Received 20 October 2015
Accepted 18 November 2015



# A place and role of an ontology in using a base of experience in designing the software intensive systems

Petr Sosnin
*Computer Department, Ulyanovsk State Technical University, Ulyanovsk, Russian Federation*

## Abstract

**Purpose** – Nowadays, experience bases are widely used by project companies in designing software-intensive systems (SISs). The efficiency of such informational sources is defined by the "nature" of modeled experience units and approaches that apply to their systematization. This paper aims to increase the efficiency of designing the SISs by the use of an ontological support for interactions with an accessible experience, models of which are understood as intellectually processed conditioned reflexes.

**Design/methodology/approach** – Both of the base of experience (BE) and ontological support in interactions with its units are oriented on precedents built in accordance with the offered normative schema when the occupational work is fulfilled by a team of designers. In creating the BE and the ontology as part of the BE, the team should use a reflection of an operational space of solved tasks on a specialized semantic memory intended for simulating the applied reasoning of the question-answer type.

**Findings** – If the occupational space of designing is reflected on the semantic memory with a programmable shell, then this environment can be adjusted on simulating the intellectual mechanisms flown in a human consciousness when designers ontologically interact with the BE and tasks being solved. The use of simulating the process in consciousness in accordance with their nature facilitates increasing the efficiency of designing the SIS.

**Research limitations/implications** – An orientation on a precedent model as a basic type of experience unit and an ontological approach to their systematization are defined by the specificity of the study described in this paper. Models of precedents are constructed in accordance with the normative schema when the occupational work is fulfilled by a team of designers.

**Practical implications** – Investigated and developed means of ontological support are oriented on effective designing of the SISs with the use of the toolkit Working In Questions and Answers (WIQA) by the team of designers. The achieved effects are aimed at increasing the level of success in collaborative designing of SISs.

**Social implications** – Offered solutions are applicable in designing the systems which supported different relations of a human with artificial and natural environment. They facilitate the naturalness in interactions of a human with computerized world.

**Originality/value** – An orientation on the precedent model as a basic type of experience unit and the ontological approach to their systematization are defined by the specificity of the study described in this paper. The novelty of this approach is defined by the framework for the precedent model, understood as the intellectually processed conditioned reflex, in which a reflection on the semantic memory (of the

An earlier version of this paper was presented at the 15th International Conference on Computational Science and its Applications (ICCSA' 2015).

International Journal of Web Information Systems
Vol. 12 No. 1, 2016
pp. 62-82

1744-0084
DOI 10.1108/IJWIS-10-2015-0035

question-answer type) is programmable in a conceptually algorithmic language. The ontological support is implemented in the environment of programming.

**Keywords** Ontology, Precedent, Question-answering, Software intensive system, Automated designing

**Paper type** Case study





## 1. Introduction

Past 20 years, the problem of an extremely low degree of success in designing of software-intensive system (SIS) is steadily registered in statistics that are presented in reports (Chaos Manifesto, 2013) of the company "Standish Group". The analysis of the successfulness problem indicates that a special attention should be given to the human factor in attempts to increase the degree of success. Designing the SIS is the intensive human–computer activity that is fulfilled in conditions of a high level of complexity.

It is necessary to note that designing (especially conceptual designing) is impossible without researching by designers of numerous and practically unpredictable situations of a task type. In such situations, the designer should behave as a researcher who uses the appropriate type of research, in which the personal and collective experience is creatively applied in real time.

In designing the definite research, the designer would rather work as a scientist who wants to conduct the definite experiment. Results of such experiment will be applied in the creation of the current project and can be useful for the future reuse. For this reason, the search for improved forms of designer interactions with own and collective experience is a promising way of increasing the degree of success in designing SIS.

Four years ago, a group of well-known researchers and developers in software engineering had initiated a process of innovations (Jacobson *et al.*, 2012), which have been named Software Engineering Methods And Theory (SEMAT). It is necessary to notice that in normative documents of SEMAT, a way of working used by the team of designers is marked as a very important essence. There "way-of-working" as a notion is defined as "the tailored set of practices and tools used by the team to guide and support their work". For this reason, the search of effective ways-of-working is perspective and can lead to an increase of success in developments of SISs.

A very promising way-of-working that will facilitate increasing the success in collective designing of SIS can be bound with creation and used by the team as an experience base (Henninger, 2003). It is necessary to note that the effectiveness of such version of working essentially depends on the kinds of experience models and ways of their systematization and use.

The paper presents an ontological approach to the systematization of the experience base and interactions with its units. Also, such units are formed as experience models corresponding to precedents. According to the Cambridge dictionary:

> […] precedents are actions or decisions that have already happened in the past and which can be referred to and justified as an example that can be followed when the similar situation arises (http://dictionary.cambridge.org/ dictionary/ british/ precedent).

Methods and means that are offered in the paper are oriented on conceptual experimenting with tasks being solved in an instrumentally technological environment Working In Questions and Answers (WIQA) (Sosnin, 2013). The interaction with the toolkit WIQA is based on question-answer reasoning (QA-reasoning) of designers.

## 2. Preliminary bases

### *2.1 Why ontology*

Ontology usually helps to achieve the following positive effects: reaching the coordinated understanding in collective actions; using the controlled vocabulary; systematizing the methods and means used in an occupational activity; specifying the conceptualization; checking the semantics of the built text and applied reasoning; operating with machine-readable and machine-understandable content.

In this paper, the use of the ontology is oriented on all called effects and the systematization of precedents' models embedded into the experience base. Also, it is planned to create such models with the use of conceptual experimenting when designers solve the project tasks in real time. This way is coordinated with an understanding of the ontology as a system of specifications of conceptualization (Garcia *et al.*, 2003).

The called effects are important as for the collective activity and so for the activity of one person (individual). Moreover, in a team, a coordinated understanding is achieved as a result of coordinating a personal understanding with a common understanding for all members of the team.

### *2.2 Why way-of-working oriented on precedent*

As a reusable unit of the occupational work, the precedent should be specified and executed by an appropriate framework. Such a framework as an artifact should be coordinated with the natural reflection of the environmental conditions of the precedent execution by an individual. If the result of the reflection is bound with an understanding that the precedent is a behavior that corresponds to a intellectually processed conditioned reflex, then such an understanding can be used for its specification in the precedent framework.

The toolkit WIQA supports the use of the precedent framework that is presented in Figure 1, where a logical component of the framework demonstrates (without explanations) some details of intellectual processing of the precedent as a behavioral unit.

The structure of the framework is coordinated with the process of task-solving and preparing the solution of the task for reuse in designing. This structure includes a

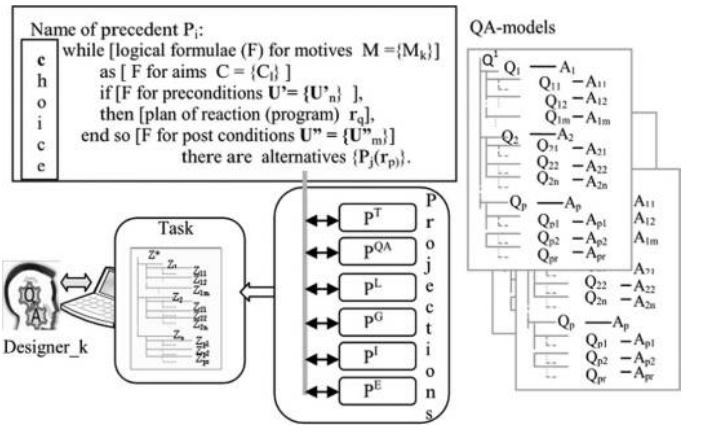


**Figure 1.**
Framework of precedent

textual model $P^T$ of the solved task Z, its model $P^{QA}$ in the form of the registered QA-reasoning about the solution, the logical formulae $P^L$ of the precedent regularity, a graphical (diagram) representation $P^G$ of the precedent on the perceptual level, conceptually algorithmic model $P^I$ in a form of a pseudo-code program and the model which presents its executable code $P^E$.

### *2.3 Why question answering*

Intellectual processing for natural precedents is being fulfilled using natural language that also supports access to units of the experience. Such effects are caused by processes in consciousness which have a dialog nature. Explicit QA reasoning helps in controlling of these processes. Thus, QA reasoning reflects processes in the consciousness of the designer interacted with own experience. This type of reasoning is expedient for applying in the creation and use of experience models that present the designer's behavior.

In reality, designers usually represent the own professional work using a plan to be written in the natural language in its algorithmic usage. Repeatable works fulfilled by individuals are represented by techniques also being written in the natural language in its conceptually algorithmic usage. It prompts to use such way for plans of experimenting. In turn, that was the reason for choosing a pseudo-code language for programming the units of designer's behavior. Interacting with such programmable models of the behavior, the designers will apply the experience of using the natural language.

## 3. Related works

A set of typical kinds of ontologies (according to their level of dependence on a particular task or point of view) includes the top-level ontologies, domain ontologies, tasks ontologies and applied ontologies. All these types of ontologies are defined in Garcia *et al.* (2003) and Guarino (1998) as means that are used in different systems.

For the SISs, an adequate type of ontologies is the applied type that usually is expanded using the other ontology types. By Guarino *et al.* (2009), the theory and practice of applied ontologies "will require many more experiences yet to be made".

It is necessary to notice that the project ontology as a sub-type of applied ontologies is essentially important for SISs. Project ontologies in the greater measure are aimed at the process of designing, but after refining, they can be embedded into implemented SISs.

The specificity of project ontologies is indicated in some publications. In the technical report (Garcia *et al.*, 2003), the main attention is concentrated on "people, process and product" and collaborative understanding in interactions. Investigating the possibility of the ontology-based project management is discussed in the paper by Bullinger *et al.* (2005).

In developing the program system and ontological problems of program products, the use of ontology possibilities is investigated in the paper by Eden and Turner (2007). In the means suggested in this article, the experience of task ontologies is taking into account also, and first of all, the role of different kinds of knowledge. The place of knowledge into the task ontologies is reflected and discussed in the study by Martins and De Almeida (2008). The role of knowledge connected with problem-solving models is presented in the papers by Fitsilis *et al.* (2014).

All papers indicated in this section were used as sources of requirements in developing the set of instrumental means provided the creation and use of the proposed ontology. It should be noted that the reports of Standish Group were used as "guides" for the planning of our research aimed at applications of QA-approach and the development of the toolkit WIQA. Step by step the QA-reasoning approach has investigated criteria and factors that facilitate increasing the degree of the success in designing of SISs (Sosnin, 2013). The study presented in this paper focuses on the ontological component of QA-reasoning approach at the use of the occupational experience. This paper consists of two parts, one of which was reported in the 15th International Conference on Computational Science and Applications (Sosnin, 2015). The second part expands on the first by describing two versions that help to discover the new components for their inclusion in the used ontology (Points 5.2 and 5.3).

## 4. Reflection of subject area

### *4.1 Question-answer memory*

As mentioned above, the proposed study is aimed at the creation and use of the project ontology by the designers who develop the family of SIS in the definite subject area. In such work, they should use (if they solved) the toolkit WIQA, which helps to include conceptual experimenting in processes (P) of designing.

At the conceptual stage of working, the means of WIQA are used by designers for the following aims:

- registering of the set of created projects, each of which is presented by the tree of its tasks in the real time;
- parallel implementation of the set of workflows $\{W_m\}$, each of which includes subordinated workflows and project tasks $\{Z_n\}$;
- pseudo-parallel solving of the project tasks in each workplace in the corporate network; and
- simulating the typical units of designers' behavior with using the precedent framework.

Named actions are implemented by designers by using the reflection of processes P on QA-memory, some details of which are presented in Figure 2.

For the process $P_l$ of designing the project $PROJ_l$, all indicated aims are achieved by the following reflections:

$$\begin{aligned} WW(P_l) \;=\; & WW(PROJ_l, \{W_m\}, \{Z_n\}, t) \xrightarrow{R^{QA}(P)} ZP_l^{QA}(t + \Delta t) \cup \\ & \cup \{ZW_m{}^{QA}(t + \Delta t)\} \cup \{Z_n{}^{QA}(t + \Delta t)\}, \\ & \{Z_n{}^{QA}(t) \xrightarrow{R^{QA}(Z^{QA})} Pr_n{}^{QA}(t + \Delta t)\}, \end{aligned} \tag{1}$$

where symbol $Z$ underlines that labeled models have a task type; $Pr^{QA}(t)$ designates a model of a precedent for the corresponding task; $R^{QA}(X)$ indicates that the reflection $R^{QA}$ is applied to the entity or artifact $X$. For example, $R^{QA}(Z^{QA})$ designates applying this reflection to the model $Z^{QA}$ of the task $Z$. In WIQA

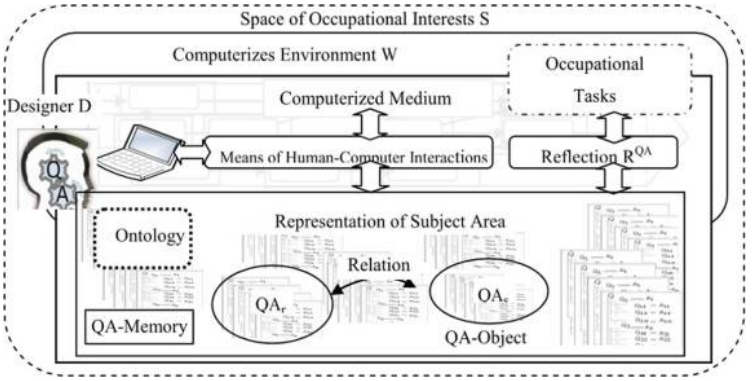


**Figure 2.**
Operational space

environment, the model of such kind is called as "QA-model of the task". It is necessary to note that the reflection $R^{QA}$ includes $R^{QA}(P)$, results of which are reflected by $R^{QA}(Z^{QA})$ in a set of precedents' models $\{Pr_n^{QA}\}$ also located in the QA-memory.

QA-memory is specified and materialized for storing the conceptual descriptions of the operational space $\boldsymbol{S}$ in memory cells in an understandable form. For this reason, cells are oriented on registering the textual units in the form of communicative constructions, any of which can be divided on "theme" (predictable starting point) and "rheme" (new information) that should receive additional meaning in answering to the corresponding question.

Such orientation has led to the solution of using two types of corresponding cells. The cell of the first type is intended for registering the simple question (Q) during the cell of the second type registers the corresponding answer (A).

Extracting the theme and rheme from units of the textual description is an important linguistic operation named "actual division". This operation facilitates the process of understanding. Its use corresponds to the dialogic nature of consciousness. Cells of both types are specified equally because any question includes a part of a waited answer, and any answer includes components of the corresponding question. For this reason, a pair of corresponding cells with a question and answer presents the description unit in details. Such pair corresponds to the QA-object in Figure 2.

QA-memory with its cells are intended for registering the conceptual content of reflected units by taking into account the semantics of their textual descriptions. The necessary semantics is fixed in basic attributes of the cell in additional attributes that can be added by the individual if it will be useful for the simple object stored in the cell. The potential structure of a simple object is presented in Figure 3.

The above not only indicates reflections of projects on the QA-memory, but it also demonstrates structures that should find their presentations in such memory. Below, for specifications of QA-objects in the QA-memory, a formal grammar $\boldsymbol{GR^{QA}}$ with extended BNF-notations will be used. For example, the structures of QA-objects should correspond to the following rules of $\boldsymbol{GR^{QA}}$:

$$\left.\begin{array}{l} QA - Memory = \{QA - object\}; \\ QA - object = Question, \text{“} \leftarrow \text{”}, Answer; \\ Question = Q\backslash(Q, \text{“} \downarrow \text{”}, \{Q\}); \\ Answer = A\backslash(A, \text{“} \downarrow \text{”}, \{A\}); \\ Q = (\{a\}, \{[aa]\}, \{[f]\}, \{[p]\}\{[r]\}); \\ A = (\{a\}, \{[aa]\}, \{[f]\}\rangle\{[p]\}\{[r]\}); \\ a = (address, type, description, time, the\ others\rangle; \\ aa = \{useful\ additional\ attribute\}, \end{array}\right\} \quad (2)$$

where "***Q***" and "***A***" are typically visualized objects stored in cells of QA-memory and symbol " ↓ " designates an operation of "subordinating". These objects have the richest attribute descriptions (Garcia *et al.*, 2003). For example, a set of attributes includes the textual description, index label, type of objects in the QA-memory, the name of a responsible person and the time of last modifying. Any designer can add necessary attributes to the chosen object by the use of the special plug-ins "Additional attributes" (object-relational mapping to C#-classes).

Also, in QA-memory, any created model is materialized by using the QA-objects that are bound by necessary relations. Any QA-object is a corresponding composition uniting the interactive objects "question" (Q) and "answer" (A) types. Such objects are used for saving of ontology concepts in structures the typical scheme of which is presented in Figure 4.

The content embedded in this framework corresponds to the following rules of grammar $\boldsymbol{GR^{QA}}$:

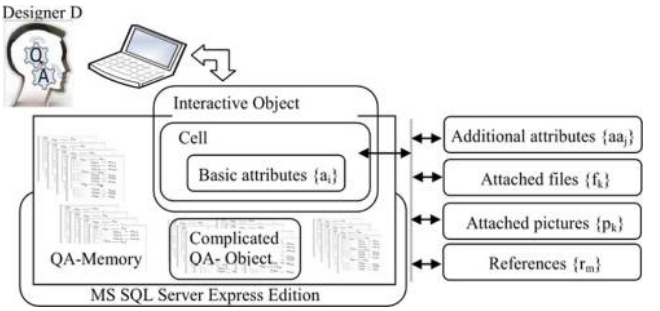


**Figure 3.**
Specification of the interactive object

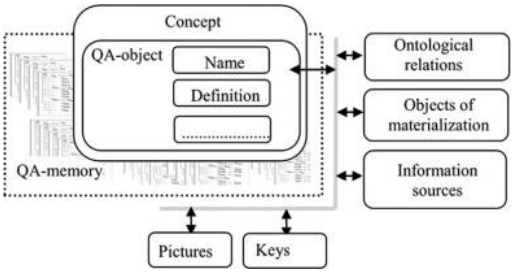


**Figure 4.**
Framework for specifying the concept

$$\left.\begin{array}{l} concept = QA(*in\ QA - memory*) \\ concept = (concept\ name, definition, \{list\}); \\ definition = text; \\ list = \{(relation, \{name\})\}; \\ relation = is - a \,|\, part - of \,|\, association \,|\, synonimity \,| \\ \qquad |\, picture \,|\, key \,|\, materialization \,|\, reference. \end{array}\right\} \quad (3)$$

Concepts can be combined in groups as in frames of project ontologies so as to create sections in the kernel ontology, the content of which is invariant to the specificity of any project fulfilled by designers. Combining is described by the following set of rules:

$$\left.\begin{array}{l} GN = (N, \text{“}\cap\text{”}, \{N\});\ (*\ group\ of\ concepts*, \text{“}\cap\ \ -\ \ operation\ of\ uniting\text{”}) \\ PO = \{GN\};\ (*project\ ontology*) \\ ontology = (kernel\ ontology, \{PO\}). \end{array}\right\} \quad (4)$$

It is necessary to note that concepts of the project ontology are usually used in definite conditions and that demands to specify variants $\{\Delta N_s\}$ of the concept N into the ontology. This necessity is supported by the following rules:

$$\left.\begin{array}{l} \Delta N = (definition, description, condition);\ (*concept\ usage*) \\ description = \{attribute\ name\}; \\ condition = programmed\ formulae; \\ N = \left(\text{“}\int\text{”}, \{\Delta N\}\right); \left(*\text{“}\int\text{”} - operation\ of\ assembling*\right) \\ concept = N \,\Big|\, \left(\text{“}\int\text{”}, \{N\}\right);\ (*simple\ or\ complicated\ concept*) \end{array}\right\} \quad (5)$$

Rules also specify the structure of complicated concepts, the content of which is defined by a group of the other concepts.

## 5. Features of extracting the ontology units

### *5.1 Typical applications of the ontology*

As told above, concepts of any project ontology are used in definite conditions of their applications. Means of the toolkit WIQA support the use of the integrated ontology, its project ontologies and their components till concepts for the following basic aims:

- systematization of the experience base as a whole and its “projections” on any designed project that is implemented in the project organization;
- access to the necessary assets, the models of which are stored in the experience base;
- extraction of requirements, their specifications and evolution in processes of designing;
- creation of the controlled vocabulary and its application in the frame of the occupationally natural language used in the project organization; and
- conceptual experimentation with the project solutions created by designers in the real time.

*5.2 Testing the used lexics*

At the bottom level of actions for achieving the called aims, designers use analytical processing the used textual units. First of all, they extract usages of concepts {ΔN} from such units. The toolkit WIQA supports three versions for the extraction of {ΔN} from textual units. In all versions, the current state of the ontology is applied.

One of the sub-tasks to create an ontology is to give the answer to a question if it is necessary to include a particular word into the ontology. Only an expert in a specific subject domain can handle this task, but there are ways to help him and reduce the sample from which he is to choose. One way is to develop a set of tools to automatically form a preliminary sample of language units (domain terms, concepts, etc.), which we consider being ontology concept variants.

Thus, the purpose of the work is to develop a toolset to form a set of concepts to include into a domain ontology on the basis of a project documentation for automated systems in Russian. While developing the toolset, we also used some tools of computational linguistics, particularly the morphological analyzer. The toolset is being developed for the QA system WIQA.

We have a collection of the *design documentation texts* ({T}) as an input and a set of *ontology concept variants* ({VarC}) as an output, which is a list of domain terms in the form of words and word collocations.

Project documentation uses a specific set of vocabulary and is created while solving design problems, because of which the ontology and expands. Thus, an expansion unit of the ontology is *a solved project task* and forming the ontology is *a cyclical process* (Figure 1): at each new turn (when referring to a new project document), the data obtained at the previous stage are used.

To define the place of forming a set of concepts in the general mechanism of creating an ontology more clearly, we give a generalized diagram (Figure 5). Note that some of the tools developed at the first step (e.g. filtering) are also used at the stage of defining links.

We form a set of concepts in stages. We select word forms from the project document, and then, the word forms are normalized with the help of a morphological analyzer

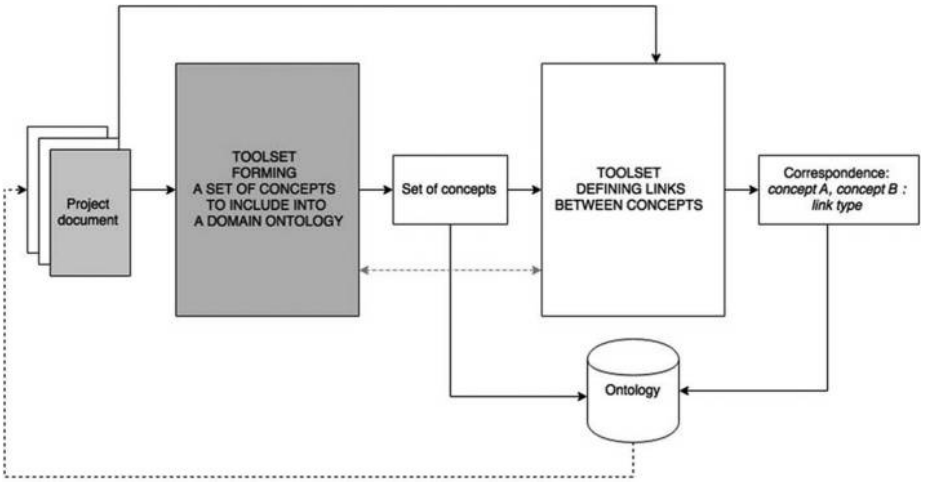


**Figure 5.**
Creating a domain ontology: generalized diagram

(parser); this is followed by automatic filtration of the given wordlist, selection of items to include into a vocabulary list and formation of collocations based on this list.

Here are the basic steps to forming a set of concepts *(the algorithm of actions)*:

- select all the word forms from the project document ({WF}) saving their order and punctuation;
- normalize a (bring to its original form) set of word forms with the help of a morphological analyzer (parser) and form a list of normalized language units of the project document ($\{WF_n\}$);
- filter the list with the help of a stopword dictionary (stopwords are words that do not carry the semantic load), a dictionary of acronyms and unknown symbols (e.g. for the Russian language such symbols are Latino ones) and form a filtered list of words ($\{WF_f\}$);
- remove duplicate words from the resulting list – get *the first part of the variant concept list* ({VarC});
- select the words (this work is done by an expert or a user) and include them into *a single-word concept list* $\{C_1\}$ (the user may resolve the cases of morphological ambiguity and correct errors made by the morphological analyzer if necessary);
- select *word collocation variants* from the project document which are two- or three-word long phrases ({VarCC}) taking into account the punctuation (the phrase cannot be divided by a comma or any other mark);
- filter the resulting list of collocation variant with the help of morphological models (described below) to get *a list of collocations* ({CC});
- filter the collocations $\{CC_f\}$ (remove duplicates and those which the ontology already includes) to get *the second part of the concept variant list* ({VarC});
- select the collocations (this work is done by an expert or a user) and include them into *a multi-word concept list* $\{C_2\}$; and
- form *a set of concepts to be included in the ontology* by creating *concept groups* based on the list $\{C_1\}$. The results obtained at this step provide good potential for detecting relations between the concepts.

Here is equation (6) of the algorithm that includes two main cycles:

$$\begin{cases} T \rightarrow WF \rightarrow WFn \rightarrow WFf \rightarrow VarC \rightarrow C1 \\ (T, C) \rightarrow VarCC \rightarrow CC \rightarrow CCf \rightarrow VarC \rightarrow C2 \end{cases} \tag{6}$$

Figure 6 opens the structure of these cycles presented them at the level of basic components. As this structure shows, presented task is easily divided into several sub-tasks that have clear inputs and outputs and can be processed independently. In addition to that, there are two cycles of processing data within our task. Thus, some sub-tasks can be processed in parallel.

So the best way to organize a given toolset is to use program agents interacted inside of common environment. The designed set of tools used following program agents:

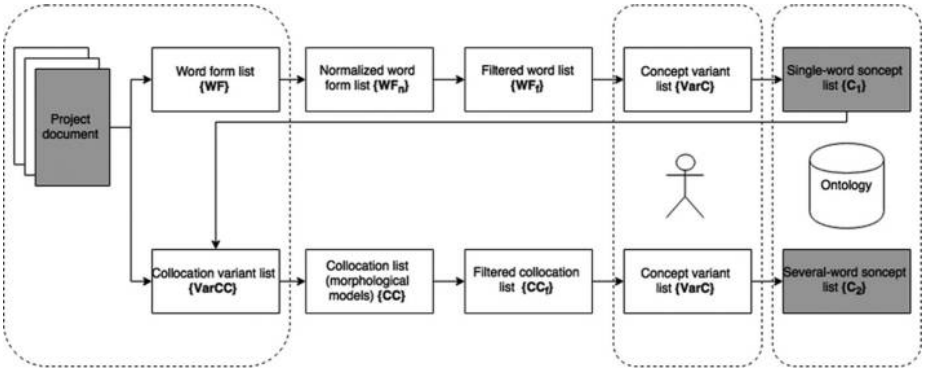


**Figure 6.**
Steps of forming a set of concepts

- *Agent A:* Splitter. It splits text in word forms.
- *Agent B:* Normalizer. It brings words to the original form with the help of a morphological analyzer (e.g. the word "компьютеров" to the word "компьютер").
- *Agent C:* Filter. It filters the list of words, removing stopwords and words with unknown symbols (e.g. the words are written in the Latin alphabet), as well as abbreviations (works with a dictionary of abbreviations).
- *Agent D:* It excludes repeating units from the list.
- *AgentE:* It works with a set of words derived from the ontology and, if necessary, filters the list taking into account the words which the ontology already contains.
- *Agent F:* It selects a word collocation (2-3 words in length) taking into account the list of concepts, the punctuation marks (collocation cannot be separated comma or another mark).
- *Agent G:* It filters the collocations on the basis of morphological models.

A number of these agents use the *morphological analyzer* inside (it can also be called a program agent) – an external program that allows one to determine the part of speech of a word and its grammatical characteristics (gender, number, case, etc.), as well as to bring it to its original (normal) form. The collaborative work of agents is presented in Figure 7, where the used dictionaries are also shown.

We carried out a comparative analysis of four morphological analyzers: LanguageTool (2014), Yandex Mystem (2014), Link Grammar Sosnin (2015) and one designed for WIQA (it uses the AOT morphology LanguageTool (2014) inside).

The analysis revealed that the most accurate is the MyStem analyzer, but its main drawback is its close code, so we cannot implement it into our work. Link Grammar Analyzer has the highest number of additional opportunities that may be useful for building relations between the concepts, but it works smoothly only in the Linux operating system. Language Tool Analyzer has its advantages, but it is written in Java, and thus, it is not possible to integrate it with the WIQA system. Taking into account all this, we will use the AOT morphology in the developed complex of utilities.

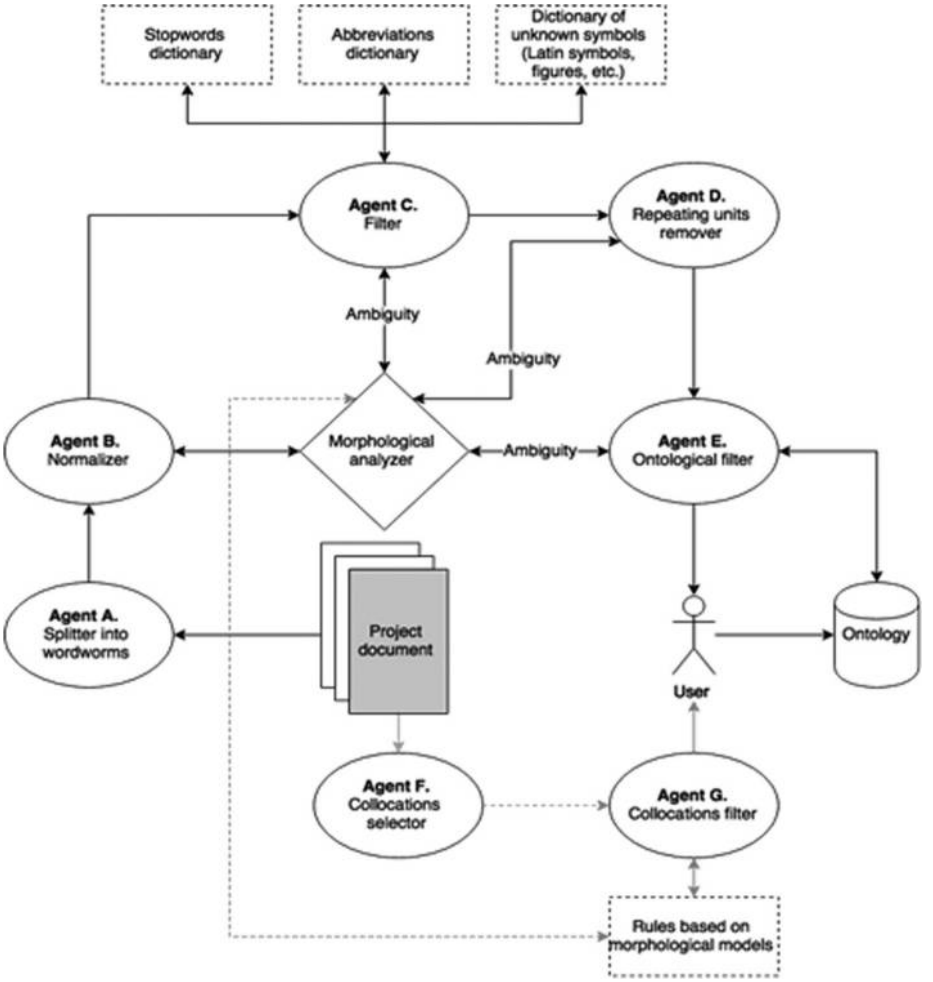


**Figure 7.**
Interaction between the program agents

*5.3 Graphical checking of predicative relation among concept usages*

The choice of words and phrases on their use in evolving the ontological units requires checking these text components from the viewpoint of their consistency to the reality of designing. In the described study, the tested consistency is a "predicativity" that opens the possibility for the check of variants for the use of concepts in sentences of the investigated text.

Moreover, in checking any chosen sentence, the designer can apply a "figuratively semantic scheme" (FS-scheme) of this sentence. Interactions of designers with the FS-schemes include a visual perception of the designers in addition to the used conceptual actions. This supplement helps in the constructive actions with understanding in the work with the use of concepts in textual units.

An instrumental complex, that provides such translation and other transformations of the FS-scheme, is presented in Figure 8.

The complex is implemented as an application in the environment of the toolkit WIQA. This complex uses the WIQA as the kernel that evolves by some means oriented on the figuratively semantic support. Only these means are obviously indicated in Figure 8, and they include:

- linguistic processor that provides automated translation of the investigated textual unit in its prolog-like description;
- converter that transforms such description in an initial version of the FS-scheme; and
- graphical editor that supports some transformations of the created FS-scheme from its initial state till the version that is needed for the designer.

The first transformation (by linguistic processor) helps to translate the textual units $T_i$ in the list of simple clause $T^*_i(\{C_j\})$, any of which has one of following structures:

$$\begin{aligned} C_j &= P_j(Ob_q), AC_j \\ C_k &= P_j(Ob_r, Ob_s), AC_k \end{aligned} \tag{7}$$

where P – predicate of the clause, Ob – name of the definite object and AC – associative component that is located out of the predicate form, but it is used in the transformed sentence of the text for the necessary utilization of the described meaning.

The second transformation fulfilled by the designer with the help of the convertor allows creating the initial state of the corresponding block-and-line FS-scheme. In this work, the description $T^*_i(\{C_j\})$ is processed as a program of the declarative type. Such understanding is inherited from the logical programming that uses prolog-like operators.

The third transformation has some varieties:

- translation in the pseudo-code program that helps to draw the FS-scheme;
- corrections of the FS-scheme on the base of information from associative components; and
- enriching the components of the FS-scheme and relations among them by information that is registered in the current state of the ontology corresponding to the executed project.

The translation to pseudo-code form opens the work with this form for appropriate elaborations, for example, aimed at the choice of the sequence of drawing the

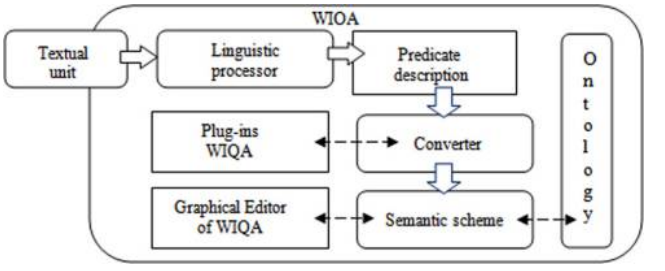


**Figure 8.**
The complex of means for transformations

components of the FS-scheme and relations among them on the chosen field of the editor or for programming the simulation in visual time (time of modeling).

The use of associative component suggests their graphical inclusion in the built FS-scheme for clarifying the graphical expression of the text meaning.

Interactions with the ontology elaborate on additional information that was hidden in the text, but it is presented in the ontology as knowledge about the subject area of the project. On the corrected FS-scheme, this information is expressed by additional indexed labels and definite color for the lines indicated the relations among components.

Below, an operational space for fulfilling the number of transformations will be demonstrated on the example of the following textual unit:

> *To define the form of a surface of liquid in a vessel sliding without friction on an inclined plane.*

For this text, its prolog-like version has the following expression:

$$\begin{array}{l} \textit{Slides}\ (\textit{a vessel}, \textit{liquid}; \textit{an inclined plane}),\ \textit{without friction}. \\ \textit{To define}\ (\textit{the form}, \textit{liquid}, \textit{a surface}). \end{array} \tag{8}$$

This description will be stored in the semantic memory of the toolkit WIQA, and it will be visually accessible to the designer in interface environment presented in Figure 9, where the designer has the possibility of correcting the description.

After that, the prolog-like description will be interpreted as a declarative program that is written in an extension of the pseudo-code language $L^{WIQA}$. Operators of this

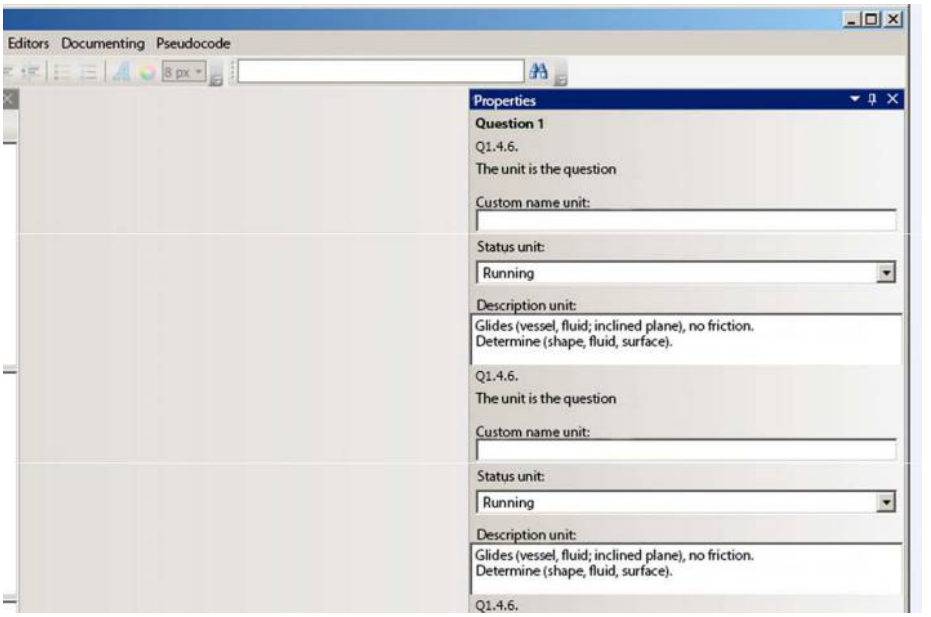


**Figure 9.** Visualizing the prolog-like description

extension have syntax that corresponds to expressions (1). The converter provides the first step of translating the declarative program to the FS-scheme.

After that, the prolog-like description will be interpreted as a declarative program that is written in an extension of the pseudo-code language $L^{WIQA}$. Operators of this extension have syntax that corresponds to expressions (1). The converter provides the first step of translating the declarative program to the FS-scheme.

For this, the converter fulfills following actions:

- Processing and analysis of the chosen textual unit for the subsequent visualization.
- Forming the FS-scheme of figurative representation of a situation given the chosen palette. There is an opportunity to visualize the FS-scheme in different palettes at the choice of the designer.
- Automatic accommodation of the FS-schemes. During translation, the converter places elements of the FS-scheme in the field of viewing for the effective perception of a situation by the designer. There are some variants of placing the elements in the field of the editor: Repulsion Layouter (uses a physical model of springs – an attraction connected and pushing away of untied figures), Spiral Layouter (a helicoid trajectory with priorities), as there is an opportunity for easy expansion of mechanisms of arrangement.
- Viewing the generated initial state of the FS-schemes. Rich opportunities of the developed editor (such as scaling, applications of various patterns, dynamic switching between palettes, scrolling, etc.) are accessible to the designer.

For a description (8), its FS-scheme is presented by the screenshot in Figure 10.

At the current state of our study, the designer takes into account associative components in a manual mode using commands of the editor. Later, this work will be automated. These components are placed in special blocks by indicating the FS-scheme components on which they influence.

In the field of the graphical editor, the FS-scheme is placed with using helical trajectory with priorities. Values of priorities (by way of their increase) specify a sequence of appointments to a place.

As mentioned above, after creating the initial state of the FS-scheme, the designer can translate it in imperative pseudo-code program, an execution of which simulate drawing the FS-scheme. Changing this program at levels of data and/or operators, the designer can achieve more suitable view for the created FS-scheme.

All presented transformations are developed and tested by authors. Now, our study is aimed at the development of means that should support enriching the FS-scheme by information from the ontology.

It is necessary to note that described semantic support for taking into account the predictivity in the work with the use of concepts is realized for the toolkit OwnWIQA intended for the personal activity of designers.

One more version of the graphical support is based on the pseudo-physics model of the sentence, words of which are interpreted as objects that take part in the "force interaction" that is visualized on the monitor screen. Formal expressions of pseudo-physics laws are similar to appropriate laws of the classic physics. In the stable

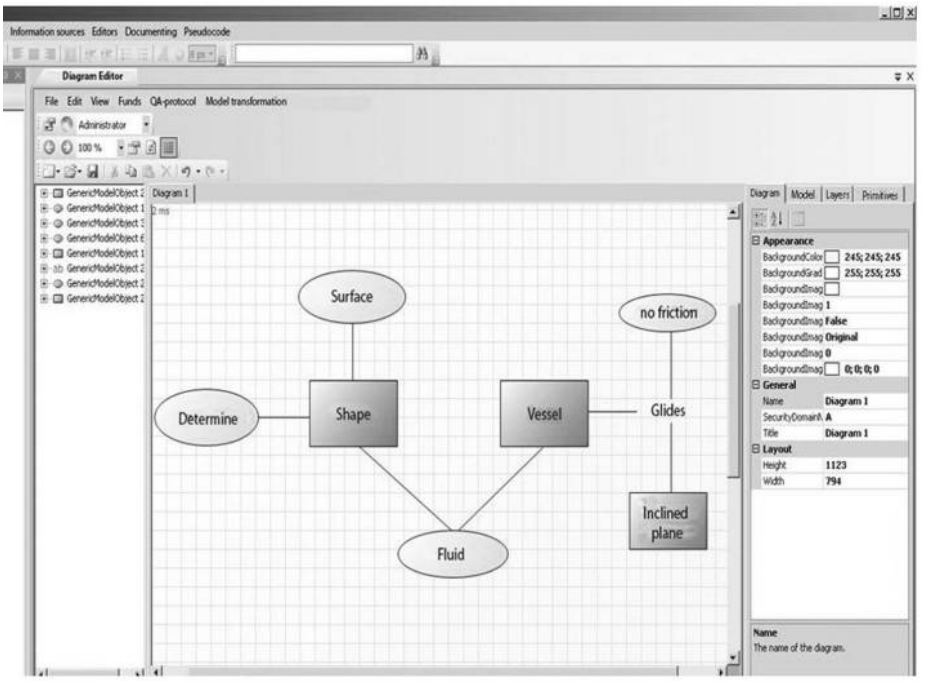


**Figure 10.**
Operational space of the graphical editor

state, (Figure 11) after finishing dynamic process on the screen, each group of words-objects will present the extracted simple sentence.

The screenshot in Figure 11 is used with labels for the generalized demonstrations of the visual forms and objects with which the designers are working. The language of this screenshot is Russian.

Let us notice that in the appointment of attributes (values of mass $m_i$, charge $q_i$ and others values and parameters), two mechanisms are applied – the automatic morphological analysis and the automated tuning of object parameters. Values are appointed by the type of part of speech. The suitable, normative values were chosen experimentally.

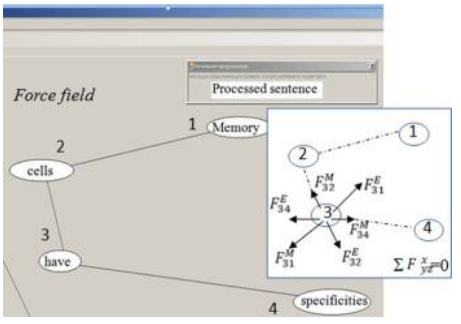


**Figure 11.**
Extraction of simple sentences

Extractions of simple sentences are aimed at their transformation in prolog-like forms to assemble logic formulae that correspond to the checked texts. This way is used not only for checking on conformity to the ontology vocabulary but also for controlled creation of logical conditions in models of precedents and the creation of ontology axioms.

It is necessary to note that the concept scheme presented in Figure 4 and details of the concept reflected in rules (5) indicate that presentations of such units in the project ontology have forms of precedents' models. For example, in the general case, the concept has QA-model presented in Figure 12.

Such presentation of the concept opens the possibility of using the means of pseudo-code programming embedded to the toolkit WIQA (Sosnin, 2013). Also, it opens the possibility for:

- controlling the used concepts;
- understanding its estimation;
- experiment conducting;
- conceptual modeling;
- conceptually algorithmic modeling;
- accumulating and improving experience;
- documenting;
- systematizing of experience; and
- programmable access to the ontology.

## 6. Main features of the proposed ontology

As told above, the proposed project ontology is oriented on the experiential way-of-working when designers solve tasks by using the conceptual experimenting. Such way-of-working is based on real-time interactions of designers with the experience base of generalized scheme, which is presented in Figure 13.

The experience base is intended for assembling the assets of a design company that create the family of SISs. Any asset is presented in this repository as the model of the precedent that presents the inclusion of the corresponding asset in designing. The definite model can be realized by the full scheme (Figure 1) or the appropriate form of its projection.

In the current state, the experience base is divided into section, each of which includes assets of the definite type. The greater part of assets is stored in the corresponding section in forms of their models. For example, the section of "Human resources" saves

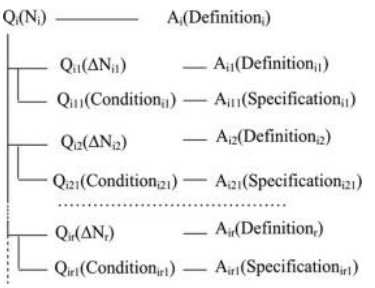


**Figure 12.**
Question-answer model of concept

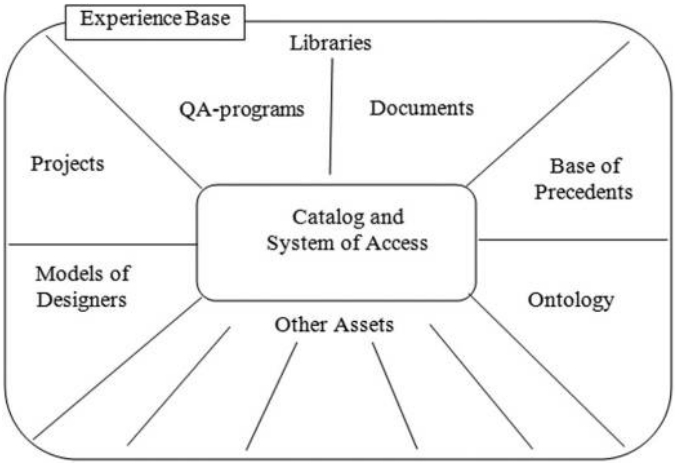


**Figure 13.**
Structure of the experience base

personified models of designers (PMDs). Section "Projects" is intended for registered information about developed projects of SISs.

There are two versions of the access to models of experience, the first of which is provided by the catalog of the experience base. The second version uses the search by keys that are included in the set of the controlled vocabulary.

Only one part of assets is placed in precedent base. The greater part of assets is stored in corresponding libraries, where they are presented in forms of precedent projections. As mentioned above, components stored in QA-memory can be bound with the attached files that are placed in corresponding libraries too. One of such libraries includes program written in C# that can be used in pseudo-code programming.

## 7. Ontological support in designing of configured templates

Offered means have been tested in the development of some projects, the last of which is a system for designing of tools. In the aircraft industry, for manufacturing the parts of a fuselage, wings and aileron, including details of their covering, technological templates are widely used. Any template for the part is a kind of its machining attachment that supports definite technological operations, for example, manufacturing of the corresponding part. Described means of the ontological support was applied for designing of the template tooling that usually consists of tens of thousands of templates. The built ontology consists of the following sections:

- the system of typical templates as concepts;
- the visualized classification of templates;
- templates in manufacturing of aviation parts;
- templates in a production control of parts; and
- templates as models of precedents.

In the section of concepts, the model of the template is described by its type and attributes, some of which are expressed through relations with other concepts. Items in this section are interactively accessible.

The used systematization of concepts and their classifier are shown in Figure 14, where relations are demonstrated on the example of one of the template. The classifier combines part-of-relations that facilitate the search for appropriate templates.

In the built ontology, any template is defined in the form of the precedent model that has the structure as shown in Figure 15. As all interfaces are in Russian, some fields are marked with labels.

The specificity of the program components used in the precedent model should be noted. The life cycle of template manufacturing includes the stages that require the creative-intensive activity of workers that participate in designing of the templates. It is caused by the need to take account of future ways to move the laser beam on all lines that represent the template for its steel billet. Therefore, the template model includes two algorithmic descriptions. The first description imitates the movement of the laser beam in tasks of searching the effective trajectories. The second description is a result of computer numerical control programming that provides the process of laser cutting.

## 8. Conclusion

This paper presents the system of means for the creation and use of the projects ontology in the development of the family of SISs when enormous quantity of project tasks is being solved by the team of designers in the corporate instrumental network.

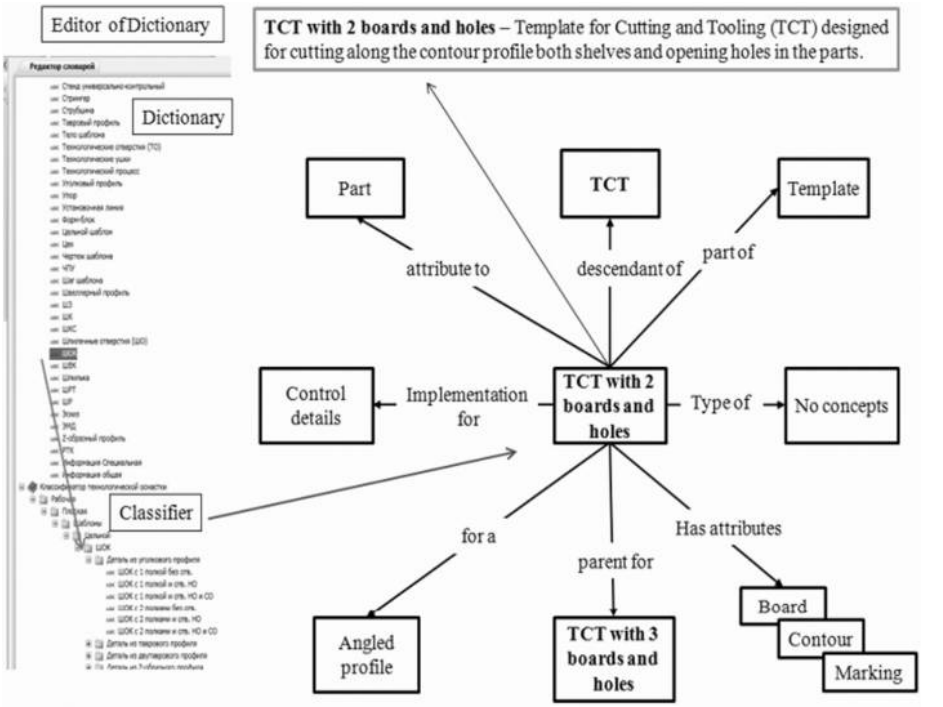


**Figure 14.**
System relations in the ontology

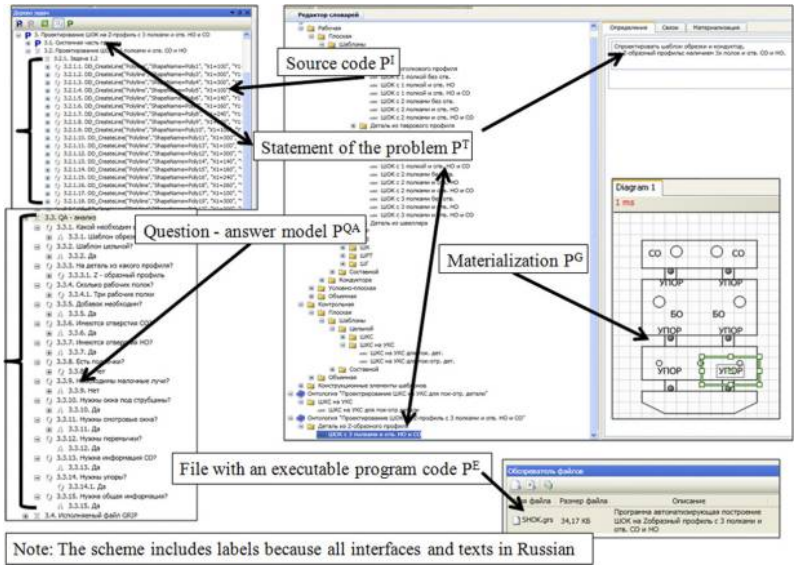


**Figure 15.**
Structure of the template definition

The success of such activity essentially depends on a mutual understanding of designers and means supported conceptual decisions of designers at early stages of their work.

The proposed approach is based on the experiential way-of-working of the designers who interact with the experience base, accumulating the assets of the design company. All assets are presented in this base as models of precedents that have been built by the normative framework. In the real-time work, the designers use the toolkit WIQA intended for the solution of tasks at the conceptual stage of collaborative designing. The toolkit provides the use of QA reasoning in conceptual experimenting in solutions of the project task. It also helps to create models of precedents and to achieve positive ontological effects described above.

## References

Bullinger, H.J., Warschat, J., Schumacher, O., Slama, A. and Ohlhausen, P. (2005), "Ontology-based project management for acceleration of innovation project", *Lecture Notes in Computer Science*, Vol. 3379, pp. 280-288.

Chaos Manifesto (2013), available at: www.versionone.com/assets/img/files/ChaosManifesto2013.pdf

Eden, A.H. and Turner, R. (2007), "Problems in the ontology of computer programs", *Applied Ontology*, Vol. 2 No. 1, pp. 13-36.

Fitsilis, P., Gerogiannis, V. and Anthopoulos, L. (2014), "Ontologies for software project management: a review", *Journal of Software Engineering and Applications*, Vol. 7 No. 13, pp. 1096-1110.

Garcia, A.C.B., Kunz, J., Ekstrom, M. and Kiviniemi, A. (2003), "Building a project ontology with extreme collaboration and virtual design & construction", CIFE Technical Report # 152, Stanford University, Stanford.

Guarino, N. (1998), *Formal Ontology and Information Systems in Proceedings of FOIS'98*, IOS Press, Trento, Italy, Amsterdam, pp. 3-15.

Guarino, N., Oberle, D. and Staab, S. (2009), "What is an ontology?", in Staab, S. and Studer, R. (Eds), *Handbook on Ontologies, Second Edition: International Handbooks on Information Systems*, Springer Verlag, Heidelberg, pp. 1-17.

Henninger, S. (2003), "Tool support for experience-based software development methodologies", *Advances in Computers*, Vol. 59, pp. 29-82.

Jacobson, I., Ng, P.W., McMahon, P., Spence, I. and Lidman, S. (2012), "The essence of software engineering: the SEMAT kernel", *Queue*, Vol. 10 No. 10.

LanguageTool (2014), "Программа для проверки грамматики и стиля [Электронный ресурс]", available at: https://languagetool.org/ru/

Martins, A.F. and De Almeida, F.R. (2008), "Models for representing task ontologies", *Proceeding of the 3rd Workshops on Ontologies and their Application, Brazil*.

Sosnin, P. (2013), "Scientifically experimental way-of-working in conceptual designing of software intensive systems", *Proceedings of the IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT)*, *Budapest*, pp. 43-51.

Sosnin, P. (2015), *An Ontological Support for Interactions with Experience in Designing the Software Intensive Systems, LNCS 9158*, Springer, Heidelberg, pp. 387-400.

Yandex Mystem (2014), "[Электронный ресурс]", available at: https://tech.yandex.ru/mystem/

## Further reading

Automated processing of text (AOT) (2014), available at: http://aot.ru/

Byrne, A. (2005), "Consciousness and nonconceptual content", *Philosophical Studies*, Vol. 113 No. 3, pp. 261-274.

Pacherie, E. (2000), "Levels of perceptual content", *Philsophical Studies*, Vol. 100 No. 3, pp. 237-254.

Temperley, D., Lafferty, J. and Sleator, D. (2014), "Link grammar parser", available at: www.abisource.com/projects/link-grammar/

## Corresponding author

Petr Sosnin can be contacted at: sosnin@ulstu.ru