System of Architectural Views on Ontological Maintenance

A. Kulikova, P. Sosnin

Ulyanovsk state technical university, Ulyanovsk, Russia
e-mail: a.push1206@gmail.com, sosnin@ulstu.ru

Abstract. The paper deals with the use of the design thinking approach when creating architectural views which consider motivation and the main goals set for the design phase of the software intensive system (SIS) development. In this case, any architectural view is formed with the help of the system of conceptual equations, which can be solved via the abductive reasoning carried out by a designer. In terms of automated design thinking, such reasoning helps to define constructive relations among motives, goals, and requirements integrated into the corresponding view. For all the views included in an architectural description, such relations are useful to combine, visualize and interpret as motivationally targeted views demonstrating which architectural decisions match the intended goals.

Keywords: Architectural Modeling, Design Thinking, Software Intensive System, Viewpoint.

1 Introduction

Professionally mature development of a modern SIS is unthinkable without the mandatory construction and the operational use of an architecture description (AD). This artifact is a conceptual version of a SIS reflecting its understanding as integrity, which is demanded by stakeholders at all the stages of the SIS lifecycle. Being a sample of verified structures and embedded understanding, the AD plays an executive role providing correspondence between this sample and the current state of the SIS in the design process. It should also be highlighted that this version is the first (the earliest) representation of a SIS as integrity and can be tested to detect dangerous semantic errors.

The abovementioned advantages of using ADs were the reasons for accumulating the experience of architectural modeling intensively that was generalized in several standards; ISO/IEC/IEEE 42010: 2011 is among them. This standard assumes that an AD is the system $S(\{V_j\})$ of "architectural views" $\{V_j\}$ which are built based on corresponding "viewpoints" $\{VP_k\}$. Each viewpoint specifies "the conventions (such as

In: P. Sosnin, V. Maklaev, E. Sosnina (eds.): Proceedings of the IS-2019 Conference, Ulyanovsk, Russia, 24-27 September 2019, published at http://ceur-ws.org

notations, languages, and types of models) for constructing a certain kind of view. That viewpoint can be applied to many systems. Each view is one such application" [1].

Thus, any viewpoint can be interpreted as a certain guide with necessary means that help to build the corresponding view or a set of views, any of which expresses a certain interest (concern C_i) in a visualized form that is understandable for a certain stakeholder involved in the corresponding project. Therefore, any new viewpoint is an artifact that should be developed, starting with the decision to take into account some important concern (or a set of concerns), the viewpoint for which is absent or must be modified.

In this paper, we offer an architectural approach applied in the ontological maintenance toolkit. The toolkit can be used when developing a certain SIS which's lifecycle begins with vague intentions. In this case, firstly, the developers must understand the work beforehand, presenting it as integrity, i.e. as a task that needs to be solved.

The remainder of the paper is structured as follows. The features of design thinking in the considered version of architectural modeling are presented in Section 2. Section 3 points out related works. The approach in the offered viewpoint is described in Section 4. In Section 5, we present the example of a motivationally targeted view, and the paper is concluded in Section 6.

2 Related works

The standard ISO/IEC/IEEE 42010: 2011 envisages that any "concern" may be architecturally described with a coordinated group of useful "points of view" and corresponding "views". For example, the authors of [2] recommend considering such a construct as "architectural decision" by using the Decision Detail viewpoint, Decision Relationship viewpoint, Decision Chronology viewpoint, and Decision Stakeholder Involvement. Moreover, the same authors in the following publication [3] proposed to expand the given set including the Decision Forces Viewpoint in it as well.

Another trick is given in the publication [4], where its authors suggest associating the following concerns with the "Context Description Viewpoint" of the developed SIS:

1. "System Scope: Where is the boundary between the system and its context, and what interactions between the system and its context cross this boundary?
2. System Users: Who are the users of the system; what are their types, roles, and characteristics; and how and where do they access and use the system?
3. External Dependencies: Which external services and/or applications are relevant for the system, including their properties and providers?
4. Execution Environment: What is the expected or desired technical execution environment that the system will be running on?
5. Stakeholder Impact: Which stakeholders, including organizations and their resources, influence the system, and in what way? What influence does the system have on organizations and stakeholders?"

Another means of accounting and materializing “concerns” is their specifications, i.e. the distribution of a set of types and the integration of a set of (distributed) constituent concerns within the framework of “architectural types”. Thus, they associate an “aspect-oriented” representation and the materialization of concerns [5].

The real practice of architectural decisions is discussed in the industrial case study published in [6]. The current retrospection view on the theory and practice of architectural descriptions is presented in the paper [7]

As far as our version of the architectural approach to the ontological maintenance of solving project tasks is concerned, one more group of related works should be considered. These works include papers on the subject area of ontologies. In this group, we highlight the paper [8] which focuses on developing software systems in the context of ontological problems. The paper [9], where a project ontology is applied for the architectural recommendations support, the paper [10] investigating the use of fuzzy measures in selecting architecture tactics and the paper [11] describing maturity modeling in specifications of the architecture maintainability should also be considered.

3 Ontological support application

The application allows processing short text units (i.e. discourses) related to a certain software project with the help of a project ontology.

Processing a discourse is a unit of ontological support for the project. We consider a discourse to be a short text consisting of 2-3 sentences which is a reasoning unit concerning the project. The text may include the requirements to the system or its specifications; in a particular case, it may also be the project general task statement.

The application is integrated into the instrumental environment OwnWIQA with the help of pseudo coding tools.

3.1 Application structure

At the moment, the application has the following structure (see Figure 1). Various interface forms are marked with blue circles; functional and auxiliary modules (files, external applications, etc. used by our application) are marked with yellow circles.

Each interface form is designed with the help of the prototyping tools of the OwnWIQA instrumental environment which allow adding some simple, functional units to the form, such as:

- buttons (with the ability to link it to a pseudocode procedure);
- text fields (the text can be written to the variable and used by a pseudocode procedure);
- other graphic elements.

Moving from one interface form to another is implemented with the help of buttons which call the following pseudocode procedure:

```
DD_Load("diagram_file_path") // open the interface form from a file
DD_LoadEvents("events_file_path") // load events linked to the units of the form
FINISH
```

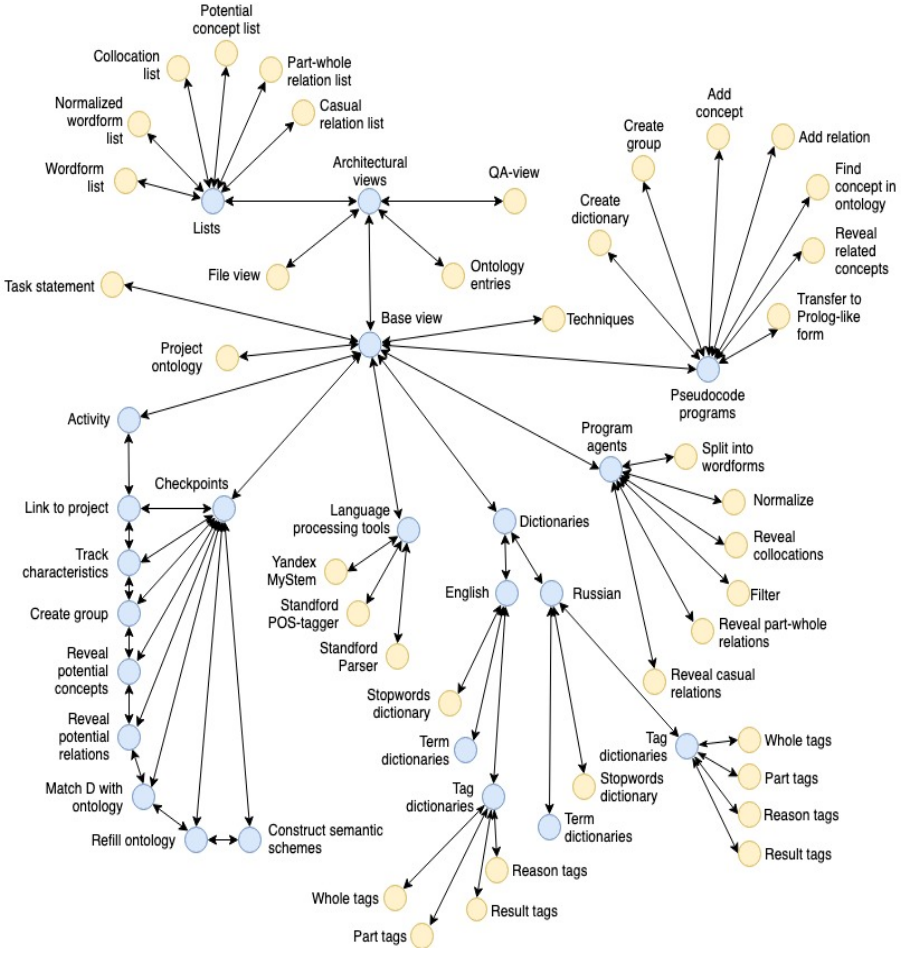

Figure 1. Ontological Support Tool Structure

Let us know describe the functionality of the interface forms and modules one by one.

3.2 Architectural views

The Architectural Views interface form (see Figure 2) shows all the possible views on a discourse being processed. It can be a text saved in a file or the question-and-answer memory of the WIQA environment. It can be a list of ontological concepts found in

the discourse. It can be a semantic scheme built in the graphic editor of the WIQA environment and so on.

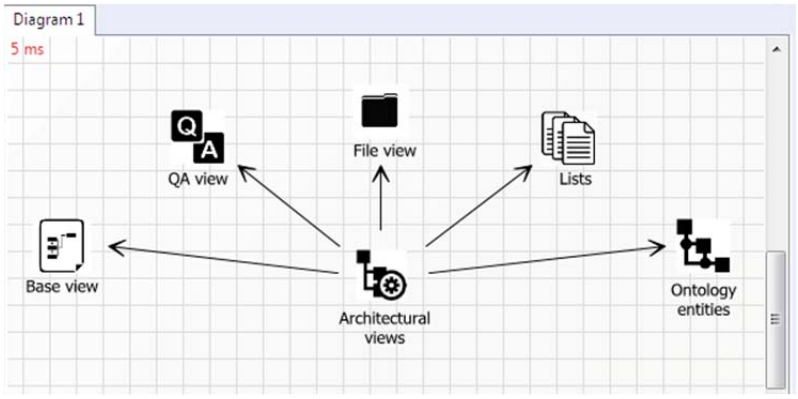

Figure 2. Architectural views

Table 1 shows the actions performed by a double click on each icon (we will use such a table structure for all the interface forms described further).

Table 1. Architectural views

Interface element (icon)	Action on element
Base view	Move to the “Base view” interface form.
QA-view	Open the project linked to the current discourse. If the project has not yet been linked to the current discourse, show a window with the message “Please, link the discourse to a project first” and two buttons (“Link to Project” and “Cancel”).
File view	Open a text file containing changelog of the current discourse.
Lists	Move to the “Lists” interface form.
Ontology entries	Open project ontology.

The changelog file mentioned in the “File view” element has the following meaning: each time the text of a discourse changes, one should add the changes to the changelog file; if there are any changes in the text of discourse, one should add the following record at the end of the file: “[HH:MM DD.MM.YYYY] [Text of discourse]”.

3.3 Base view

The Base View form can be considered to be the main form which a designer deals with. It shows the current version of the discourse as well as all the important conceptual items related to it. At the bottom of the form, one can see all the instruments and additional modules used during the ontological support activity.

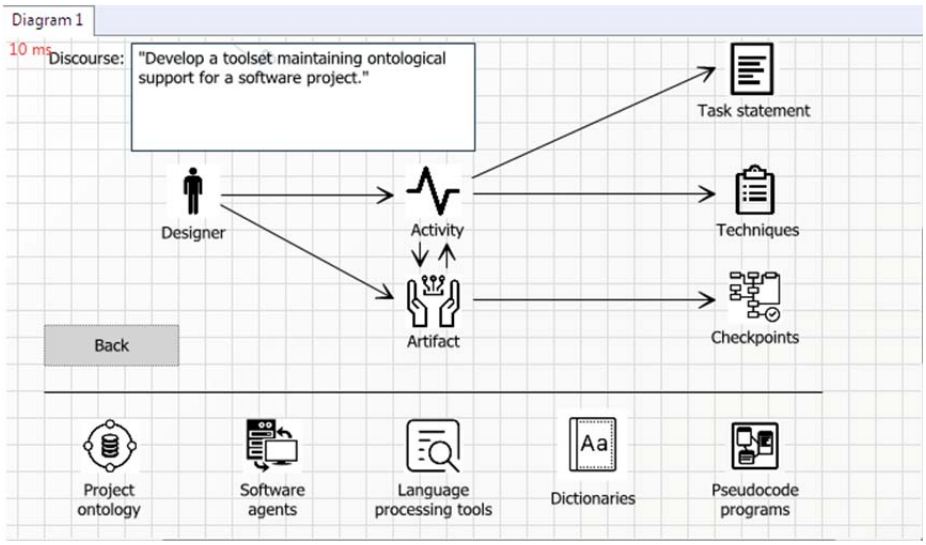

Figure 3. Base view

Table 2. Base view

Interface element (icon)	Action on element
Save	Save the text of the current discourse to the changelog file.
Back	Move to the “Architectural views” interface form.
Designer	Show the description of the required designer skills.
Activity	Move to one of the Activity interface forms according to the workflow.
Artifact	Open a text file containing changelog of the current discourse.
Task statement	Move to the “Task statement” interface form.
Techniques	Open a text file describing possible techniques to process a discourse.
Checkpoints	Move to the “Checkpoints” interface form.
Project ontology	Open the Ontology module in the OwnWIQA environment.
Program agents	Move to the “Program agents” interface form.
Language processing tools	Move to the “Language processing tools” interface form.
Dictionaries	Move to the “Dictionaries” interface form.
Pseudocode programs	Move to the “Pseudocode programs” interface form.

3.4 Task statement

The next interface form plays a substantiating and theoretical role. It describes the task statement, which was used to develop the ontological support toolset (see Figure 4).

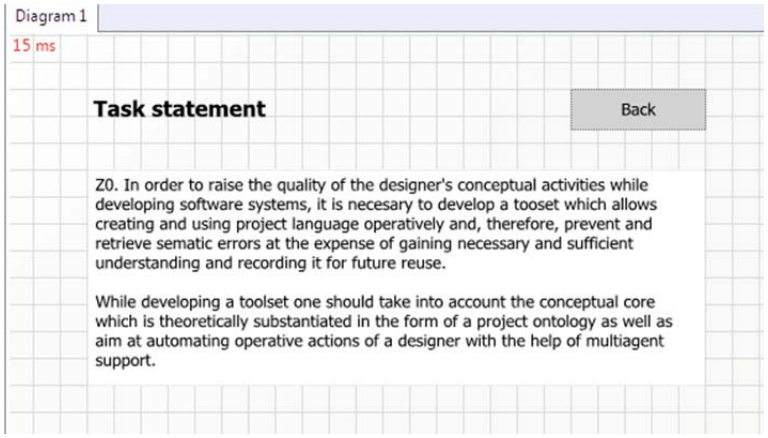

Figure 4. Task statement

Table 3. Task statement

Interface element (icon)	Action on element
Back	Move to the “Base view” interface form.

3.5 Software agents

Our toolkit includes some activities which can be fully automated. So, we decided to use software agents for that purpose. The agents perform such activities as splitting a text into wordforms, normalizing them, retrieving collocations, filtering out stop-words, discovering pairs of semantically related text fragments and others. When clicking on each icon, a designer can launch the corresponding agent which will immediately start processing the current discourse and print the result.

Agents are also used during the main workflow and, in that case, are launched automatically as soon as there are data to process. The interface form in Figure 5 allows launching each software agent manually for demonstration purposes.

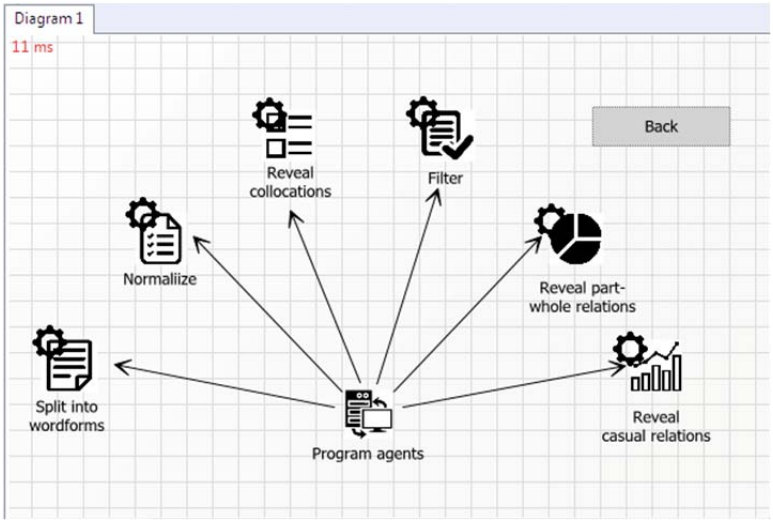

Figure 5. Program agents

Table 4. Program agents

Interface element (icon)	Action on element
Back	Move to the "Base view" interface form.
Split into wordforms	Launch the agent which processes a text and splits it into word-forms. The agent is implemented in C#.
Normalize	Launch the agent which processes wordforms and normalizes them (brings them to their original form). The agent is implemented in C#.
Reveal collocations	Launch the agent which processes a text and reveals collocations in it. The agent is implemented in C#.
Filter	Launch the agent which processes normalized wordforms and filters them out with the help of a stop-word list. The agent is implemented in C#.
Reveal part-whole relations	Launch the agent which processes a text and reveals phrase pairs in it linked by a part-whole relation. The agent is implemented in C#.
Reveal casual relations	Launch the agent which processes a text and reveals phrase pairs in it linked by a causal relation. The agent is implemented in C#.

3.6 Language processing tools

Language processing tools are external auxiliary programs integrated into the ontological support toolset to raise the efficiency of text processing.

We use part-of-speech taggers and parsers for the Russian and the English languages to extract linguistic information from a text and use it for our purposes.

The corresponding interface form is available in Figure 6.

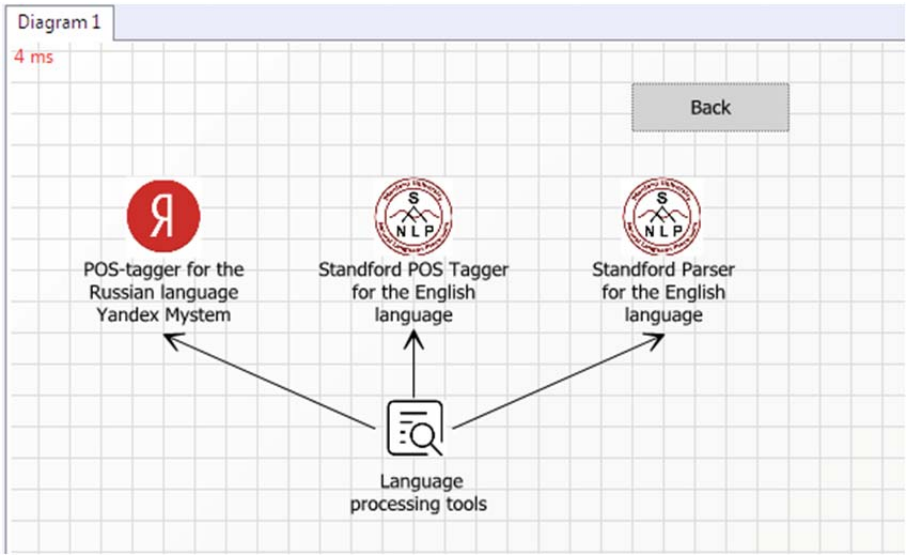

Figure 6. Language processing tools

Table 5. Language processing tools

Interface element (icon)	Action on element
Back	Move to the “Base view” interface form.
POS-tagger for the Russian language Yandex Mystem	The tool splits the text in Russian into wordforms and normalizes them. It also has the functionality to reveal the part of speech of each wordform if needed and print the result in various formats (for details see [12]).
Stanford POS Tagger for the English language	The tool splits the text in English into wordforms and normalizes them. It also has the functionality to reveal the part of speech for each wordform if needed and print the result in various formats (for details see [13]).
Stanford Parser for the English language	The tool reveals syntactical relations from a text (for details see [14]).

3.7 Dictionaries

The next interface forms were designed to demonstrate all the auxiliary dictionaries that the ontological support tool uses. Since our application allows processing texts both in English and in Russian, first of all, a user has to choose the language (see Figure 7).

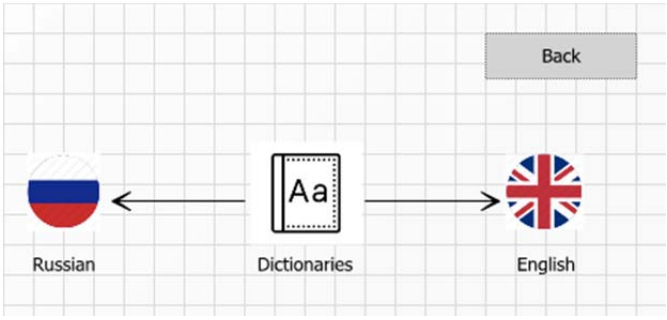

Figure 7. Dictionaries

Table 6. Dictionaries

Interface element (icon)	Action on element
Back	Move to the “Base view” interface form.
Russian	Move to the “Russian Dictionaries” interface form.
English	Move to the “English Dictionaries” interface form.

For each language, we have three types of dictionaries: stop-word dictionaries, term dictionaries (can be optionally added by a user if he/she is working with some specific domain) and tag dictionaries (used for retrieving semantic relations). The interface form showing dictionary types for the English language is presented in Figure 8.

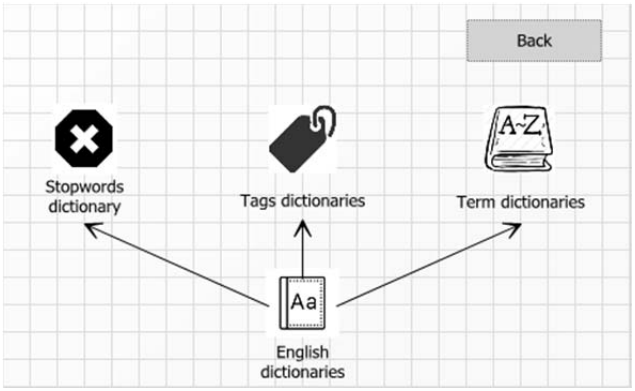

Figure 8. English dictionaries

Table 7. Dictionaries

Interface element (icon)	Action on element
Back	Move to the “Dictionaries” interface form.
Stop-word dictionary	Open the file with English stop-words.

Tag dictionaries	Move to the “Tags dictionaries” interface form.
Term dictionaries	Open the window to connect term dictionaries if needed.

We designed a separate interface form for tag dictionaries so that a user could check which tags the tool uses to retrieve semantic relations from a text and add or remove some of them if needed. Tags are such words or collocations that signalize and highlight if there are relations of a certain type in the current phrase.

At the moment, the tool can retrieve two types of semantic relations: part-whole and casual ones because they are most useful for building prototypes. Correspondingly, four types of tags are available in the form (see Figure 9).

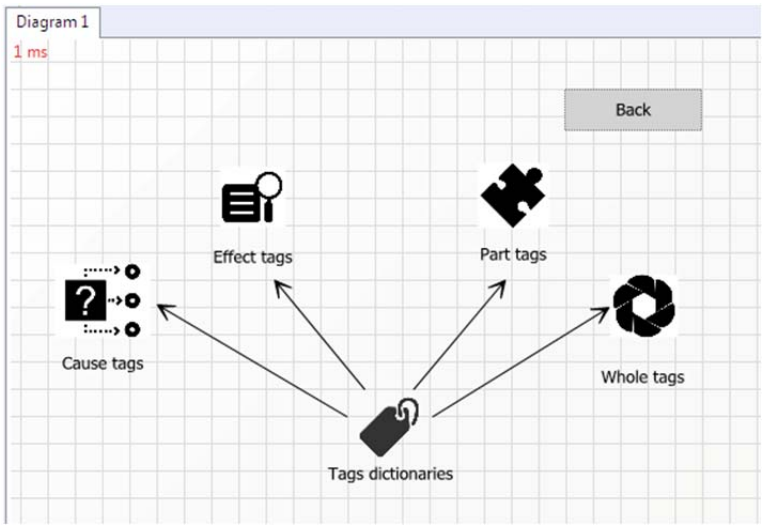

Figure 9. Tags dictionaries

Table 8. Tags dictionaries

Interface element (icon)	Action on element
Back	Move to the “English dictionaries” or “Russian dictionaries” interface form depending on the logic.
Cause tags	Open a text file containing the cause tag list.
Effect tags	Open a text file containing the effect tag list.
Part tags	Open a text file containing the part tag list.
Whole tags	Open a text file containing the whole tag list.

3.8 Pseudocode programs

Along with the modules in C#, we use separate procedures and functions developed with the help of the pseudocode programming module of the OwnWIQA instrumental

environment. This helps to make our toolset more flexible since any pseudocode procedure can be integrated into the tool without changing its architecture.

Furthermore, pseudocode programs have simple syntax and can easily be understood by any person, which also makes the work of the toolset more transparent.

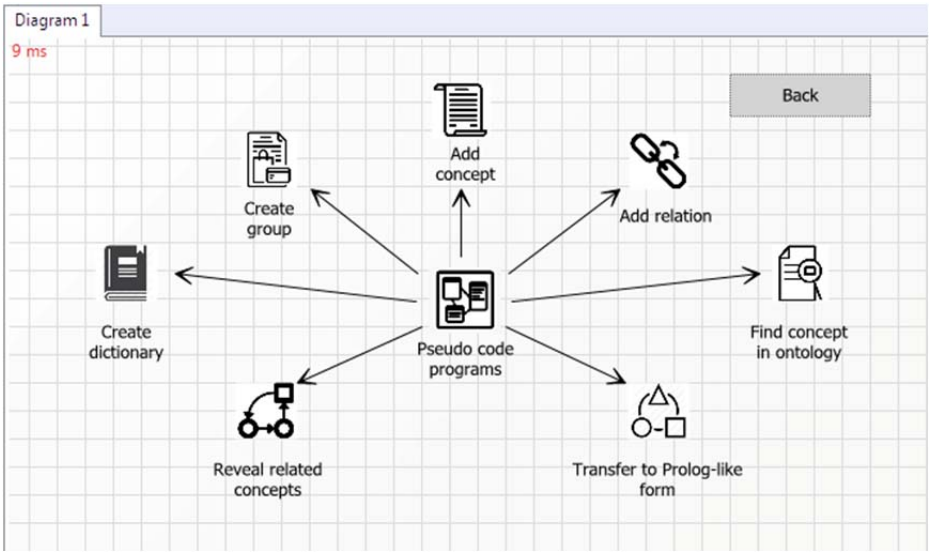

Figure 10. Pseudocode programs

Table 9. Pseudocode programs

Interface element (icon)	Action on element
Back	Move to the “Base view” interface form.
Create a dictionary	Execute the pseudocode procedure which creates a new dictionary in the ontology module with a name provided by the user.
Create a group	Execute the pseudocode procedure, which creates a new group in the project dictionary with a name provided by the user.
Add concept	Execute the pseudocode procedure, which adds a new concept to the group selected by the user.
Add relation	Execute the pseudocode procedure which adds a new relation between 2 concepts selected by the user. The type of the relation is also specified by the user.
Find concept in the ontology	Execute the pseudocode procedure, which checks if the current concept has already been added to the project ontology.
Reveal related concepts	Execute the pseudocode procedure, which checks if two selected concepts have a semantic relation between them added to the project ontology.
Transfer to Prolog-like form	Execute the pseudocode procedure, which transfers the desired set of related concept pairs to the Prolog-like form which can be

used for automated building semantic schemes [15].

3.9 Lists

At various stages of processing a discourse, different lists are created. Lists are usually the results of the software agents' work, i.e. they can be used both as the input and the output data of these agents. Figure 11 shows all the possible lists a user can deal with.

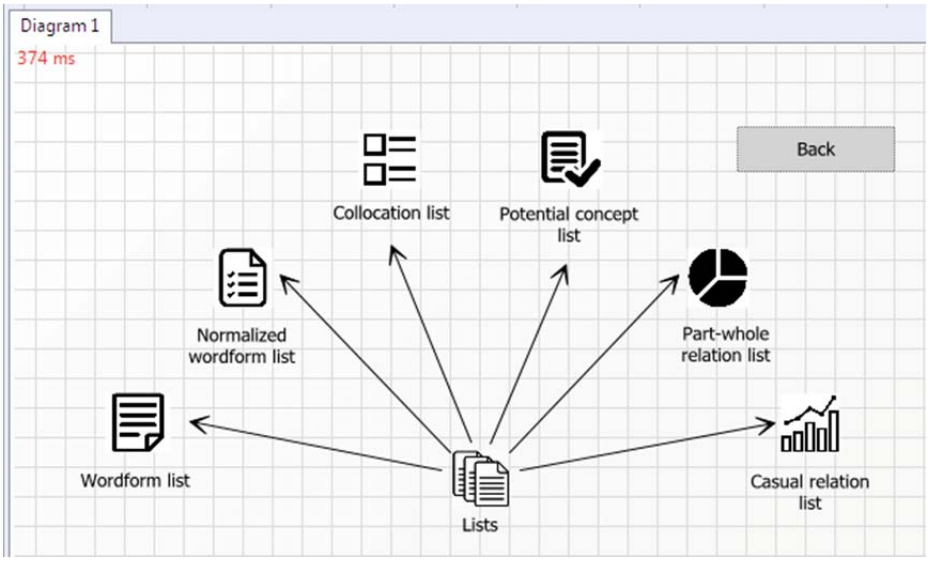

Figure 11. Lists

Table 10. Lists

Interface element (icon)	Action on element
Back	Move to the "Architectural view" interface form.
Wordform list	Open a list in a text file. The list contains all the wordforms of the current discourse.
Normalized wordform list	Open a list in a text file. The list contains all the wordforms of the current discourse in their initial form.
Collocation list	Open a list in a text file. The list contains all the collocations of the current discourse (retrieved with the help of a set of rules).
Potential concept list	Open a list in a text file. The list contains all the word and collocations of the current discourse, which can be considered to be concepts and can be potentially added to the project ontology.
Part-whole relation list	Open a list in a text file. The list contains pairs of phrases of the current discourse linked by a part-whole relation.
Casual relation list	Open a list in a text file. The list contains pairs of phrases of the

current discourse linked by a causal relation.

3.10 Activity

This subsection describes the workflow that a user of the ontological support tool has to undergo to process one discourse.

Checkpoints. While being processed, a discourse can often be edited and corrected (if any errors are revealed). Thus, it is important for a user to have an opportunity to track all the changes in the discourse and revert to one of its previous versions if needed.

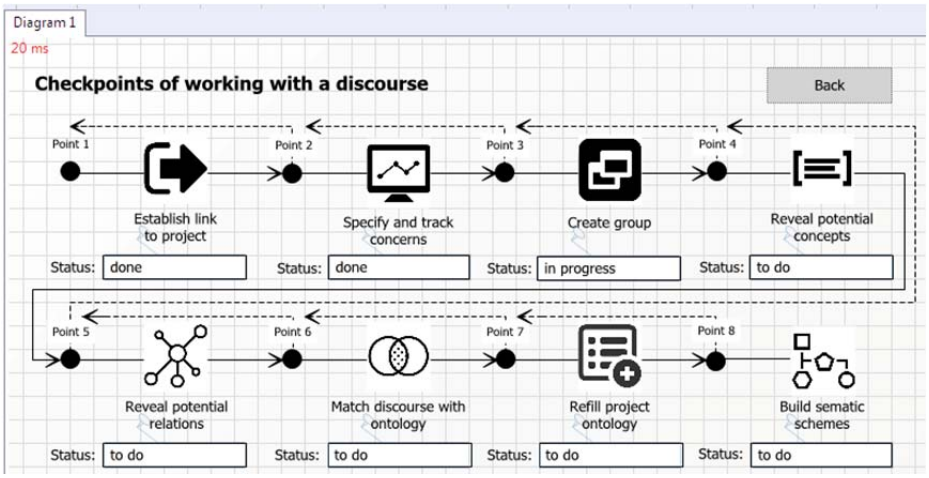

Figure 12. Checkpoints

The interface form in Figure 12 shows all the eight steps a user has to undergo one by one when working with a discourse. Apart from that, at any phase he/she can easily go back to the previous one.

The form also shows the status of each phase which can have three states:

- to do (if a user has not started working on it yet);
- in progress;
- done.

Double click on each phase icon starts a corresponding activity. Each activity will be described further.

Establish a link to the project. While a project is being designed, its language is being constructed. This language is unique for each project. Moreover, the project language is changing from phase to phase and requires to be registered. To register the project language, we use a separate dictionary in the ontology module of the OwnWIQA environment. The dictionary is constructed in the form of ontology and

allows storing concepts related to the current project, their definitions as well as different types of semantic relations between them.

So, the first step of processing a discourse is to establish a link between the discourse and the project it relates to, i.e., choose the corresponding dictionary in the ontology module (Figure 13 shows the interface form designed for that purpose). If it is the first discourse a user deals with within the current project, a new dictionary has to be created. From this point on, all the changes made in the discourse will be stored in this dictionary. Otherwise, a corresponding dictionary has to be selected

At this phase (as well as at all the further ones) a user can make any changes in the text of discourse and see the changelog if needed.

After establishing a link to a project, one can click “Next” and move to the next phase or click “Back” and move back to the checkpoints.

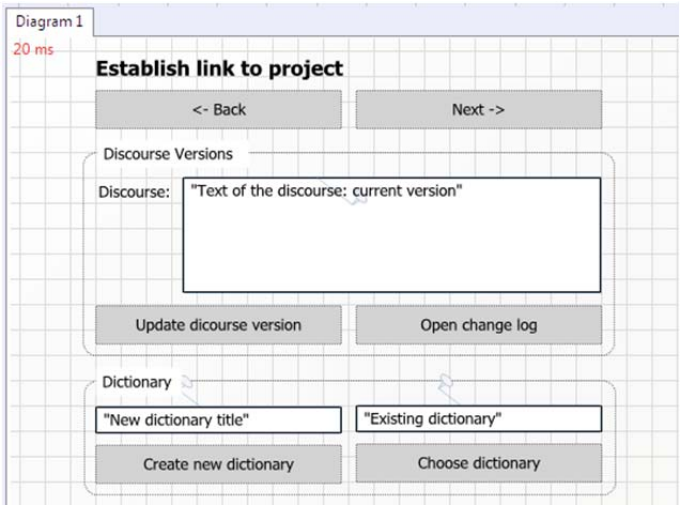

Figure 13. Establish a link to the project

Specify and track concerns. When working on a project a designer may need to track some quality indicators (for example, understandability, testability, flexibility, and others), i.e. concerns or requirements of the main project stakeholders. Each concern can have its membership function which shows how its quality lever changes at different project design phases.

Our tool allows tracking these concerns and saving them in a separate group of the project ontology. The principles of this activity, as well as some examples, are given in paper [16].

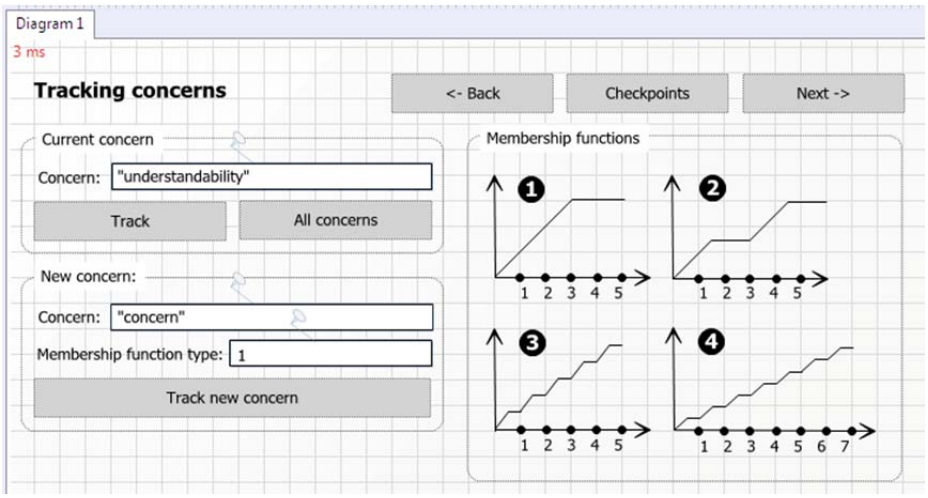

Figure 14. Specify and track concerns

Create group. When we start working with a new discourse, we need to create a separate group in the project ontology where all the concepts related to this discourse will be stored. Figure 15 shows the interface form, which allows creating a new group in the ontology or make some changes in the text of discourse if needed.

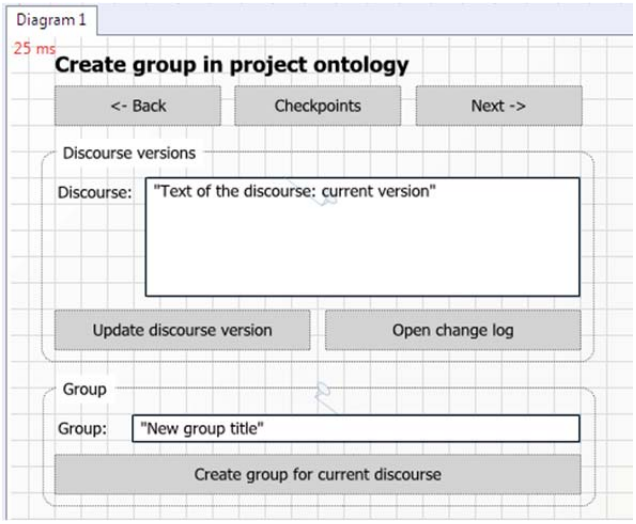

Figure 15. Create a group

Reveal potential concepts. This phase is one of the most important activities in the ontological support process. Let us consider what has to be done at each subphase:

1) Split the text into wordforms. Double-click on this icon launches the software agent which splits the text into wordforms by spaces and punctuation marks.
2) Normalize wordforms. Double-click on this icon launches the software agent which gets the initial form of each word (for example, “projects” become “project,” “had” becomes “have,” etc.).
3) Filter out stopwords. Double-click on this icon launches the software agent which removes stopwords (i.e., words that do not have any significant semantic meaning) from the list of normalized wordforms.
4) Reveal collocations. Double-click on this icon launches the software agent which uses a rule-based approach and syntactic models to get all the possible collocations from the text. This subphase was added to the tool because a potential concept can be not only a word but also a collocation.
5) Get a list of potential concepts. Uses the result of the subphases 3.3 and 3.4 as well as additional term dictionaries (if any are linked to the project) to form a list of words and collocations that could be included in the project ontology.

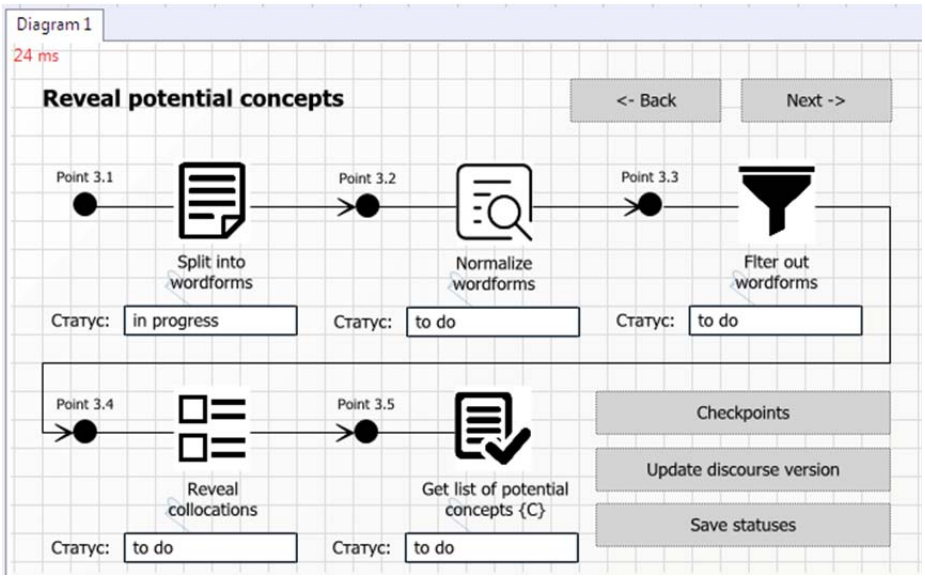

Figure 16. Reveal potential concepts

Reveal potential relations. The next phase is to reveal possible relations between the concepts presented in the text. In the current toolkit version, we reveal part-whole relations and casual relations since they are most useful to build semantic prototypes.

The algorithm of revealing semantic relations is based on tags which “highlight” relations of a certain type in the text.

After the relations are revealed, lists of related phrases are matched with the list of potential concepts got at the previous phase – as a result, we get a list of related concepts.

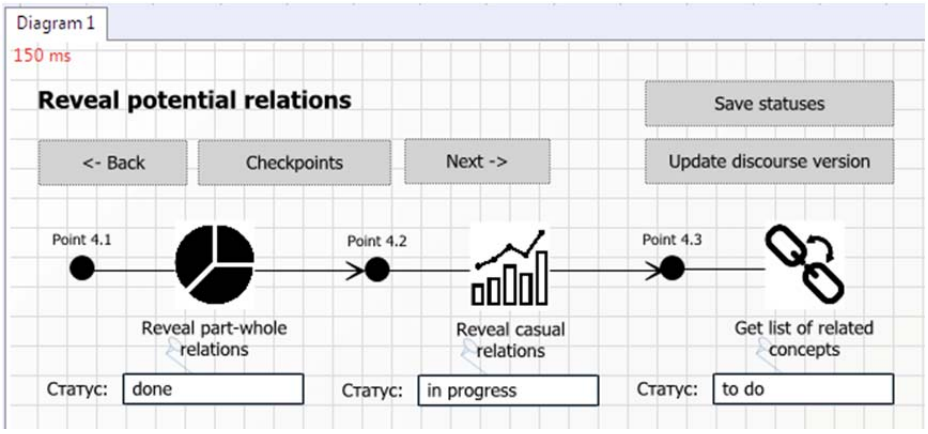

Figure 17. Reveal potential relations

Match discourse with ontology. At this phase, all the concepts revealed form the text of discourse are matched with the project ontology. If the project ontology already contains any of them, all the information related to these concepts (definitions, related concepts, etc.) is retrieved from the ontology and shown to the user, so that he/she could correct possible mistakes and inaccuracies.

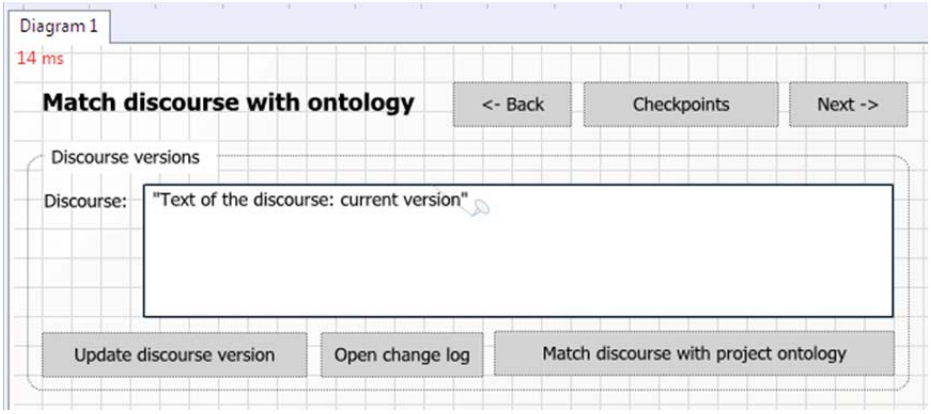

Figure 18. Match discourse with ontology

Refill project ontology. Finally, new information should be added to the project ontology. This can be done either manually or with the help of the lists of potential concepts and relations created at previous phases.

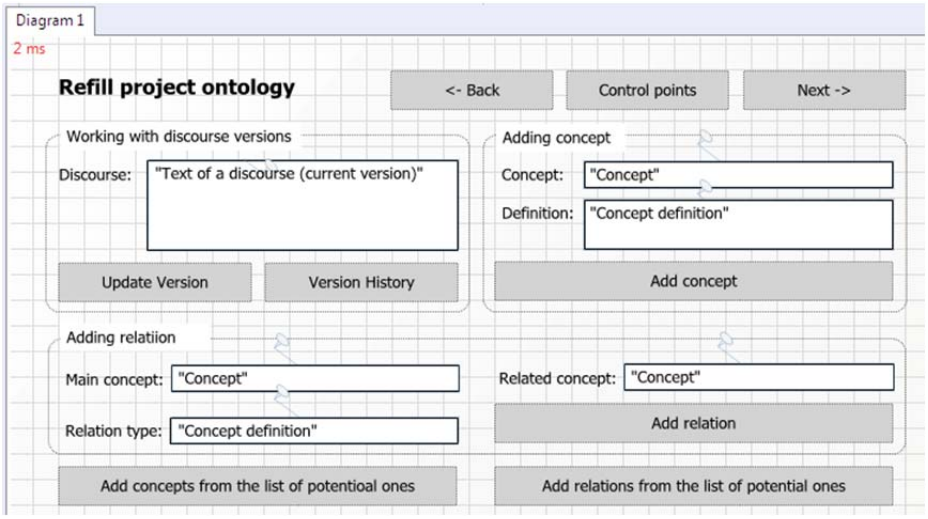

Figure 19. Refill project ontology

4 Conclusion

Creating and using architectural models is the key factor in the successful development of modern SISs. Such models register the necessary understanding, reflecting corresponding essences as integrity, which is especially important for architectural views combining semantized graphics with necessary symbolic descriptions written in the project language.

In the offered approach, a designer can develop any view in the process of solving an architectural task with the use of automated design thinking, means of which are embedded into the WIQA toolkit. Among these means, the specialized graphical editor plays a very important role. This editor helps to detailed visualized structures that express not only separate views but also their programmed compositions.

We apply this approach to improve the existed version of the ontological maintenance (OM) embedded into the WIQA toolkit. The current version of the OM is implemented in the prototype form, graphical elements of which are created by means of the graphic editor with indexed references when it is necessary. In other words, functions of the OM are accessible to a designer via interfaces any of which is the prototype version of the corresponding architectural view on the OM.

References

1. Standard ISO / IEC / IEEE 42010: 2011, Available at https://www.iso.org/standard/50508.html
2. Sosnin, P.: Experience-Based Human-Computer Interactions: Emerging Research and Opportunities, IGI-Global, (2017).

3. Sosnin, P.: Substantially Evolutionary Theorizing in Designing Software-Intensive Systems, *Information Vol. 9*(4), 1-29 (2018).
4. Dorst, K.: The Nature of Design Thinking, in DTRS8 Interpreting Design Thinking, In Proceeding of Design Thinking Research Symposium, pp. 131–139, (2010).
5. Bedjeti, A.; Lago, P.; Lewis, G. A.; Boer, R. D. D.; Hilliard, R.2017. Modeling Context with an Architecture Viewpoint, In Proceeding of IEEE International Conference on Software Architecture (ICSA), pp. 117-120, 2017.
6. Van Heesch, U.; Avgeriou, P.; Hilliard, R.: 2012. Forces on Architecture Decisions - A Viewpoint. In *Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, IEEE Computer Society, Washington, DC, USA, pp. 101-110, 2012
7. Van Heesch, U.; Avgeriou, P.; Hilliard, R.: 2012. A documentation framework for architecture decisions. *J. Syst. Softw.* Vol. 85(4), pp. 795-820, 2012.
8. Hilliard, R.: 2007. Using Aspects in Architectural Description, Lecture Notes in Computer Science, Vol. 4765, pp. 65-68, 2007.
9. Dasanayake, S.; Markkula, J.; Aaramaa, S.; Oivo, M.: 2015. *Software Architecture Decision-Making Practices and Challenges: An Industrial Case Study,* In Proc. of 24th Australasian Software Engineering Conference, 2015, pp. 88-97.
10. Hasselbring, W.: 2018. Software Architecture: Past, Present, Future, *The Essence of Software Engineering*, Springer, pp. 168-184
11. Eden, A. H., Turner, R.P: Problems in the Ontology of Computer Programs (Applied Ontology, vol 2 1, Amsterdam, IOS Press), pp. 13–36, (2007).
12. Ilya Segalovich. A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine. URL: https://cache-nnov05.cdn.yandex.net/download.yandex.ru/company/iseg-las-vegas.pdf.
13. Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003, pp. 252-259.
14. Marie-Catherine de Marneffe, Bill MacCartney and Christopher D. Manning. 2006. Generating Typed Dependency Parses from Phrase Structure Parses. In LREC 2006.
15. Sosnin, P.; Galochkin, M. Way of Coordination of Visual Modeling and Mental Imagery in Conceptual Solution of Project Task. In Advances in Artificial Intelligence: From Theory to Practice; Lecture Notes in Computer Science; Springer: Cham, Switzerland,, 2017; Volume 10350, pp. 635–638.
16. Ontology-Based Specifications of Concerns in Architectural Modeling of a Software Intensive System. 2018 26th Telecommunications Forum (TELFOR). Proceedings of Papers. Belgrade, Serbia, November, 20-21, 2018. – p. 843-845.

∗ This work was supported by the Russian Fund for Basic Research (RFBR), Grant #18-07-00989a, Grant # 18-47-73001r-a, and the State Contract №2.1534.2017/4.6