntax analysis by MBT tagger, semantics analysis to discover roles of words in the sentence (subject, predicate and object) and connections among them and then created a semantic network for text models. At the final step, the authors transformed this semantic network to one of the target models using patterns. Automated composition of conceptual diagrams can be performed by using the natural language analysis [26]. However, the authors have noted a need for human participation in the natural language analysis since the result can be affected by several issues of natural language itself, i.e., sentence structures may have different forms that are not completely predictable, syntactical correctness of sentences, as well as ambiguity in determining attributes as aggregations and in generalization.

The overview of existing solutions in the field of UML model creation from textual requirements and business process model creation from textual documents [27] showed that existing means allowed creating class diagrams, object diagrams, use case diagrams, and several of them provide composition of sequence, collaboration and activity diagrams. However, all the solutions have certain limitations, for example, some require user intervention, some cannot perform analysis of irrelevant classes, some require structuring a text in a certain form before processing, and some cannot correctly determine several structural relationships between classes. The only approach mentioned by the authors [27] that allows deriving a complete business process model is presented by Friedrich, Mendling and Puhlmann [28].

Some approaches use ontologies predefined by experts in the field and self-developed knowledge acquisition rules in order to extract knowledge on necessary properties or elements and their values from text documents [29], [30].

Some approaches for analysis scenarios and data in textual use cases use NLP. The purpose is scenarios for test cases [31]–[34] and behavioural models. Use cases itself can be transformed to test case scenarios. For example, generation of system test cases from uses cases "in the use case modelling language USL" [31], [34]. In this approach the authors transform a use case to test cases using predefined mappings between their metamodels and domain concepts of the system captures in an UML class diagram. Besides that, a USL model is ought to be created manually. Therefore, the authors use already pre-processed specifications in such a way dealing with texts in the natural language. More interesting are two approaches that use NLP for requirements transformation to test scenarios [32] and behavioural models [33], [35]. The same as the previous authors, Sarmiento et.al. [32] use a specific language, RNL, that restricts the vocabulary used to write scenarios to unambiguous declarative and imperative sentences. Wang et.al. [33] use a similar approach. They suggest using of Restricted Use Case Modeling (RUCM). Preconditions and postconditions are reformulated by a software developer as OCL (Object Constraint Language) constraints. Generation of scenarios for Petri Net-Based behavioural models from textual use cases [35] uses similar restrictions to the source text. However, here the authors analyse also a limited set linguistical patterns of sentences. As mentioned, all of these mentioned approaches try to restrict templates and languages of use cases, while our proposed idea is an attempt to proceed unrestricted textual descriptions.

VI. Conclusion

The paper presents a discussion on extracting core TFM elements from the use case specifications. The presented specification has a complex structure and allows using normal, not simplified sentences.

During the research, issues related to processing of the natural language as well as to the structure are determined. Issues that originate from the natural language are related to different language constructs used for expressing direct objects, initiators of actions, conditions, titles/names of the domain objects and the voice used. Issues related to the specification structure mostly originate from the flexibility of path descriptions. Sometimes, branching conditions and flow return places are ambiguously indicated, if indicated at all.

It is possible to conclude that although the benefit of use case specifications is that information is structured in the predefined format, the permitted flexibility in expressions and language

constructions has a high risk of low accuracy in automated processing of use cases.

The future research area is related to more accurate analysis of the constructs expressed in the natural language.

REFERENCES

[1] J. Miller and J. Mukerji, "Model Driven Architecture (MDA)," 2001.
[2] J. Osis and U. Donins, "Topological UML Modeling", in *Computer Science Reviews and Trends*, Elsevier, pp. 133–151, 2017. https://doi.org/10.1016/B978-0-12-805476-5.00005-8
[3] V. Nazaruks and J. Osis, "Joint Usage of Frames and the Topological Functioning Model for Domain Knowledge Presentation and Analysis," in *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: MDI4SE*, 2017, pp. 379–390. https://doi.org/10.5220/0006388903790390
[4] V. Nazaruks and J. Osis, "Verification of Causality in the Frame System based on the Topological Functioning Modelling," in *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, Portugal, Funchal, Madeira, 23-24 March, 2018*, 2018, pp. 513–521. https://doi.org/10.5220/0006817905130521
[5] M. Elstermann and T. Heuser, "Automatic Tool Support Possibilities for the Text-Based S-BPM Process Modelling Methodology," in *Proceedings of the 8th International Conference on Subject-oriented Business Process Management - S-BPM '16*, 2016, pp. 1–8. https://doi.org/10.1145/2882879.2882882
[6] J. Osis and E. Asnina, "Topological Modeling for Model-Driven Domain Analysis and Software Development : Functions and Architectures," in *Model-Driven Domain Analysis and Software Development: Architectures and Functions*, Hershey, PA: IGI Global, 2011, pp. 15–39. https://doi.org/10.4018/978-1-61692-874-2.ch002
[7] E. Asnina and J. Osis, "Computation Independent Models: Bridging Problem and Solution Domains," in *Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development*, 2010, pp. 23–32. https://doi.org/10.5220/0003043200230032
[8] J. Osis, E. Asnina, and A. Grave, "Computation Independent Representation of the Problem Domain in MDA," *e-Informatica Softw. Eng. J.*, vol. 2, no. 1, pp. 29–46, 2008.
[9] E. Asnina, "The Computation Independent Viewpoint: a Formal Method of Topological Functioning Model Constructing," *Appl. Comput. Syst.*, vol. 26, pp. 21–32, 2006.
[10] J. Osis, E. Asnina, and A. Grave, "MDA oriented computation independent modeling of the problem domain," in *Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering*, 2007, pp. 66–71. https://doi.org/10.1109/SwSTE.2007.20
[11] J. Osis, E. Asnina, and A. Grave, "Formal Problem Domain Modeling within MDA," in *Software and Data Technologies: Second International Conference, ICSOFT/ENASE 2007, Barcelona, Spain, July 22-25, 2007, Revised Selected Papers*, J. Filipe, B. Shishkov, M. Helfert, and L. A. Maciaszek, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 387–398. https://doi.org/10.1007/978-3-540-88655-6_29
[12] A. Šlihte and J. Osis, "The Integrated Domain Modeling: A Case Study," in *Databases and Information Systems: Proceedings of the 11th International Baltic Conference (DB&IS 2014)*, 2014, pp. 465–470.
[13] E. Asnina and V. Ovchinnikova, "Specification of decision-making and control flow branching in Topological Functioning Models of systems," in *ENASE 2015 - Proceedings of the 10th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2015. https://doi.org/10.5220/0005479903640373
[14] G. Schneider and J. P. Winters, *Applying Use Cases: A practical Guide*, 2nd ed. Pearson Education, Inc., 2001.
[15] J. Osis and A. Slihte, "Transforming Textual Use Cases to a Computation Independent Model," in *Model-Driven Architecture and Modeling-Driven Software Development: ENASE 2010, 2ndMDA&MTDD Whs.*, 2010, pp. 33–42. https://doi.org/10.5220/0003043300330042
[16] A. Slihte, J. Osis, and U. Donins, "Knowledge Integration for Domain Modeling," in *Model-Driven Architecture and Modeling-Driven Software Development: ENASE 2011, 3rd Whs. MDA&MDSD*, 2011, pp. 46–56.
[17] D. Leffingwell and D. Widrig, *Managing Softqare Requirements: a Use Case Approach*, 2nd ed. Addison-Wesley, 2003.
[18] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. Mcclosky, "The Stanford CoreNLP Natural Language Processing Toolkit," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2014, pp. 55–60. https://doi.org/10.3115/v1/P14-5010
[19] A. Bies *et al.*, "Bracketing Guidelines for Treebank II Style," 1995.
[20] E. Nazaruka, "Processing Use Case Scenarios and Text in a Formal Style as Inputs for TFM-based Transformations," *Balt. J. Mod. Comput.*, p. submitted, 2019.
[21] S. Jabbarin and N. Arman, "Constructing use case models from Arabic user requirements in a semi-automated approach," in *2014 World Congress on Computer Applications and Information Systems, WCCAIS 2014*, 2014, pp. 1–4. https://doi.org/10.1109/WCCAIS.2014.6916558
[22] I. N. Nassar and F. T. Khamayseh, "Constructing Activity Diagrams from Arabic User Requirements using Natural Language Processing Tool," in *2015 6th International Conference on Information and Communication Systems (ICICS)*, 2015, pp. 50–54. https://doi.org/10.1109/IACS.2015.7103200
[23] H. Krishnan and P. Samuel, "Relative Extraction Methodology for class diagram generation using dependency graph," in *2010 International Conference On Communication Control And Computing Technologies*, 2010, pp. 815–820. https://doi.org/10.1109/ICCCCT.2010.5670730
[24] M. Elbendak, P. Vickers, and N. Rossiter, "Parsed use case descriptions as a basis for object-oriented class model generation," *J. Syst. Softw.*, vol. 84, no. 7, pp. 1209–1223, Jul. 2011. https://doi.org/10.1016/j.jss.2011.02.025
[25] M. G. Ilieva and O. Ormandjieva, "Models Derived from Automatically Analyzed Textual User Requirements," in *Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*, 2006, pp. 13–21. https://doi.org/10.1109/SERA.2006.51
[26] V. Bhala, R. Vidya Sagar, and S. Abirami, "Conceptual modeling of natural language functional requirements," *J. Syst. Softw.*, vol. 88, pp. 25–41, 2014. https://doi.org/10.1016/j.jss.2013.08.036
[27] C.-C. Osman and P.-G. Zalhan, "From Natural Language Text to Visual Models: A survey of Issues and Approaches," *Inform. Econ.*, vol. 20, no. 4, pp. 44–61, Dec. 2016. https://doi.org/10.12948/issn14531305/20.4.2016.05
[28] F. Friedrich, J. Mendling, and F. Puhlmann, "Process Model Generation from Natural Language Text," in *Proceedings of the 23rd International Conference on Advanced Information Systems Engineering (CAiSE 2011)*, 2011, pp. 482–496. https://doi.org/10.1007/978-3-642-21640-4_36
[29] F. Amardeilh, P. Laublet, and J.-L. Minel, "Document annotation and ontology population from linguistic extractions," in *Proceedings of the 3rd international conference on Knowledge capture - K-CAP '05*, 2005, pp. 161–168. https://doi.org/10.1145/1088622.1088651
[30] D. E. Jones, S. Igo, J. Hurdle, and J. C. Facelli, "Automatic Extraction of Nanoparticle Properties Using Natural Language Processing: NanoSifter an Application to Acquire PAMAM Dendrimer Properties," *PLoS One*, vol. 9, no. 1, p. e83932, Jan. 2014. https://doi.org/10.1371/journal.pone.0083932
[31] C. T. M. Hue, D.-H. Dang, N. N. Binh, and A.-H. Truong, "USLTG: Test Case Automatic Generation by Transforming Use Cases," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 29, no. 09, pp. 1313–1345, Sep. 2019. https://doi.org/10.1142/S0218194019500414
[32] E. Sarmiento, J. C. S. P. Leite, E. Almentero, and G. Sotomayor Alzamora, "Test Scenario Generation from Natural Language Requirements Descriptions based on Petri-Nets," *Electron. Notes Theor. Comput. Sci.*, vol. 329, pp. 123–148, Dec. 2016. https://doi.org/10.1016/j.entcs.2016.12.008
[33] C. Wang, F. Pastore, A. Goknil, L. Briand, and Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis - ISSTA 2015*, 2015, pp. 385–396. https://doi.org/10.1145/2771783.2771812
[34] C. T. M. Hue, D. D. Hanh, and N. N. Binh, "A Transformation-Based Method for Test Case Automatic Generation from Use Cases," in *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, 2018, pp. 252–257. https://doi.org/10.1109/kse.2018.8573372
[35] Z. Ding, M. Jiang, and M. Zhou, "Generating Petri Net-Based Behavioral Models From Textual Use Cases and Application in Railway Networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 12, pp. 3330–3343, Dec. 2016. https://doi.org/10.1109/TITS.2016.2518745

Erika Nazaruka received M. sc. in computer systems in 2003 and Doctoral degree (Dr. sc. ing.) in information technology with specialisation in system analysis, modelling and design from Riga Technical University in 2006.
She has been an Associate Professor at the Department of Applied Computer Science of Riga Technical University since 2013. She has also worked 4 years as a Software Developer. She is the author of 60 conference papers, 4 book chapters and 1 book. Her research interests include topological functioning modelling, software quality assurance, model-driven and object-oriented software development, and natural language processing.
The Latvian Academy of Sciences has awarded her and her co-author, Jānis Osis, for the book "Model-Driven Software Development: Architectures and Functions", which was recognised as one of the most significant theoretical achievements of the Latvian science in 2011.
Contact address is the Department of Applied Computer Science, Riga Technical University, Sētas Str. 1, Riga, LV-1048, Latvia.
E-mail: erika.nazaruka@rtu.lv
OIRCID iD: https://orcid.org/0000-0002-1731-989X

Jānis Osis is a Professor at the Faculty of Computer Science and Information Technology, Riga Technical University, Latvia. He holds Dr. habil. sc. ing. degree and is an honorary member of the Latvian Academy of Sciences. The list of publications contains more than 250 titles, including 16 books. During many years, his main research interest was topological modelling of complex systems. Recent fields of interest have been object-oriented system development, formal methods of software engineering, software development within the framework of MDA by means of topological functioning model support.
The Latvian Academy of Sciences has awarded him and his co-author, Erika Nazaruka, for the book "Model-Driven Software Development: Architectures and Functions", which was recognised as one of the most significant theoretical achievements of the Latvian science in 2011.
Contact address is the Department of Applied Computer Science, Riga Technical University, Sētas Str. 1, Riga, LV-1048, Latvia.
E-mail: janis.osis@rtu.lv
ORCID iD: https://orcid.org/0000-0003-3774-4233

Viktorija Gribermane received M. sc. in computer systems in 2015 from Riga Technical University, Latvia.
Currently she is a Doctoral student and Researcher at the Institute of Applied Computer Systems of Riga Technical University. She is the author of ten conference and journal papers.
Her current research interests include topological functioning modelling, model-driven software development, and natural language processing.
Contact address is the Department of Applied Computer Science, Riga Technical University, Sētas Str. 1, Riga, LV-1048, Latvia.
E-mail: viktorija.gribermane@rtu.lv
ORCID iD: https://orcid.org/0000-0002-8368-9362