The detection of conflicts in the requirements specification based on an ontological model and a production rule system

M Sh Murtazina[1] and T V Avdeenko[1]

[1]Novosibirsk State Technical University, Karla Marksa ave., 20, Novosibirsk, Russia, 630073

e-mail: murtazina@corp.nstu.ru; tavdeenko@mail.ru

Abstract. The paper presents an approach to organizing the detection of conflicts between requirements based on an ontological model and a system of production rules. Requirements in the form of plain text are converted to instances of OWL ontologies for analysis. There are three basic elements “subject”, “action” and “object” in the requirements. These elements become classes of the ontology. The object properties between instances of the same classes are defined in the ontology. In the system of rules it is determined that one of four types of relations can be established between a pair of the requirements: isConflict, isDuplicate, isInterconnect, isNotInterconnect. We develop the software product in the Python language for building and applying production rules system for classes and property objects of the ontology. Protégé 5.2 is used to work with the ontology. Also Python library PySwip and development environment SWI-Prolog 7.6.4 are used in the work. The paper also considers the issues of extracting requirements ontology instances from the automated processing results of textual requirements. The UDPipe with Russian language model is used for text processing

CEUR-WS.org/Vol-2416/paper9.pdf

1. Introduction

Essential precondition for successful implementation of the software products is the preliminary development of high-quality software requirements specification (SRS). The latter task is solved during requirements engineering (RE) process, in which inaccurate and incomplete ideas about what services should the software product provide in order to solve the potential users’ problems are converted into a formal SRS. Moreover, it is necessary to take into account the limitations under which the software product should be implemented and work. According to SEBoK (version1.9.1) a set of requirements must satisfy the following characteristics: complete, consistent, feasible, comprehensible and able to be validated.The completeness means that the set of requirements need not be further refined. The consistency criterion assumes that the requirement set does not contain conflicting requirements, the requirements are not duplicated, and the same term is used for the same element in the whole set of requirements. The feasibility criterion means that a solution can be developed for a set of requirements that meets the requirements taking into account the limitations of the project (for example, cost) with an acceptable level of risk. The comprehensibility criterion implies that a set of requirements has to be formulated so that to give a clear idea of what is expected of the software product, and in what way it relates to the whole system. The validability criterion implies that

it should be possible to prove that the set of requirements will lead to the user needs satisfaction under existing constraints of the project [1].

For the agile software development approach, the requirements specification is usually not a single document, but logical structure filled with requirements to the software product. The creation of such specifications is usually preceded by the development of a document called «Vision and scope», which gives an overview of the developed software product in terms of the key needs and constraints under which the project will be implemented. Then, the work begins on the creation of the product backlog containing prioritized list of requirements. The major feature of an agile approach is permanent readiness to change the requirements, therefore only those elements of the product backlog that have the highest priorities are worked out in detail at the current time. The requirements with low priorities remain in the product backlog, where they can be added, removed, or changed at any time. Under conditions of variability of the requirements their coordination becomes one of the most difficult problems in the requirements engineering.

Requirements negotiation involves detection and resolution of conflicts. Therefore, it seems that at the stage of collecting the product backlog elements, the most important of the above quality characteristics is to ensure the consistency of the set of product backlog elements. In this paper we will consider the main component of this quality characteristic, namely, conflict-free set of requirements. The proposed approach can be used as part of the intellectual support of the engineering process based on ontological models. The ontology data of requirements can be processed together with the data of other ontologies in the field of requirements engineering described by us in works [2-4].

The paper is organized as follows. Section 2 provides an overview of publications on the research theme. In section 3 we propose an approach to converting text requirements into the ontology. In section 4, we describe the developed software based on production model to search for conflicts between the requirements represented by instances of the ontology. Section 5 concludes on the prospects of the proposed approach.

2. Theoretical background

The paper [5] describes a prototype of the «Oz» design system, which provides automated methods for conflict detection, generation of conflict resolution variants, and explanation of the decision choice. So the experimental tool «Oz» is designed to develop harmonized requirements. Requirements from different stakeholders are presented in the form of scenarios. The «Oz» prototype includes requirements language, specification language, specification planner, and requirements negotiation tools. The work [6] of the same authors presents an approach to the identification and resolution of conflicts between the requirements within the framework of the paradigm of Conflict-Oriented Requirements Analysis (CORA). Requirements ontology is used to describe the structure of the requirements, which providing formulation of the requirements and their interaction. Conflicts between the requirements are presented in the requirements ontology as types of interaction. The interactions between the requirements are determined directly from the requirements concept description. Conflict resolution is based on the generation of alternative requirements' variants resolving the conflict.

The authors of [7] mentioned that manually produced conflict-analysis of the requirements needs considerable effort and is prone to the errors. They propose to use semantic technologies as a basis for automating the analysis of conflicts between the requirements implementing ontological approach OntRep. In [7] three types of conflicts are distinguished: conflicts between the requirements and the constraints, conflicts between the requirements and the guidelines (for example, on formalization of requirements), conflicts between the requirements. Conflicts between the requirements are divided into two groups: simple (between two requirements) and complex (between several requirements). To implement the conflict analysis approach, all terms used in the requirements must be formulated as Glossary terms in the ontology OntRep, and the requirements must be formulated using a specific structure. The OntRep tool consists of two basic components. The first component is an instance collector that takes input data. For example, these might be requirement tickets from the project management tool and bug tracking in Trac software. A ticket in this system means a virtual card with a description of the requirement (task) or the error that needs to be corrected. The OntRep tool instance

collector analyzes the contents of the ticket and assigns categories of the requirements (classes) defined in the ontology (for example, security requirement). The second component of OntRep tool is intended to generate a report on conflicts between the requirements [8].

The paper [9] describes an automated tool EA-Analyzer for revealing conflicts. This tool is designed to identify conflicts in aspect-oriented requirements specified in natural language text. Requirements are written using RDL (Requirements Description Language), which uses XML tags to annotate the natural language specification. The main elements of the language are <subject>, <object> and <relationship> tags. The <subject> tag grammatically corresponds to the subject of the sentence, the <object> tag corresponds to the object to which the action applies, and the <relationship> tag defines the action performed by the subject over the object. There can be several objects in one sentence. Classification of verbs in RDL is based on semantic categories proposed in the works of R. Dixon [10]. Also RDL uses categories of degrees (for example, showing the differences between the modal verbs), reflecting the importance degree of the requirement. Contamination (merging heterogeneous factors) consists of three sub-elements: constraint, basic requirement and result. In EA-Analyzer, the conflict detection problem is formulated as a classification problem that is a well-studied machine learning problem. The EA-Analyzer tool uses annotated requirements in RDL to decide if there are inconsistencies in the requirements [9].

The paper [11] proposes a set of rules for conflict analysis of class diagrams. The conflict-analysis process is a four-step cycle: modeling a priori knowledge, modeling new requirements, identifying conflicts, and resolving conflicts. The rules process four possible types of conflicts: inconsistency, redundancy, overriding, and missing parts.

The paper [12] deals with the definition of cooperation and conflicts between the requirements. The requirements conflict if they contain conflicting statements about common software attributes, and they are in a cooperative state if they complement each other's statements about attributes. The approach proposed in [12] includes manual transformation of the requirements into the software attributes, automatic detection of conflicts and cooperation between the requirements based on their attributes, automatic tracing of dependencies between the requirements.

In [13] an approach is proposed that allows comparing the stakeholder’s points of view on the subsystems of the information system. Requirements of stakeholders, written in natural English language, are processed with the help of the syntactic StanfordParser and the tool TreeTagger for morphological tagging of texts. On the basis of the analysis of the received semantic-syntactic relations on a set of rules the ontology of functional requirements is created. The concepts of the resulting ontologies are collated using external resources such as WordNet and string matching methods such as the N-gram method. As a result of comparison between the concepts of two textual descriptions of the requirements the following semantic relations are determined: the concepts are equivalent, one concept is more or less general against another concept, two or more concepts belong to the same subset or not.

The paper [14] proposes an approach to the automatic building the ontology out of a set of user stories. SpaCy library is used for processing texts in natural English language that allows to make parsing sentences based on the dependency tree, searching named groups, extracting registered entities, as well as permitting coreference. The problem of extracting knowledge in a natural language for further reflection in the ontology of a subject area is reflected in [15].

Thus, the conducted analysis of publications on the topic of the study allows us to conclude that one of the promising areas in search for conflicts between the requirements are approaches involving presentation of text requirements in the form of ontology instances. This can be done manually and semi-automatically. Automatic construction of ontologies from the texts of requirements is currently extremely relevant and still not well covered topic. The considered papers on the construction of ontologies from the texts of requirements does not directly affect the problem of conflict determination, but the ideas used in them can be used as a basis for extracting from the results of automatic text processing of requirements of instances of ontology, structured for the task of finding conflicts between requirements.

3. The conversion of textual requirements into ontology instances

In a sentence that expresses a requirement, regardless of the requirements recording technique used (for example, requirements can be written in the IEEE 830 or user stories style), three main parts can be distinguished: the subject, the action, and the object on which the action is directed. Each requirement must be represented in the ontology as a set of instances of these three classes. The object properties between pairs of instances for each class must be determined. For each requirement it is necessary to create an instance of the class "Requirement", which will be associated with corresponding instances of the classes "subject" ("actor"), "action" and "object". It is also necessary to determine the possible relations between instances of classes. It should be noted that a class, the elements of which grammatically correspond to the subjects of sentences with requirements, in an ontology may be called differently, for example, "actor" or "functional user role". In the example below which illustrates the proposed approach the term "actor" will be used to designate the subject.

First, in the Protégé 5.2 environment, the structure of classes was implemented and the object properties between them were set. Figure 1 shows the object properties of ontology, their domains and ranges.

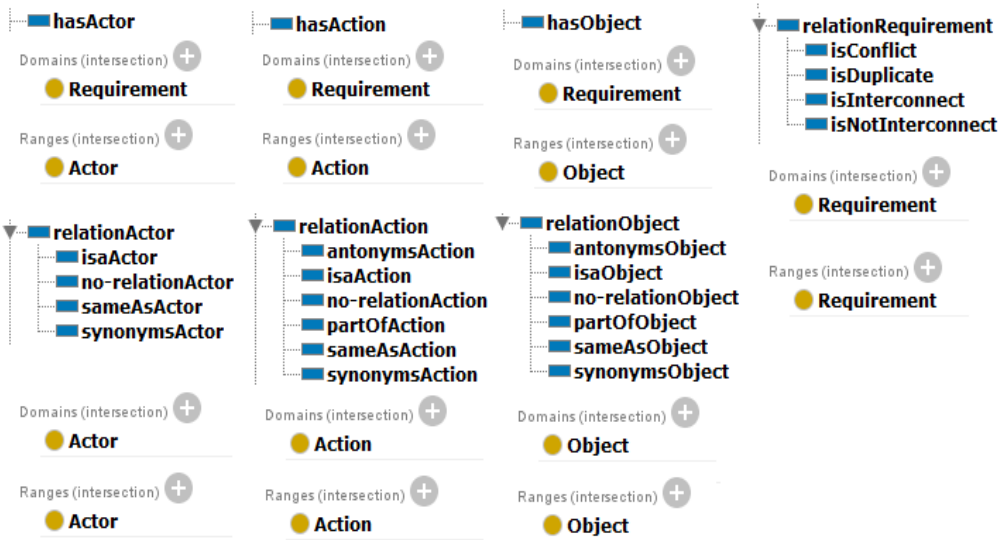

Figure 1. Object properties of ontology, their domains and ranges.

The object properties between instances of the same class "Actor" are grouped into the object properties class "relationActor". Also, the relations between the instances of the classes "Action" and "Object" are grouped into object properties classes "relationAction" and "relationObject".

The object properties of the "sameAsActor", "sameAsAction", "sameAsObject" are set up between instances of the corresponding classes if one name is used for the elements of two requirements or the full name is in one and an abbreviation in the second. These properties are symmetric. For example, instances of the class "Actor" with the values "senior manager" ("sr. manager") and "senior manager" will be connected by the object property "sameAsActor". Also, each of them will be connected with itself by this property. It is necessary for comparing two requirements without including duplicates of the same instances of the class "Actor" in the ontology (see Figure 2).

The object properties "isaActor", "isaAction" and "isaObject" are used to setting hierarchical relations. For example, the "senior manager" is a "manager".

The object properties "synonymsActor", "synonymsAction" and "synonymsObject" are set if the values of the corresponding requirements elements have a synonymous value. For example, the action "to view" records is synonymous with the action "to list" records.

The object properties "antonymsAction" and "antonymsObject" are set between instances of the corresponding classes if they have the opposite meaning. For example, the object "my comment" and the object "someone else's comment".

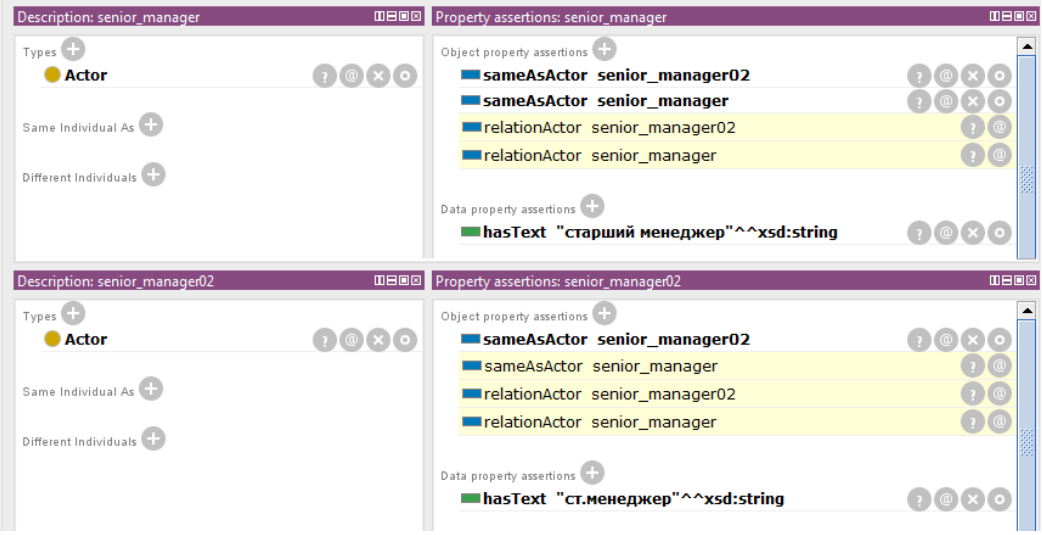

Figure 2. Object property for individuals «senior manager».

The object properties “partOfAction” and “partOfObject” are set between instances of the respective classes if one is part of the other. For example, the “product” object is part of the object “shopping basket”. Or, for example, the “delete” action is part of the “edit” action.

In all other cases, it is considered that the corresponding elements of the requirements are bound by the object properties “no-relationActor”, “no-relationAction” and “no-relationObject”. In other words, the relation between them is not revealed.

Figure 3 shows an example of requirement elements transformed into ontology instances.

Requirements:

User story 01: Как *менеджер*, *Я* могу удалить только *добавленные мною описания товаров*…
As a *manager*, I can only delete the *product descriptions I added* ...

User story 02: Как *менеджер*, *Я* могу редактировать *описания всех товаров из каталога*….
As a *manager*, I can edit *descriptions for all products from the catalog* ....

It should be noted that the conversion of textual requirements into ontology instances can be carried out in semi-automatic mode using automatic text processing tools. The UDPipe with the Russian language model can be used for processing of textual requirements in Russian. The UDPipe parses the proposal and provides the results in the format CoNLL-U.

The following fields are used to describe the annotation of text token (word, punctuation mark, or special character) of a sentence in the CoNLL-U format:

1) ID: token index
2) FORM: word form or punctuation symbol,
3) LEMMA: lemma of word form,
4) UPOSTAG: universal part of speech tag,
5) XPOSTAG: language tag of a part of speech (if not, a dash is indicated),
6) FEATS: list of morphological features from the universal feature inventory or from a specific language extension,
7) HEAD: number of the parent token or zero for the root token,
8) DEPREL: universal dependency relation to parent token (set to “root” if HEAD = 0),
9) DEPS: list of secondary dependencies
10) MISC: additional information.

Figure 4 illustrates the presentation in the CoNLL-U format of the sentence “Как менеджер, Я хочу удалять товар из каталога” (“As a manager, I want to remove product from the catalog”).

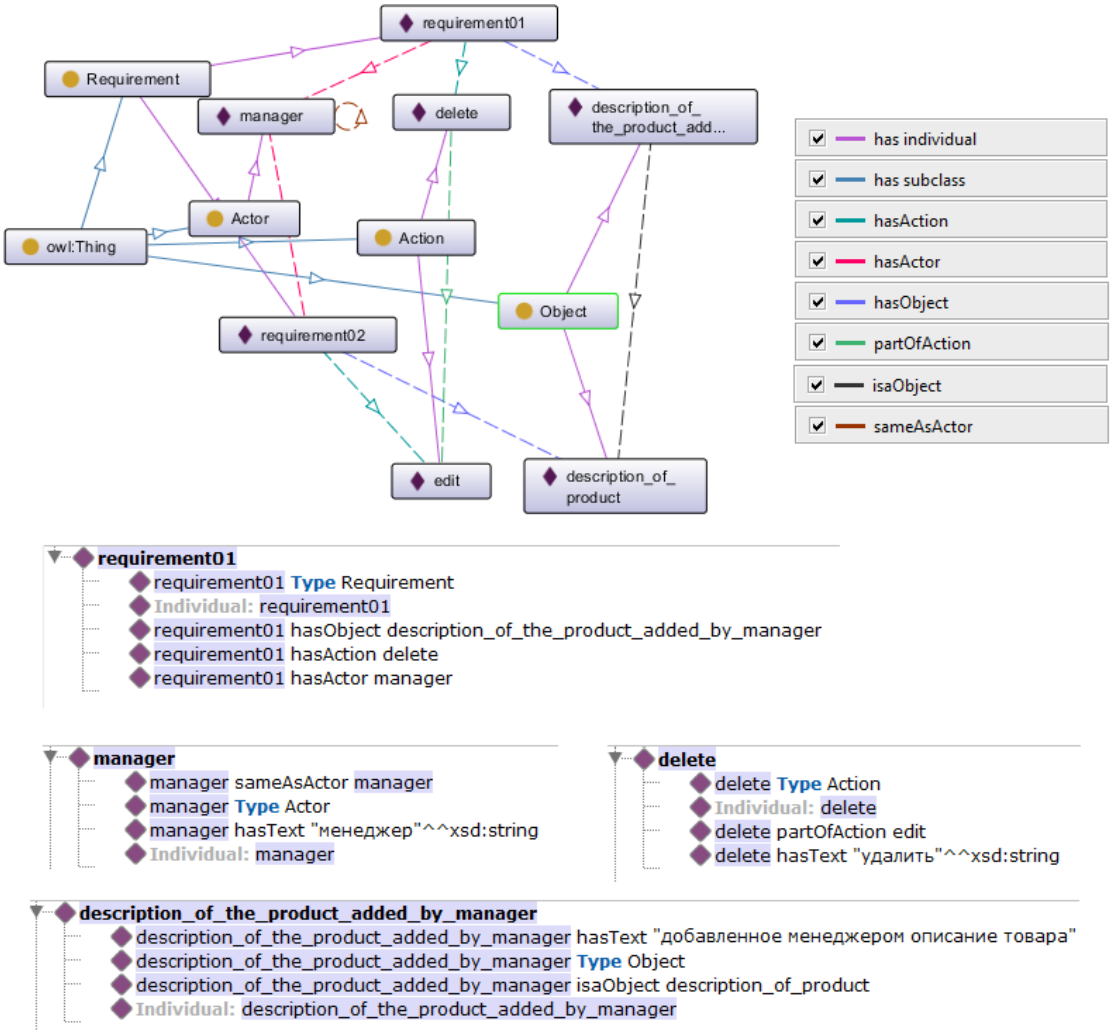

Figure 3. Fragment of ontology in Protégé.

Id	Form	Lemma	UPosTag	XPosTag	Feats	Head	DepRel	Deps	Misc
# newdoc									
# newpar									
# sent_id = 1									
# text = Как менеджер, Я хочу удалять товар из каталога.									
1	Как	как	SCONJ	_	_	2	mark	_	_
2	менеджер	менеджер	NOUN	_	Animacy=Anim|Case=Nom|Gender=Masc|Number=Sing	5	parataxis	_	SpaceAfter=No
3	,	,	PUNCT	_	_	2	punct	_	_
4	Я	я	PRON	_	Case=Nom|Number=Sing|Person=1	5	nsubj	_	_
5	хочу	хотеть	VERB	_	Aspect=Imp|Mood=Ind|Number=Sing|Person=1|Tense=Pres|VerbForm=Fin|Voice=Act	0	root	_	_
6	удалять	удалять	VERB	_	Aspect=Imp|VerbForm=Inf|Voice=Act	5	xcomp	_	_
7	товар	товар	NOUN	_	Animacy=Inan|Case=Acc|Gender=Masc|Number=Sing	6	obj	_	_
8	из	из	ADP	_	_	9	case	_	_
9	каталога	каталог	NOUN	_	Animacy=Inan|Case=Gen|Gender=Masc|Number=Sing	6	obl	_	SpaceAfter=No
10	.	.	PUNCT	_	_	5	punct	_	SpaceAfter=No

Figure 4. Phrase parsing done by the UDPipe tool.

A subject (actor), an action and an object can be automatically extracted from the results of a sentence analysis, provided that the sentence with the requirement is formulated simply enough, that is does not contain a lot of speech turns (see Figure 5).

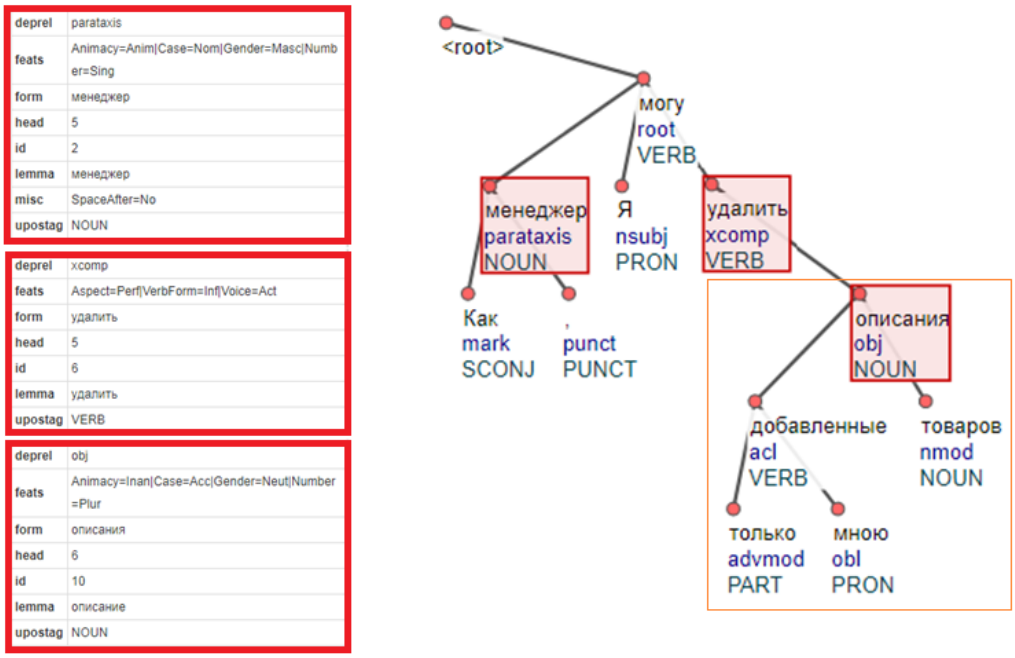

Figure 5. Dependency tree obtained by UDPipe.

The setting of object properties between instances of requirements elements can also be performed in semi-automatic mode. For example, using regular expressions, modal verbs and negatives can be searched. The linguistic ontologies like WordNet can be used to search for synonyms. It should be said that the search for antonyms with the help of linguistic ontologies for verbs can lead to undesirable results, since, for example, the actions “add” and “delete” can be viewed as antonyms, however, the verb phrases “can remove” and “cannot remove” will have an antonymous meaning .

Production rules can be applied for processing the parser results of sentences. For example: If X is an animated noun (UPOSTAG = NOUN and FEATS: Animacy = Anim), then this is a subject (actor). Next, it is necessary to check whether the noun has a definitive in order to extract the whole name of the subject. For example, if the actor “senior manager” is used instead of the actor “manager” then the actor is described as an animated noun (ID = 3, UPOSTAG = NOUN, FEATS: Animacy = Anim) and definitive “senior” (ID = 2, UPOSTAG = ADJ, DEPREL = amod, HEAD = 3). The universal dependence relation “amod” indicates that the adjective “senior” serves as a definitive for the noun “manager”. In the case of user story, the case of actor search is simplified due to a certain position of the actor in the sentence, so the actor can be extracted even without the use of special tools.

To extract the «action», it is necessary to analyze the tokens that are defined as verbs (UPOSTAG = VERB). At the same time, attention should be paid to the morphological features of the token in order to separate the verbs from the participles (FEATS: VerbForm = Part) and the adverbial participles (FEATS: VerbForm = Conv). In the example in Figure 5, the words «“want” (UPOSTAG = VERB, FEATS: VerbForm = Fin) and “delete” (UPOSTAG = VERB, FEATS: VerbForm = Inf) are the verbs. The phrase “I want” in the user Story is predefined, therefore, we extract the word “delete” as an action.

The object will be a noun associated with the word-action. In this example, this is the word “description” (ID = 11, UPOSTAG = NOUN, DEPREL = obj, HEAD = 6). The universal dependence relation “obj” indicates that the noun “description” plays the role of a direct object for the verb “delete”. Next, it is need to check whether the word “description” has attribute relations with other words of a sentence in order to extract the whole object.

Determining object properties between the instances of the requirements elements can also be performed in semi-automatic mode. For example, modal verbs and negatives can be searched using regular expressions. Besides, the linguistic ontologies like WordNet can be used to search for synonyms. It should be noted that the search for antonyms of verbs with the help of linguistic

ontologies can lead to undesirable results. For example, the actions “add” and “delete” can be viewed as antonyms, however, the verb phrases “can delete” and “cannot delete” will have an antonymous meaning in terms of the problem to be solved.

4. An approach to detection of conflicts between the requirements

To determine the type of relationship between the requirements, a system of production rules of the following form can be generated:

IF ObjectProperty_between_class_instances (Actor1, Actor2)
AND ObjectProperty_between_class_instances (Action1, Action2)
AND ObjectProperty_between_class_instances (Object1, Object2)
THEN Objectproperties_between_class_instances (Requirement1, Requirement2)

For the ontology with the properties of objects presented in figure 1, 144 rules were formed with the proposed approach. Below we give some examples of conflict detection rules:

1. **IF** isaActor (Actor1, Actor2) **AND** antonymsAction (Action1, Action2)
AND isaObject (Object1, Object2) **THEN** isConflict (Requirement1, Requirement2)

2. **IF** isaActor (Actor1, Actor2) **AND** antonymsAction (Action1, Action2)
AND partOfObject (Object1, Object2) **THEN** isConflict (Requirement1, Requirement2)

3. **IF** isaActor (Actor1, Actor2) **AND** partOfAction (Action1, Action2)
AND antonymsObject (Object1, Object2) **THEN** isConflict (Requirement1, Requirement2)

Depending on the used rule, one of the following four relations between the requirements are detected: “conflicting requirements”, “requirements are duplicates”, “requirements are related” or “connection is not detected”.

To implement the proposed approach, an application with a graphical interface in Python was developed. The application provides the following functions:

- building a system of production rules to process relations between instances of ontology classes and saving them as a model into XML file;
- editing the system of production rules previously saved as a model;
- application of the model rules to the requirements, presented in the form of class instances of OWL ontology.

To implement the first function, a constructor was created that helps the user to build a system of rules for analyzing the relations between the requirements elements in a few «clicks». The constructor allows you to select a file with OWL ontology for which the model was built. In the next steps of working with the designer, you can define which ontology classes correspond to the elements «actor», «action» and «object», as well as select the properties of objects that will be used for analysis. This allows the end user to create their own rules for processing the data they have. At the last step of working with the application, the generated combinations of antecedents for the production rules are issued. The user needs to define rule consequents by selecting the type of relation between the requirements for each rule. Semi-automatic rule generation does not allow the user to skip any rule because it covers all possible combinations for rule antecedents. Figure 6 shows the last step in building the model.

Further we consider two examples for rules obtained from the ontology described above. The first example analyzes the requirements in the form of user stories; the second example analyzes the requirements in the form of simple statements.

Example 1:
IF sameAsActor(Actor1, Actor2)
AND partOfObject(Action1, Action2)

AND isaObject(Object1, Object2)
THEN isConflict(Requirement1, Requirement2)
Substitution of data into the rule:
IF sameAsActor(«менеджер», «менеджер»)
AND partOfAction(«удалить», «редактировать»)
AND isaObject(«добавленное менеджером описание товара», «описание товара в каталоге»)
THEN isConflict(Requirement1, Requirement2)

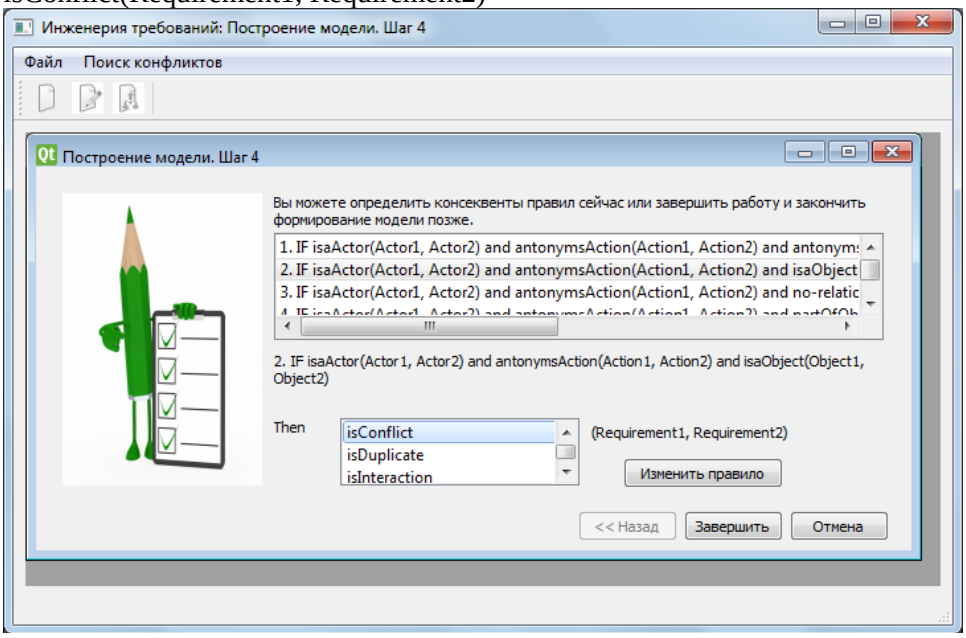

Figure 6. Constructor form screen.

Example 2:
IF sameAsActor(Actor1, Actor2)
AND antonymsAction(Action1, Action2)
AND sameAsObject(Object1, Object2)
THEN isConflict(Requirement1, Requirement2)

Requirements*:*
R1: Пользователь может удалять комментарии.
The user can delete comments
R2: Пользователь может только просматривать комментарии.
The user can only view comments.

Substitution of data into the rule:
IF sameAsActor(«пользователь», «пользователь»)
AND antonymsAction(«может удалять», «может только просматривать»)
AND sameAsObject(«комментарии», «комментарии»)
THEN isConflict (Requirement1, Requirement2)

In the first example, the Manager on the one hand can edit the descriptions of any goods (regardless of whether he added them or not), and on the other hand he has a ban on editing the descriptions of goods added not by him. In the second example, we see a contradiction in the actions of the user on one object “comments”, which also requires clarification and correction, because it is not clear what kind of comments we mean: only the comments created by the user himself, or all the comments to the objects created by the user in the system. And at first glance, “understandable flaws” can lead to the implementation of functionality that does not correspond to the original idea.

In the course of requirements analysis, data in the form of classes' instances and relationships between them are imported into the database of facts in Prolog language, the generated production rules are imported into Prolog Rules. Then the Prolog Inference Engine is used to detect conflicts between the requirements. The results of the requirements analysis are saved into a text file. Python library PySwip and SWI-Prolog 7.6.4 development environment were used to implement this functionality.

5. Conclusion

The paper proposed an approach to detection of conflicts in the set of requirements based on an ontological model and a production rules system. To organize the processing of textual requirements, it was proposed to extract from the textual requirements the subject (actor), the action and the object on which the action is directed and save them in the form of ontology instances. It was also proposed to establish relations between instances of classes belonging to a pair of requirements that are defined in the ontology as object properties. The production rules system for determining the type of relation between a pair of requirements is based on these object properties. The developed toolkit to be able to facilitate the process of detection conflicting requirements since it performs the matching of requirements in a semi-automatic mode, which contributes to the detection of seemingly minor inaccuracies in the requirements formulation. It should be noted that the application of this approach requires the recording of textual requirements in the form of simple sentences that can be partially processed automatically, since it is quite labor-intensive to completely manually convert textual requirements into ontology instances.

6. References

[1] SEBoK *Guide tothe Systems Engineering Body of Knowledge* (Version 1.9.1)
[2] Murtazina M S and Avdeenko T V 2018 Ontology-Based Approach to the Requirements Engineering in Agile Environment *Proc. of 14th International Scientific-Technical Conference on Actual Problems of Electronic Instrument Engineering* **8546144** 496-501
[3] Avdeenko T and Murtazina M 2018 Intelligent Support of Requirements Management in Agile Environment *Studies in Computational Intelligence* **803** 97-108
[4] Murtazina M Sh and Avdeenko T V 2019 An ontology-based approach to support for requirements traceability in agile development *Procedia Computer Science* **150** 628-635
[5] Robinson W N and Volkov S 1998 Supporting the Negotiation Life Cycle *Communications of the ACM* **41(5)** 95-102
[6] Robinson W N and Volkov S 1999 Requirement Conflict Restructuring *GSU CIS Working Paper* **99-5** 1-47
[7] Heindl M, Moser T, Winkler D and Biffl S 2011 Automating the Detection of Complex Semantic Conflicts between Software Requirements: An empirical study on requirements conflict analysis with semantic technology *Proc. Int. Conf. on Software Engineering and Knowledge Engineering* (Miami Beach: USA Knowledge Systems Institute Graduate School) 729-735
[8] Moser T,Winkler D, HeindlM and Biffl S 2011 Requirements Management with Semantic Technology: An Empirical Study on Automated Requirements Categorization and Conflict Analysis *Proc. Int. Conf. on Advanced Information Systems Engineering* 3-17
[9] Sardinha A, Chitchyan R, Araújo J, Moreira A and Rashid A 2013 Conflict identification with EA-Analyzer *Aspect-oriented requirements engineering* 209-224
[10] Chitchyan R, Rashid A, Rayson P and Waters R 2007 Semantics-Based Composition for Aspect-Oriented Requirements Engineering *Proc. Int. Conf. on Aspect-oriented software development* 36-48
[11] Liu C-L and Huang H-H 2015 Ontology-Based Requirement Conflicts Analysis in Class Diagrams *Proc.of the World Congress on Engineering* **1** 471-476
[12] Egyed A and Grunbacher P 2004 Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help *IEEE Software* **21(6)** 50-58
[13] Assawamekin N, SunetnantaT and Pluempitiwiriyawej C 2010 Ontology-based

multiperspective requirements traceability framework *Knowledge and Information Systems* **25(3)** 493-522
[14] Robeer M, Lucassen G, van der Werf J M E M, Dalpiaz F and Brinkkemper S 2016 Automated Extraction of Conceptual Models from User Stories via NLP *Proc. Int. Requirements Engineering Conf.* 196-205
[15] Mikhaylov D V, Kozlov A P and Emelyanov G M 2016 Extraction the knowledge and relevant linguistic means with efficiency estimation for formation of subject-oriented text sets *Computer Optics* **40(4)** 572-582 DOI: 10.18287/2412-6179-2016-40-4-572-582

Acknowledgments
The reported study was funded by Russian Ministry of Education and Science, according to the research project No. 2.2327.2017/4.6.