Baltic J. Modern Computing, Vol. 8 (2020), No. 1, 48-68
https://doi.org/10.22364/bjmc.2020.8.1.03

Processing Use Case Scenarios and Text in a Formal Style as Inputs for TFM-based Transformations

Erika NAZARUKA

Department of Applied Computer Science, Riga Technical University, Sētas Str. 1, Riga, Latvia

erika.nazaruka@rtu.lv

Abstract. TFM (Topological Functioning Model) based transformations start from text fragments as inputs and end with source code. Automated processing of use case scenarios is likely to be more predictable than text in a formal style thanks to their structure. The goal of the research is to understand whether the differences in processing these two text forms are essential for getting core elements of a TFM, or even a structured form has essential limitations. The theoretical results illustrate that use case specifications may have more structured and less structured formats. Even in the former format, use case steps may contain explanations and even text fragments in a formal style that increases unpredictability. Analysis of text in the both cases requires part-of-speech tagging, lemmas, constituency and dependency parsing, coreference resolution, and language pattern matching. Thus, structuring the initial documents is questionable but cases when they are to be managed in projects.

Keywords: use cases, text in a formal style, natural language processing, topological functioning model, computation independent model, model transformation, knowledge acquisition

1. Introduction

Quality of an input model for a chain of model transformations is crucial. Transformation of models is one of the key elements in Model Driven Software Development (MDSD). The flagman of the MDSD is a framework called Model Driven Architecture (MDA) proposed by the Object Management Group (OMG) in 2001 (Miller and Mukerji, 2001). The MDA considers development of an architecture of a software system as a chain of enhancing transformations of models starting from computation independent to platform independent to platform specific models. In this view, the computation independent model (CIM) is a starting point or an input that further is enhanced with application logic and details specific to selected platforms. In an ideal case, a CIM must contain complete unambiguous knowledge on a problem (or business) domain or, in practice, it should be at least modifiable and keep integrity of knowledge.

The CIM represents a problem (or business) domain from a computation independent viewpoint (Miller and Mukerji, 2001). This means that it is independent from "computational" particularities that have origin in application behaviour (or logic) and functionality of platforms. In the light of this, CIM representation means also must be computation independent. They could be descriptions in prose (instructions, position descriptions, interview protocols), descriptions in structured text (e.g., use case scenarios, user stories, templates for requirements), graphical schemes (e.g., use case models, business process models in different notations such as BPMN (Business Process Model and Notation), structural models in different notation such as Entity-Relationship diagrams) and mathematical or physical formulas. All these means do not include any computation *dependent* information.

An ideal CIM contains complete unambiguous knowledge. However, CIM completeness and unambiguity of representations are questionable since descriptions inherit natural language ambiguity, as well as schemes usually provide a fragmentary view on a problem domain or represent just one or several aspects of it. There must be a model that can serve as a ground onto which gathered knowledge could be projected and verified and as a starting point for further automated transformations.

A model that has these abilities is a Topological Functioning Model (TFM). The TFM is based on principles of system theory and algebraic topology. It specifies a system in a holistic manner, showing its inner functionality and interaction with external systems at the high level of abstraction. The TFM can be manually (but according to precise rules) transformed into most used UML diagram types: class diagrams, activity diagrams, use cases with their specifications (Osis and Asnina, 2011a) and Topological UML (Donins et al., 2011) diagrams such as Topological Class diagrams, Topological Use Case diagrams, Activity diagrams, State Chart diagrams, Sequence and Communication diagrams (Osis and Donins, 2010). Application of the principles of the theoretical foundation of the model leads to discovering complete knowledge and verifying its accuracy.

Sources of knowledge for the TFM differ in their representation formats, structure or its absence, and have different readiness for automated processing and projecting to the TFM. Automated processing of input sources is crucial since additional modelling requires additional resources like staff, budget and time. Automated processing in comparison with manual allows reducing time needed. However, what knowledge and of what quality could be processed and projected to the TFM is a question that requires additional research. The aim of this research is to clarify what an input form of knowledge sources, a text in a formal style or a use case scenario, has essential advantages for the TFM at the present. In this research focus is put on automated processing that includes less parsing and transforming notation elements but more application of Natural Language Processing (NLP) for acquiring and verifying knowledge from the corresponding text.

The paper is organized as follows. Section 2 presents brief overview of the TFM and its place and role in TFM-based transformations. Section 3 discusses initial theoretical results on natural language processing in prose in a formal style and in a numbered step form of use case scenarios. Section 4 gives a brief overview in related work. Conclusion presents main results and speculations on issues and further research.

2. Topological Functioning Model Based Transformations

2.1. Chain of Transformations

The meta-picture of the TFM driven transformations (Figure 1) illustrates a general vision of the TFM driven transformations. There are two groups of the input, i.e., knowledge about a problem domain or system's processes and data and knowledge about required processes and data of a corresponding sub-system. Pairs of a system and its subsystem can be, for example, an organization and its information system, or an information system and its implementation in software. Sources of knowledge on the system's processes and data usually are presented in official documents, instructions, specifications, interviews with domain experts and so on, thus in text in a formal style. Requirements to the sub-system's processes and data are presented in a form of structured text such as requirements specifications, use case scenarios, user stories and features.

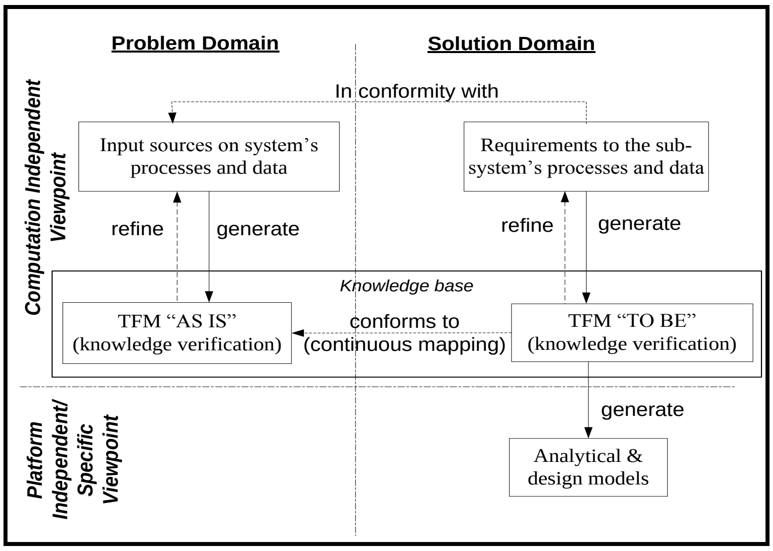
Problem Domain
Solution Domain
Computation Independent Viewpoint
In conformity with
Input sources on system's processes and data
Requirements to the sub-system's processes and data
refine
generate
refine
generate
Knowledge base
TFM "AS IS" (knowledge verification)
conforms to (continuous mapping)
TFM "TO BE" (knowledge verification)
Platform Independent/ Specific Viewpoint
generate
Analytical & design models

Figure 1. Meta-picture of TFM-driven transformations at the CIM and PIM/PSM levels

Taking knowledge from the first group, the TFM "as is" that describes the current situation is created (the left side). Taking requirements to the solution, the TFM "to be" that describes the desired processes and data is created (the right side). The two models are continuously mapped that means that changes in one model are projected onto the second. This allows checking requirements against existing functionality in the system. After that, the computation independent viewpoint on the solution domain is transformed to the platform independent (or specific) one. During this transformation, knowledge about the functionality and data are transformed onto the application constructs shown in Table 1 (Nazaruka and Osis, 2019). According to (Osis and Donins, 2017), the TFM can be transformed to the following Topological UML (that is an UML extension) diagrams: topological class diagrams, topological use case diagram, communication diagrams, and object diagrams; state diagrams; component and deployment diagrams. The final step of the transformations is generation of source code for selected platforms.

In both cases creation of the TFM is manual at the present, but the vision is to generate the TFM from available texts using NLP techniques. The initial results of research on application of NLP techniques to the descriptions showed that it is not enough to deal just with basic NLP operations such as tokenization, part-of-speech tagging, chunking and Name Entity Recognition. In order to analyse text in prose analysis of dependencies between clauses, in complex noun phrases, in predicates and in verb phrases is required as well as one of the most difficult tasks, i.e., discourse analysis in the text (Nazaruka and Osis, 2018). Besides that, processing of the text in prose (in a formal style) has issues related to a technical side and to particularities of a natural language (Nazaruka et al., 2019). Thus, parsing models and NLP outcome representation formats as well as a lack of needed knowledge, different structures of sentences and implicit synonyms may substantially affect the result.

Table 1. Tracing TFM elements into elements of software architecture (Nazaruka and Osis, 2019): FR – a functional requirement, NFR – a non-functional requirement

Requirements	Elements in TFM	Application constructs in UML
FR and NFR	Action A	Activities, operations, messages, events, entry and exit effects
FR and NFR	Object O_i	Classes, objects
FR and NFR	Result R_i	Classes, objects, states, associations between certain classes
FR	Precondition c_i	Guards in behavioural diagrams, states
FR	Postcondition c_i	States
FR	Providers Pr_i	Actors, classes, subject
FR	Executors Ex_i	Actors, classes, objects
FR	Subordination S	None
Dependencies among FRs, and NFRs	Cause-and-effect relation T_i	Topological relationships, structural relationships, control flows, transitions
Dependencies among FRs, and NFRs	Functioning cycle	Topological relationships, structural relationships, control flows

The uncertainty of a natural language can be partially solved either by using machine learning or manual pre-processing of knowledge, e.g., using specifications of use case scenarios or user stories and exhaustive software requirements. Research on NLP techniques used for discovering cause-effect relations in texts in prose showed two clear trends (Nazaruka, 2019). The first is increasing the accuracy of the results using ontology banks, machine learning and statistical inferring. The second is decreasing the cost of these activities. In case of construction of software models, the main challenge is a lack of corpuses and statistical datasets for potential problem domains. Nevertheless, this issue can be potentially solved by limitation of those source documents to specifications (requirements, scenario, etc.) having less variability in expressing causality.

2.2. Topological Functioning Model in a Nutshell

The TFM is a formal mathematical model that was proposed by Janis Osis at Riga Technical University in 1969. It allows modelling and analysing functionality of the system (Osis and Asnina, 2011b). At the beginning this model has been invented for mathematical specification of functionality of complex *mechanical* systems (Osis and Asnina, 2011b). However, the system can be business, software, biological, mechanical or represent 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.

TFM models can be compared for similarities using a continuous mapping mechanism (Asnina and Osis, 2010). Since 1990s the TFM has been elaborated for software development (Osis et al., 2008a) starting from principles of object-oriented system analysis and design and ending with principles of the MDA.

The TFM characteristics can be divided into topological and functioning properties (Osis and Asnina, 2011b). The topological properties take their origin in topological algebra. They are connectedness, neighbourhood, closure and continuous mapping.

Connectedness ensure that all functional characteristics of the system depend from 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's 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 run (or function), these dependencies form a cycle (or cycles) of functionality. Behaviour of the system depends on input signals from the external environment as well as of output signals of the system (reaction) to the external environment.

The composition of the TFM is presented in (Osis and Asnina, 2011b). 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 (Asnina, 2006; Osis et al., 2007, 2008b). 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 (Šlihte and Osis, 2014).

The main TFM concept is a functional feature that represents a system's functional characteristic, e.g., a business process, a task, an action, or an activity (Osis and Asnina, 2011b). It can be specified by a unique tuple (1).

$$\langle A, \mathbf{R}, \mathbf{O}, \mathbf{PrCond}, \mathbf{PostCond}, \mathbf{Pr}, \mathbf{Ex} \rangle \tag{1}$$

where (Osis and Asnina, 2011b):

- A is an object's action,
- **R** is a set of results of the object's 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) which 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) which 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 (Asnina and Ovchinnikova, 2015; Osis and Donins, 2017). 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, at the beginning the elements of the functional features must be extracted from text. And then, the cause-effect relations between them must be identified using discourse analysis of the text.

3. Text in a Formal Style and Use Case Scenarios as Inputs

Let us consider several types of sources of knowledge that are used in software development and evaluate their suitability as inputs for composing the TFM using NLP outcomes. The formats are text in prose in a formal writing style and structured text in a form of use case scenarios.

3.1. NLP using Stanford CoreNLP

The Stanford CoreNLP toolkit (Manning *et al.*, 2014) contains components that deal with tokenization, sentence splitting, part-of-speech tagging, morphological analysis (identification of base forms), NER (Name Entity Recognition), syntactical parsing, coreference resolution and other annotations such as gender and sentiment analysis. The NER component recognizes names (PERSON, LOCATION, ORGANIZATION, MISC – miscellaneous) and numerical (MONEY, NUMBER, DATE, TIME, DURATION, SET) entities. Phrases can be parsed using both constituent and dependency representations based on a probabilistic parser that is more accurate according to the parsers that relate to some predefined structures. Discovering basic dependencies helps in identification of actions and corresponding objects, results, modes (that can serve for identification of causal dependencies), executors and providers. Besides that, the Stanford CoreNLP implements mention detection and pronominal and nominal coreference resolution that helps in dealing with pronouns and noun phrases that denote concrete phenomena.

For the given research Stanford CoreNLP version 3.9.2 is used. For POS tagging it uses tags listed in Penn Treebank II (Bies *et al.*, 1995). In this research the following tags are mentioned: S – simple declarative clause, NN – noun, single, NNS – noun, plural, NP – noun phrase, PRP – preposition, RP – particle, 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.

Formal descriptions of patterns in Section 3.2 make a use of additional elements: '|' is used for a sequence of alternatives; '[+ sub-pattern]' – for optional elements; arrows '→*edge_name:modifier*→' – for edges between elements in the dependencies analysis where 'element →' is a source element and '→ element' is a target one; round brackets

‘( )’ contains elements in a noun phrase or verb phrase according to results of constituency parsing of text fragments.

3.2. Identification of Core Elements of Functional Features

Formal Text in Prose. Results of processing text in prose (in a formal style) using Stanford CoreNLP (Nazaruka et al., 2019) showed that one of the main difficulties is a variety of sentence structures that could express one and the same knowledge. Application of Stanford CoreNLP with explanatory examples is described in more detail in (Nazaruka et al., 2019) and is not discussed here.

Identification of *action A* is searching instances of the pattern S(VP(VBZ|VBP|VBD|VBN|VBG|VB v_i [+ →*compound:prt*→RP *particle*]) → *nsubjpass*|*dobj* → NP(NN|NNS|PRP n_1)). The result v_i must be returned in the infinitive form. The *particle* is optional.

Identification of *executors **Ex*** is searching instances of the pattern S(VP(VBZ|VBP|VBD|VBN|VBG|VB v_i [+ →*compound:prt*→RP *particle*]) → *nsubj*|*nmod:agent* → NP(NN|NNS|PRP n_i)). The result n_i must be returned in its original form.

Identification of *objects **O** and results **R*** is searching of a direct object of the verb. There are two patterns for search. The first one is applicable for the active voice, since there is a direct object, i.e. a noun 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)) for v_i. The second one is applicable for the passive voice, when a noun fits the pattern S(VP(VBZ|VBP|VBD|VBN|VBG|VB v_i [+ →*compound:prt*→RP *particle*]) → *nsubjpass* → NP(NN|NNS|PRP n_1)). After identification of the linked noun, the object and the result must be determined.

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

- 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 O_i is equal to n_2.
- 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.
- 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.

Otherwise, if VP(v_i) **is linked** with another NN|NNS|PRP n_2 by *nmod*: but *nmod:agent* then the following is true:

- The object O_i is equal to n_2 from PP(NP(n_2)).
- The result R_i is equal to NP(n_1) + IN *preposition*, where *preposition* is in the PP(NP(n_2)).

Processing sentences in prose even in a formal style has difficulties, since a natural language admits short incomplete sentences and complex noun phrases.

As a result, A, **O**, **R**, **Ex** will be obtained together with the corresponding noun prepositions and verb particles. It can help in identification of structural relations between domain objects (entities) in the future. Providers **Pr** are hard to be identified, because they fit the same pattern as the executors.

Preconditions **PreCond** and postconditions **PostCond** can be expressed using multiple natural language constructs; therefore, in prose they must be identified during the so-called discourse analysis (Section 3.3).

For illustration, let us consider several sentences. "The customer pays out the debt" (Figure 2) is in the active voice and very simple. It fits the pattern for action A, which is a verb "pay out". An executor is a noun that matches the pattern with *nsubj* (nominal subject) edge, i.e., "customer". Matching patterns for objects and results, n_1 is a noun linked with "pays" using *dobj* (direct object) edge, i.e., "debt". Then, following the analysis logic that "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", the result is the object "debt" and the empty result.

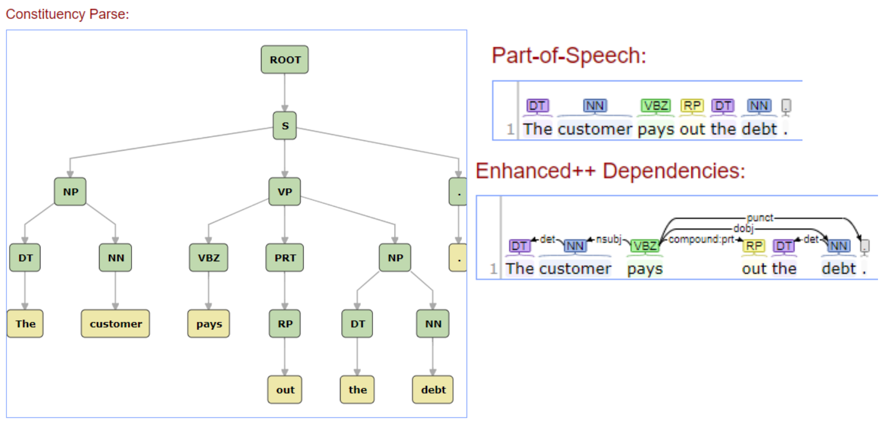

Figure 2. Processing the sentence in the active voice

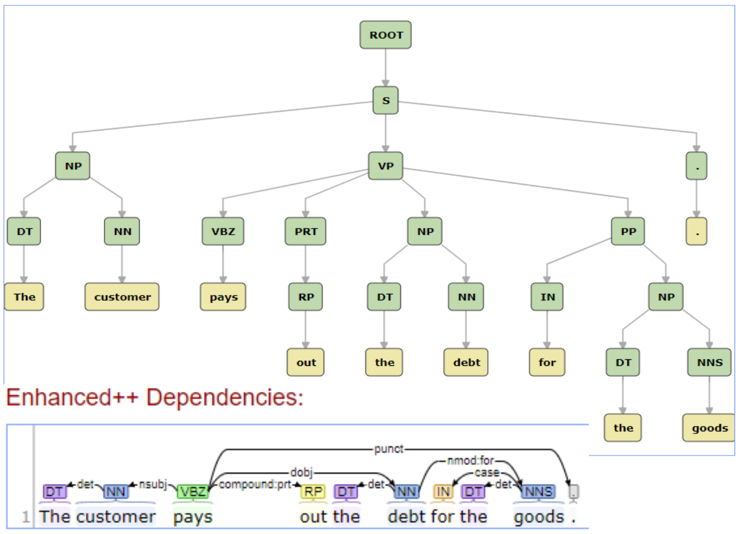

Figure 3. Processing the sentence in the active voice with the compound direct object

"The customer pays out the debt for the goods" is a light modification of the first sentence (Figure 3). Here, the action A and the executors are the same. But while analysing the direct object one can found that it fits the third pattern: If VP(NP(n_1)→ nmod:poss|of|to|into|from|for→ NP(n_2)) then *Ri* is equal to the NP(n_1)→[NP(n_2)→] *case*→IN *preposition*. The object *Oi* is equal to *n2*. Thus, the object is a noun "goods" and the result is "the debt for".

The next sentence in the passive voice "The goods debt is paid out by the customer" (Figure 4) fits the pattern for a verb linked with a noun n_1 by *nsubjpass*. The action is the verb "pay out". The executor fits the pattern with *nmod:agent* and is the noun "customer". The noun n_1 "debt" fits the pattern in the rule "If NP(*n1* →*compound*→ n_2), then *Ri* is equal to the NP(n_1)+" of"; and the object *Oi* is equal to n_2". Hence, the result is "the goods debt of" and the object is "goods". The whole noun phrase is kept in order not to lose some relevant knowledge.

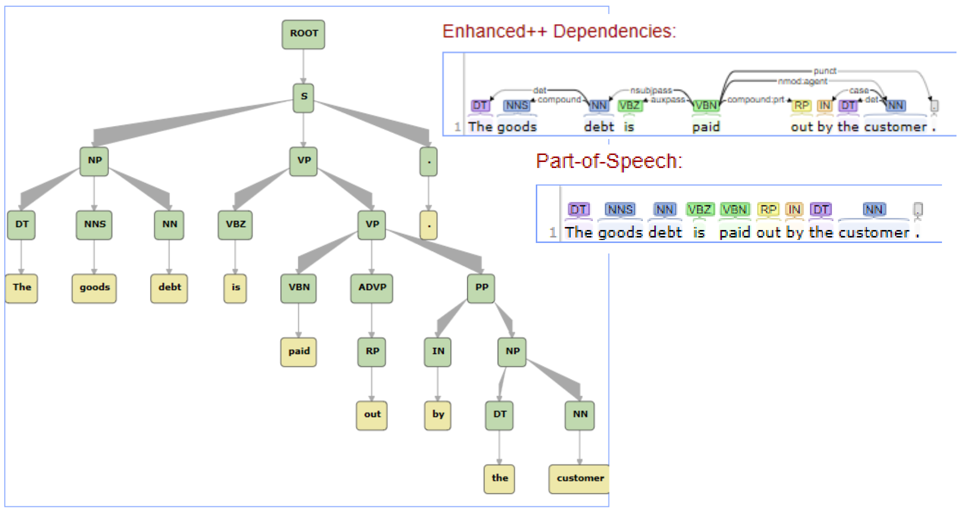

Figure 4. Processing the sentence in the passive voice with the compound direct object

Certainly, noun phrases can also contain adjectives and adverbs indicating characteristics of objects those can lead to further specialization of them. The same is with verb phrases that may include modifiers (such as may, could, should and so on) indicating on obligation of an action. Besides that, verb phrases can form dependencies between clauses of a complex sentence. For example, lets us take two sentences "The customer pays out the debt to close its credit obligations" and "The customer pays out the debt and closes its credit obligations" (Figure 5).

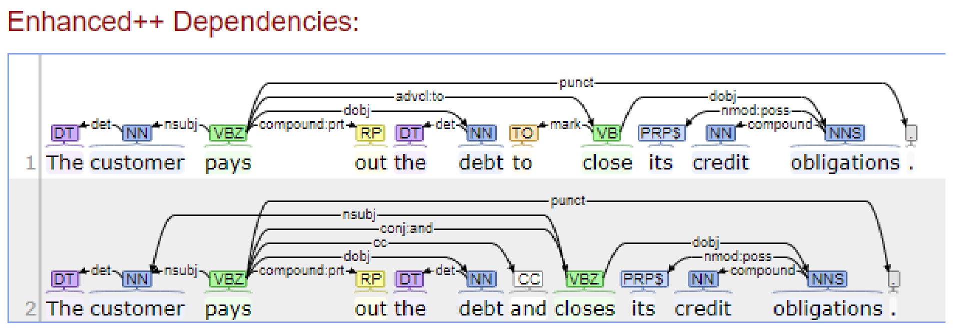
Enhanced++ Dependencies:
1 The customer pays out the debt to close its credit obligations .
DT NN VBZ RP DT NN TO VB PRP$ NN NNS
det nsubj compound:prt dobj det advcl:to mark punct dobj nmod:poss compound
2 The customer pays out the debt and closes its credit obligations .
DT NN VBZ RP DT NN CC VBZ PRP$ NN NNS
det nsubj compound:prt dobj det cc conj:and nsubj punct dobj nmod:poss compound

Figure 5. Dependencies analysis in the complex sentences

The former indicates the intention of the action “pay out” to perform the action “close” using the adverb clause, the latter indicates a logical sequence of the actions “pay out” and “close”. In both cases two functional features will be extracted. However, according to the current sequence of patterns the former will lose the executor of the action “close”. In this research, such constructs are omitted.

Use Case Scenarios. Use cases can be expressed in both informal and structured manners (as a flow of numbered steps, in a table form, or as a diagram). Processing of the former is the same as processing of the text in prose. Processing of the latter is a combination of NLP tasks and scenarios structure analysis.

Use case scenario structures may have different forms. For example, one of the most completed forms is presented by Winters and Schneider (Schneider and Winters, 2001), where it has the following parts: a use case name, a brief description, a context diagram that is a part of the entire use case diagram, preconditions of a use case, a flow of events that includes a basic path and alternative paths, postconditions of a use case, a subordinate use case diagram, subordinate use cases, an activity diagram for the flow of events, a view of participating classes, sequence diagrams, a user interface, business rules, special requirements, other artifacts, and outstanding issues. In this research, manually proceeded scenarios in the numbered step form are considered.

A use case scenario in this form usually has predefined *keywords* such as “the use case begins when” – for an entry point into the use case, “for each… end loop” – for iterations, “and the use case ends” – for an exit point form the use case, “basic path” – a title of the section of the normal flow of steps, “alternative paths”- a title for the section of branches in the execution logic, “Alternative <number: <explanation>” – a title of a branch in the execution logic, “special requirements” – a section for non-functional requirements, as well as others used in a project. Thus, analysis of a use case scenario starts from identifying the structure, the keywords and then analysing the corresponding descriptions.

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 the description of the functional feature *D* that is the *verb phrase VP* from the text of a step in a scenario (Osis and Slihte, 2010; Slihte et al., 2011).

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 (Schneider and Winters, 2001; Leffingwell and Widrig, 2003).

Parsing in the IDM is done according to these steps:

- Identify *coordinating conjunctions* to split a sentence into several clauses, and, thus, several functional features;
- Identify the *verb phrase* (VP tag) in a clause that is considered as a union of action A, object O and result R (if it is indicated) and forms the so-called *description* of the functional feature *D*;
- Identify the *noun phrase* (NP tag) that is marked as executor Ex_i if it meets the same noun in the list of actors for the use case;
- Preconditions and postconditions 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 Ex_i. In order to extract those elements from *D*, the following active voice patterns must be analysed.

Identification of *action A* is searching instances of the pattern S(VP(VBZ|VBP|VBD|VBN|VBG|VB v_i [+ →*compound:prt*→RP *particle*]) → *dobj* → NP(NN|NNS|PRP n_1)). The result v_i must be returned in the infinitive form. The *particle* is optional.

Identification of *executors* ***Ex*** is searching instances of the pattern S(VP(VBZ|VBP|VBD|VBN|VBG|VB v_i [+ →*compound:prt*→RP *particle*]) → *nsubj* → NP(NN|NNS|PRP n_i)). The result n_i must be returned in its original form. This means that a number of executors is greater than a number of actors, since the system is an executor but is not an actor. Providers **Pr** fits to the same pattern.

Identification of *objects **O** and results **R***. First, a noun 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)) for v_i must be found. Then, the object and the result must be determined in the same way as in the formal text in prose.

Preconditions **PrCond**, if they are specified for use case steps, usually has a predefined form "IF/WHEN<condition> THEN: step(s)". Therefore, a clause S can be checked for this pattern. The text between IF/WHEN and THEN part must be taken as a precondition to a functional feature or the first feature in the block of steps. Sometimes, the text may contain additional functional characteristic and form a separate functional feature. Another difficulty is that the IF/WHEN…THEN form is not prescribed, then analysis of conditionals is required that is not a trivial case (Section 3.3).

Postconditions **PostCond** in a use case describe the state of the system after the step or the use case execution. A postcondition must be expressed explicitly, otherwise it must be inferred from the context that also is not a trivial case (Section 3.3).

3.3. Identification of Cause-Effect Relations

Cause-effect relations are binary topological relations between functional features. They represent that successful termination of one functional feature, a cause, triggers initiation of another, causally dependent, functional feature called an effect. Cause-effect relations may form groups of incoming and outgoing relations. A group may have subgroups of the relations joined by using one of logical operators AND, OR, or XOR.

Connection between a cause and an effect is represented by a certain conditional expression, the causal implication. It is established by nature or rules. In causal connections "something is allowed to go wrong", whereas logical statements allow no exceptions. Using this property of cause-effect relations a logical sequence, wherein the execution of the precondition guarantees the execution of the action, can be prescinded. This means that even if a cause is executed, the corresponding effect can be not generated because of some functional damage (Nazaruka, 2019).

Formal Text in Prose. Cause-effect relations can be expressed implicitly and explicitly. The thorough analysis of implicitly and explicitly expressed relations (Khoo et al., 2002) illustrated that the latter relations use causal links, causative verbs, resultative constructions, conditionals as well as causative adverbs, adjectives and prepositions. The former relations are usually inferred by a reader associating information in the text with their background knowledge (Khoo et al., 2002; Solstad and Bott, 2017; Ning et al., 2018).

Theories for identification, modelling and analysis of cause-effect relations have their origin in psycholinguistics, linguistics, psychology and artificial intelligence (Waldmann and Hagmayer, 2013). According to Waldmann and Hagmayer, those of theories attempting to reduce causal reasoning to a domain-general theory can be grouped as associative theories, logical theories and probabilistic theories. Each group has limitations in identification of causes and effects. However, logical theories seem to be more suitable to software development in processing verbally expressed information, since they model causal reasoning as a special case of deductive reasoning. Logical theories frequently analyse conditionals (if/when…then constructs) in the text. Although conditionals do not distinguish between causes and effects. In the formal text (instructions, descriptions of processing, etc.) they usually have a form "if/when <a cause occurs> then <an effect occurs>" as well as temporal priorities can be helpful in distinguishing them (Solstad and Bott, 2017; Pearl, 2019). Unfortunately, if/when…then constructs can be also used for simple sequential storytelling without any causality between parts of the sentence. Moreover, they may form *counterfactual* conditionals (with words *might*, *would*, *if only*) that are hard for NLP analysis (Solstad and Bott, 2017), e.g., "If a librarian would not have ordered a book, a manager assistant would have".

Thus, processing and analysis of causes and effects in the text in prose must be performed at clause/sentence and discourse levels as well as conditionals and temporal reasons must be checked

Let us consider each level in more detail (Table 2):

- A clause is a group of verbs that includes at least a subject *and* a verb. A clause can be independent and express a complete thought. An independent clause is considered as a standalone sentence or as a part of the sentence of several clauses. A dependent clause can act as a noun, an adjective, or an adverb. A sequence of sentences forms a discourse. At the clause/sentence level a cause-event and an effect-event are analysed (Solstad and Bott, 2017), while at the discourse level one deals with propositions instead of events (Kang *et al.*, 2017; Solstad and Bott, 2017). Causal verbs are used for event identification, while causal links are suitable for identification of causal dependencies between clauses and sentences.

Table 2. Processing and analysis of causes and effects in the text in a formal style

Processed parts	Parts-of-speech, constructs, patterns
Clause/sentence	Pattern: [[event1]] CAUSE [[event2]] More researched constructs are: - causal links: adverbial (*so, hence, therefore, etc.*), prepositional (*because of, on account of, etc.*), subordination (*because, as, since, so…that, etc.*), clause-integrated (*that is why, the result is/was, etc.*); - causative verbs (including action verbs as a part of this verb group). Less researched are: - resultative constructions (a state of the direct object after the action); - causative adverbs (e.g., *successfully, consequentially, mechanically*), adjectives and prepositions; - multiple causes and effects.
Discourse	[[proposition1]] CAUSE [[proposition2]]; Connections between causal relations expressed either by causal links or implicitly by human inferring of reasons and explanations (sometimes even in the same sentence).
Conditionals	*If/When…then constructs* and *counterfactual* conditionals that indicate the possible state of the world in case of an action.
Temporal reasons	Joint consideration of causal and temporal models helps in correct identification of counterfactual clauses as well as more valuable identification of causal relations.

- Conditionals *If/When…then* may either fit the pattern with events or indicate a state of the object before some event. In the former case, the clause of event 1 will be a cause functional feature for event 2. In the latter case, a phrase from the *IF/When* part will be a precondition for the functional feature of event 2.
- Attention to analysis of temporal aspects has become greater since 2016 (Mirza, 2014; Asghar, 2016; Mostafazadeh et al., 2016; Ning et al., 2018). Temporal relations may indicate hidden causality, not only simple sequence of events, especially for counterfactual clauses (Ning et al., 2018).

Use Case Scenarios. In the scenarios in the numbered step form, topology is determined according to the sequences of steps specified in flows of the scenarios and explicitly indicated dependencies between the steps and subordinated use cases. Analysing different alternative forms for a use case specification (Schneider and Winters, 2001; Leffingwell and Widrig, 2003), it can be concluded that the causal dependencies can be expressed by the following means:

- A sequence of steps. Each step has its number and represents an action (or event) that must be done. Thus, successful termination of each preceding step initiates its direct subsequent step. In other word, causality is a logical sequence of steps.
- Redirection to the indicated step. The redirection may be expressed either using some predefined phrases, e.g., "the use case continues at <flow> step <number>", or another phrase with the similar meaning.
- Redirection to an alternative flow. There are two possibilities. If an alternative flow takes only a few sentences, it can be located directly within the flow-of-events section. Otherwise, a separate section is recommended. In the latter case the transition can be expressed with the similar phrases as in the previous point. In the former, a group precondition indicates an alternative sequence of steps,

where causality is the sequence. However, sometimes an alternative flow is not structured and then analysis of sentences or discourse is required.

- 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 an alternative starts, is indicated in the alternative itself using phrases like "In step <number>, <precondition>, <step/event>".
- Redirection to a subordinated use case (included and extending). An included use case has its own name that is used as a marker with a keyword "include" to express a transition to it. 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 also must be indicated.
- Cycle constructs. In order to indicate iterative sequences constructs *For each <element> … end loop* and *While <action/event>… end loop* are used.
- A dependent clause. This clause can be inferred by analysing the causal links (Table 2): adverbial, prepositional, subordination and clause integrated. For example, "The customer enters a number of the product to make an order". A clause "to make an order" acts as an adverb. But "making an order" is a separate event that must be analysed whether it is a cause for the "entering" or it is just an explanatory statement.
- A dependency link to another independent use case. In a typical template such a link is not specified. However, if this dependency is indicated, a name of another independent use case is located in the certain section of the specification.

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

3.4. Summary

Table 3 summarizes what must be analysed in the both formats. The benefit of a structured scenario lies mostly in identification of causal dependencies between sentences, postconditions, preconditions (less), and iterations. In other words, results of analytical work done by a human and expressed by certain structures is convenient for parsing.

However, huge work on NLP still remains. Identification of an action, objects and results, executors, and providers requires part-of-speech tagging, lemma analysis, tokenization, constituency and dependency parsing and analysis. Besides that, a description of a step in a scenario may be quite detailed and include explanations and sequential actions. Thus, such parts must also be analysed using sentence and discourse analysis.

Taking into consideration the results, benefits of a structured form for automated processing are evident but are not critically essential. The reason is that requirements (even in a scenario) are written by a human in a natural language. The large part of essential knowledge may be described there and is the same dependent on the quality of NLP. Thus, benefit of converting existing documents to a more structured form, especially when constructing a TFM of the existing system, is questionable. Nevertheless, use cases have proved themselves as a good means of capturing requirements. They contain knowledge already inferred by a human that makes it easier to extract it in an automated way. This means that they can be successfully used for construction of a TFM of the planned subsystem.

Table 3. Extracting TFM elements from text fragments using NLP and structure parsing (SP)

TFM element	Text in a formal style	Use case scenario
Functional feature <v_i, R, O, PrCond, PostCond, Ex, Pr>		
Action	NLP — a verb v_i from VP with its particle linked with a noun n_1 from NP with a link *nsubjpass* or *dobj.*	NLP — a verb v_i from VP with its particle linked with a noun n_1 from NP with a link *dobj.*
Object and Result	NLP — related noun phrase	NLP — a related noun phrase
Preconditions	NLP — discourse analysis, conditionals (*SP). Could be omitted in the sentence.	SP — a part “Preconditions”. NLP — conditionals (*SP).
Postconditions	NLP — discourse analysis. Could be omitted in the sentence.	SP — a part “Postconditions”
Executors	NLP — a noun n_i that is a target of a link *nsubj* or *nmod:agent* from a verb v_i. Could be omitted in the sentence.	NLP — a noun n_i that is a target of a link *nsubj* from a verb v_i. Must be presented in the first sentence of a step but could be omitted in the sequential sentences within the step.
Providers	NLP — discourse analysis. May fit to the pattern for executors.	NLP — “the system” or another nickname used in the specification. May fit the pattern for executors.
Cause-effect relation		
Sentence	NLP — the pattern ‘<event> CAUSE <event>’; causal links, causative verbs, adverbs, adjectives, preconditions; resultative constructions.	NLP — a dependent clause, cycle constructs (see Iterations).
Discourse	NLP — the pattern ‘<proposition> CAUSE <proposition>’; causal links, explanations, inferring; temporal reasoning.	SP — a sequence of steps; a dependency link to another independent use case. NLP or parsing — a redirection to an indicated step, an alternative flow, or a subordinate use case, or a redirection from an alternate flow; cycle constructs (see Iterations).
Iterations (repetitive action)	NLP — language constructs analysis; inferring from the context.	NLP or SP — for each <element>…end loop; while <action/event>…end loop.

4. Related Work

Looking at the CIM as an input for further transformation to analytical or design models, descriptions in prose, structured text, graphical schemes for behavioural and structural models are wide used. The main attention in recent research is put on graphical schemes.

Taking as an input graphical schemes it is possible to get a design model in UML. For example, Data Flow Diagram (DFD) can be transformed to use case diagrams, activity diagrams, sequence diagrams and domain diagrams, which are the base for further obtaining of class diagrams (Kardoš and Drozdová, 2010). The transformation to behaviour diagrams allows correct mapping to control flows between activities, messages between objects, but a mapping to domain diagrams is incomplete. This approach allows defining concepts and navigations among them, but information about structural relationships and multiplicity must be added by a modeler.

Transformation from BPMN models to use cases (Kriouile et al., 2015) to behavioural and domain classes models resulted in complete acquisition of control flows and message flows, however, the domain classes model contained only aggregation relationships obtained from the BPMN pools and lanes (Kriouile et al., 2014). Using *structural* business rules allows keeping knowledge about terms and facts, as well as relations among them (Bousetta et al., 2013), thus getting necessary static knowledge such as names of classes, compositions and aggregations among them, generalization/specialization relationships, navigations, and multiplicity in associations in semi-automatic way. The business rules are expressed by natural languages and supplemented by Object Constraint Language (OCL). Transformation from BPMN diagrams to UML class diagrams and state diagrams for each class presented in (Mokrys, 2012) also requires additional participation of a modeler in order to refine relationships among classes.

Transformation from use cases and activity diagrams to a class diagram, where control flows of the activity diagrams are transformed to bidirectional navigations with many-to-many multiplicity in the class diagram still requires human participation to refine interclass relationships (Rhazali et al., 2015, 2016). Transformation from a business process model (containing both manual and automated activities as well as data objects) and requirements models in a form of use cases to be automated to support the business activities expressed in terms of the activity diagram (Kherraf et al., 2008) into a process component that is linked with various entity components supplemented with roles applies a set of patterns and four target archetypes: Moment-Interval that usually corresponds to a process component, and PPT (Party, Place, Thing), Role and Description that correspond to an entity component. A use case model extended with data objects and business rules in an alternative to the natural language, SBVR (Semantic of Business Rules and Vocabulary), similarly to (Bousetta et al., 2013) are transformed to the class diagram. However, elements in it are linked with bidirectional associations and require additional refinement (Essebaa and Chantit, 2016).

Another more advanced approach implemented in a tool called ReDSeeDS (Requirements-Driven Software Development System) was developed during the European ICT project (2008-2012), which main objective was solving a problem of the complexity of requirements descriptions (ReDSeeDS, 2020). Requirements in this tool are presented as use case scenarios, where a developer may manually indicate nouns, noun phrases, verb phrases and assign their meaning in a domain model. According to these marks the tool automatically creates actors, classes and methods. The obtained model can be transformed to code using predefined transformation patterns. The code is executable (it has presentation, controller and model layers); however, it should be manually supplemented (Kalnins et al., 2011; Smialek and Straszak, 2012). The idea implemented in the tool is very similar to the researched one with one distinction – identification of nouns, verbs, corresponding phrases and their meanings must be conducted in an automated way.

Thus, the static viewpoint of the system represented as a [domain] class diagram and proposed in many approaches is limited with relationships obtained from control flows. It is possible to derive aggregation and composition (from BPMN models), and (intuitive) bidirectional navigation between domain classes. More advanced characteristics such as a specific navigation, multiplicity and roles in associations as well as generalization/specialization must be added manually or explicitly defined in business rules specified in formalized (or controlled) natural language, i.e. by using a predefined subset of a natural language or in the form of SBVR statements. Thus, most of these approaches are focused on processing graphic structures or pre-defined structures for knowledge (as in SBVR).

In recent years, analysis of informal and semiformal texts gained significant results using a deep learning approach. The open-source SUMMA platform (Germann et al., 2018) offers extraction and storage of factual claims from recorded live broadcast, spoken contents and text as well as storyline clustering and cluster summarization. The text is generated from semantic graphs produced by the parsing module. Identification of text is based on semantic-syntactic valence patterns recognition extracted from FrameNet annotated corpora (Dannélls and Gruzitis, 2014). The idea of the use of the semantic-syntactic valence patterns is interesting and in general features echoes our idea of text processing. However, patterns for TFM elements must be more refined.

5. Conclusion

Processing of use case scenarios has its benefits thanks to preliminary analytical work done by a human and results of it expressed in a structured form. The advantage is more predictable identification of causal dependencies between functional features, postconditions, preconditions (to a lesser extent), and cycle structures. However, a scenario may contain expanded explanations, details, even sequences of events and small alternative flows. This means that natural language processing could not be omitted for identification of actions, objects, results, preconditions, executors and providers as well as a sub-set of cause-effect relations. Moreover, NLP tasks are the same as in case of text fragments in a formal style, i.e., part-of-speech tagging, lemma analysis, tokenization, constituency and dependency parsing and analysis.

Results of identification of those elements are used in composing a TFM of the system “as is” and a TFM of its planned sub-system or sub-systems “to be”. TFM elements helps in discovering the same functionality as well as similarities and differences in behaviour of systems.

Having two independent sources of knowledge, namely, documents in a formal style for the “as is” case and use case scenarios for the “to be” case, and continuous mapping between the TFMs allows projecting functional features to make an analysis of functional coverings, completeness, similarities and differences. This increases quality of extracted knowledge, quality of the built root model, the TFM, that further is to be propagated to the design model and the source code.

Further research direction is closely related to NLP application to text fragments in a formal style. At the beginning, a set of language constructs and patterns for relations among objects participating in an action must be defined. Then, the most unresearched part, namely, cause-effect relations identification at the sentence and discourse levels must be solved as well as multi causes and multi effects relations. The potential results may be used for event flows identification in text and automated creation of use cases and user stories (or other similar representation formats).

List of abbreviations used

MDSD	Model Driven Software Development
MDA	Model Driven Architecture
OMG	Object Management Group
CIM	Computation Independent Model
BPMN	Business Process Model and Notation
TFM	Topological Functioning Model [of a system]
UML	Unified Modelling Language
NLP	Natural Language Processing
IDM	Integrated Domain Modelling
NER	Name Entity Recognition
DFD	Data Flow Diagram
OCL	Object Constraint Language
SBVR	Semantic of Business Rules and Vocabulary

References

Asghar, N. (2016) Automatic Extraction of Causal Relations from Natural Language Texts : A Comprehensive Survey, *CoRR*, abs/1605.0. Available at: http://arxiv.org/abs/1605.07895.

Asnina, E. (2006) The Computation Independent Viewpoint: a Formal Method of Topological Functioning Model Constructing, *Applied computer systems*, 26, 21–32.

Asnina, E., Osis, J. (2010) Computation Independent Models: Bridging Problem and Solution Domains, In: *Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development*. Lisbon, SciTePress - Science and and Technology Publications, 23–32. doi: 10.5220/0003043200230032.

Asnina, E., Ovchinnikova, V. (2015) 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*. Lisbon, SciTePress - Science and and Technology Publications, 364–373

Bies, A., Ferguson, M., Katz, K., MacIntyre, R. (1995) *Bracketing Guidelines for Treebank II Style*. Available at: http://languagelog.ldc.upenn.edu/myl/PennTreebank1995.pdf.

Bousetta, B., Beggar el, O., Gadi, T. (2013) A methodology for CIM modelling and its transformation to PIM, *Journal of Information Engineering and Applications*, 3(2), 1–21. Available at: www.iiste.org.

Dannélls, D., Gruzitis, N. (2014) Controlled Natural Language Generation from a Multilingual FrameNet-Based Grammar, In: Davis, B., Kaljurand, K., Kuhn, T. (eds) *Controlled Natural Language*. Cham, Springer International Publishing, 155–166.

Donins, U., Osis, J., Slihte, A., Asnina, E., Gulbis, B. (2011) Towards the refinement of topological class diagram as a platform independent model, In: Čaplinskas, A. et al. (eds) *Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development, MDA and MDSD 2011, In: Conjunction with ENASE 2011*. Vilnius, Žara, 79–88.

Essebaa, I., Chantit, S. (2016) Toward an automatic approach to get PIM level from CIM level using QVT rules, In: *2016 11th International Conference on Intelligent Systems: Theories and Applications (SITA)*. Mohammedia, IEEE, 1–6. doi: 10.1109/SITA.2016.7772271.

Germann, U., Miranda, S., Nogueira, D., Liepins, R., Gosko, D., Barzdins, G. (2018) The SUMMA Platform: A Scalable Infrastructure for Multi-lingual Multi-media Monitoring, In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics-System Demonstrations*. Association for Computational Linguistics, 99–104.

Kalnins, A., Smialek, M., Kalnina, E., Celms, E., Nowakowski, W., Straszak, T. (2011) Domain-Driven Reuse of Software Design Models, In: Osis, J., Asnina, E. (eds) *Model-Driven Domain Analysis and Software Development*. Hershey, PA, IGI Global, 177–200. doi: 10.4018/978-1-61692-874-2.ch009.

Kang, D., Gangal, V., Lu, A., Chen, Z., Hovy, E. (2017) Detecting and Explaining Causes from Text For a Time Series Event, In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, The Association for Computational Linguistics, 2758–2768.

Kardoš, M., Drozdová, M. (2010) Analytical method of CIM to PIM transformation in model driven architecture (MDA), *Journal of Information and Organizational Sciences*, 34(1), 89–99.

Kherraf, S., Lefebvre, É., Suryn, W. (2008) Transformation from CIM to PIM Using Patterns and Archetypes, In: *19th Australian Conference on Software Engineering (aswec 2008)*, IEEE, 338–346. doi: 10.1109/ASWEC.2008.4483222.

Khoo, C., Chan, S., Niu, Y. (2002) The Many Facets of the Cause-Effect Relation, In: Green, R., Bean, C. A., Myaeng, S. H. (eds) *The Semantics of Relationships: An Interdisciplinary Perspective*. Dordrecht, Springer Netherlands, 51–70. doi: 10.1007/978-94-017-0073-3_4.

Kriouile, A., Addamssiri, N., Gadi, T., Balouki, Y. (2014) Getting the static model of PIM from the CIM, In: *2014 Third IEEE International Colloquium in Information Science and Technology (CIST)*. Tetouan, IEEE, 168–173. doi: 10.1109/CIST.2014.7016613.

Kriouile, A., Addamssiri, N., Gadi, T. (2015) An MDA Method for Automatic Transformation of Models from CIM to PIM, *American Journal of Software Engineering and Applications*. Science Publishing Group, 4(1), 1–14. doi: 10.11648/j.ajsea.20150401.11.

Kriouile, A., Gadi, T., Balouki, Y. (2013) CIM to PIM Transformation: A Criteria Based Evaluation, In: *J.Computer Technology & Applications*, 4(4), 616–625.

Leffingwell, D., Widrig, D. (2003) *Managing Softqare Requirements: a Use Case Approach*. 2nd edn. Addison-Wesley.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., McClosky, D. (2014) The Stanford CoreNLP Natural Language Processing Toolkit, In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 55–60. Available at: https://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf.

Miller, J., Mukerji, J. (2001) *Model Driven Architecture (MDA), Architecture Board ORMSC*. Available at: http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01.

Mirza, P. (2014) Extracting Temporal and Causal Relations between Events, In: *Proceedings of the ACL 2014 Student Research Workshop*. Baltimore, Maryland, USA: Association for Computational Linguistics, 10–17. doi: 10.3115/v1/P14-3002.

Mokrys, M. (2012) Possible transformation from Process Model to IS Design Model, In: *ICTIC - PROCEEDINGS IN CONFERENCE OF INFORMATICS AND MANAGEMENT SCIENCES*. EDIS - Publishing Institution of the University of Zilina, 71–74.

Mostafazadeh, N., Grealish, A., Chambers, N., Allen, J., Vanderwende, L. (2016) CaTeRS : Causal and Temporal Relation Scheme for Semantic Annotation of Event Structures, In: *Proceedings of the Fourth Workshop on Events*. San Diego, California: Association for Computational Linguistics, 51–61. doi: 10.18653/v1/W16-1007.

Nazaruka, E. (2019) Identification of Causal Dependencies by using Natural Language Processing: A Survey, In: Damian, E., Spanoudakis, G., and Maciaszek, L. (eds) *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: MDI4SE*. SciTePress, 603–613. doi: 10.5220/0007842706030613.

Nazaruka, E., Osis, J. (2018) Determination of Natural Language Processing Tasks and Tools for Topological Functioning Modelling, In: *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering*. Funchal, Madeira, Portugal, SCITEPRESS – Science and Technology Publications, Lda., 501–512.

Nazaruka, E., Osis, J. (2019) The Formal Reference Model for Software Requirements, In: Damiani, E., Spanoudakis, G., and Maciaszek, L. (eds) *Evaluation of Novel Approaches to Software Engineering. ENASE 2018. Communications in Computer and Information Science, vol 1023*. Springer, Cham, 352–372. doi: 10.1007/978-3-030-22559-9_16.

Nazaruka, E., Osis, J., Griberman, V. (2019) Extracting Core Elements of TFM Functional Characteristics from Stanford CoreNLP Application Outcomes, In: Damian, E., Spanoudakis, G., and Maciaszek, L. (eds) *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: MDI4SE*. SciTePress, 591–602. doi: 10.5220/0007831605910602.

Ning, Q., Feng, Z., Wu, H., Roth, D. (2018) Joint Reasoning for Temporal and Causal Relations, In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Long Papers)*. Melbourne, Australia, Association for Computational Linguistics, 2278–2288.

Osis, J., Asnina, E. (2008) Enterprise Modeling for Information System Development within MDA, In: *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. Waikoloa, USA, IEEE, 490–490. doi: 10.1109/HICSS.2008.150.

Osis, J., Asnina, E. (2011a) Derivation of Use Cases from the Topological Computation Independent Business Model, In: *Model-Driven Domain Analysis and Software Development*. Hershey, PA, IGI Global, 65–89. doi: 10.4018/978-1-61692-874-2.ch004.

Osis, J., Asnina, E. (2011b) 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, 15–39. doi: 10.4018/978-1-61692-874-2.ch002.

Osis, J., Asnina, E., Grave, A. (2007) MDA oriented computation independent modeling of the problem domain, In: *Proceedings of the 2nd International Conference on Evaluation of Novel Approaches to Software Engineering - ENASE 2007*. Barcelona, INSTICC Press, 66–71.

Osis, J., Asnina, E, Grave, A. (2008a) Computation Independent Representation of the Problem Domain in MDA, *e-Informatica Software Engineering Journal*, 2(1), 29–46. Available at: http://www.e-informatyka.pl/index.php/einformatica/volumes/volume-2008/issue-1/article-2/.

Osis, J., Asnina, E., Grave, A. (2008b) Formal Problem Domain Modeling within MDA, In: Filipe, J. et al. (eds) *Software and Data Technologies: Second International Conference, ICSOFT/ENASE 2007, Barcelona, Spain, July 22-25, 2007, Revised Selected Papers*. Berlin, Heidelberg, Springer Berlin Heidelberg, 387–398. doi: 10.1007/978-3-540-88655-6_29.

Osis, J., Donins, U. (2010) Formalization of the UML Class Diagrams, In: *Evaluation of Novel Approaches to Software Engineering*. New York, Springer, Berlin, Heidelberg, 180–192. doi: 10.1007/978-3-642-14819-4_13.

Osis, J., Donins, U. (2017) *Topological UML modeling : an improved approach for domain modeling and software development*. Elsevier.

Osis, J., Slihte, A. (2010) Transforming Textual Use Cases to a Computation Independent Model, In: Osis, J,Nikiforova, O. (eds) *Model-Driven Architecture and Modeling-Driven Software Development: ENASE 2010, 2ndMDA&MTDD Whs.* SciTePress, 33–42.

Pearl, J. (2019) The Seven Tools of Causal Inference, with Reflections on Machine Learning, *Communications of Association for Computing Machinery*, 62(3), 54–60. doi: 10.1145/3241036.

ReDSeeDS (2020) *Model-Driven Requirements Engineering in action, ReDSeeDS | Requirements-Driven Software Development System*. Available at: http://smog.iem.pw.edu.pl/redseeds/

Rhazali, Y., Hadi, Y., Mouloudi, A. (2015) Disciplined approach for transformation CIM to PIM in MDA, In: *Model-Driven Engineering and Software Development (MODELSWARD), 2015 3rd International Conference on*. IEEE, 312–320.

Rhazali, Y., Hadi, Y., Mouloudi, A. (2016) CIM to PIM Transformation in MDA: from Service-Oriented Business Models to Web-Based Design Models, In: *International Journal of Software Engineering and Its Applications*, 10(4), 125–142. doi: 10.14257/ijseia.2016.10.4.13.

Schneider, G., Winters, J. P. (2001) *Applying Use Cases: A practical Guide*. 2nd edn. Pearson Education, Inc.

Šlihte, A., Osis, J. (2014) The Integrated Domain Modeling: A Case Study, In: *Databases and Information Systems: Proceedings of the 11th International Baltic Conference (DB&IS 2014)*. Tallinn, Tallinn University of Technology Press, 465–470.

Slihte, A., Osis, J., Donins, U. (2011) Knowledge Integration for Domain Modeling, In: Osis, J., Nikiforova, O. (eds) *Model-Driven Architecture and Modeling-Driven Software Development: ENASE 2011, 3rd Whs. MDA&MDSD*. SciTePress, 46–56.

Solstad, T., Bott, O. (2017) Causality and causal reasoning in natural language, In: Waldmann, M. R. (ed.) *The Oxford Handbook of Causal Reasoning*. Oxford University Press. Available at: http://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199399550.001.0001/oxfordhb-9780199399550-e-32.

Waldmann, M. R, Hagmayer, Y. (2013) Causal reasoning, In: Reisberg, D. (ed.) *Oxford Handbook of Cognitive Psychology*. New York: Oxford University Press. doi: 10.1093/oxfordhb/9780195376746.013.0046.

Received August 22, 2019, revised February 3, 2020, accepted February 11, 2020