# The concept of a software complex for interdisciplinary problems solving based on self-organization principle

Alexander Berman[1][0000-0001-8339-7338], Olga Nikolaychuk[1][0000-0002-5186-0073], Alexander Pavlov[1][0000-0002-7753-7514]

[1] Matrosov Institute for System Dynamics and Control Theory
of Siberian Branch of Russian Academy of Sciences, Irkutsk, Lermontova, 134, Russia
idstu.irk.ru

**Abstract.** The paper presents the concept of a software complex for solving interdisciplinary problems based on self-organization features. In particular, the basic principles and stages of self-organization process during solving an interdisciplinary problems of designing complex technical systems, as well as the architecture of proposed software along with some details of implementation are considered. The architecture of the software complex includes subject and problem ontologies, data and knowledge bases, set of "solvers" as well as intelligent scheduler. The intelligent scheduler implements the self-organization algorithm and provides creation the computing environment for solving considered problem using "solvers" as a buildings blocks. The implementation of self-organizing algorithm base on combination of knowledge representation as an ontologies, group decision-making, component and model-oriented approaches. Self-organization features in context of designing complex technical systems task are implemented on the stages of defining the design methodology, determining the source data, solving the problem, and training the system. The intelligent scheduler can analyze the state of current task with set of indicators and manage it through a set of local rules. This paper presents examples of local rules for each stage. The stages specifications in the form of technological diagrams that contains components used ("solvers"), their connections along with the results of their work describe the self-organization process features related to each stage. The software implementation are based on the capabilities of the used platform for creating knowledge-based systems developed by the authors and component approach applied to specialized component development.

**Keywords:** Interdisciplinarity, Complex Technical System, Design, Information Technology, Subject Ontology, Problem Ontology, Self-Organization, Local Rules Of Self-Organization.

## 1 Introduction

The task of designing complex technical systems is an interdisciplinary one, which requires processing huge amounts of data and knowledge of various scientific, technical and scientific-technical disciplines. At the stage of creating technical systems, there is always some uncertainty of strength, resource and structural reliability and safety due to imperfection and violation of methods and means of creating objects, the inability to adequately test elements and components of complex unique technical systems, which leads to their sudden failures. Due to incompleteness of some sets of data and knowledge belonging to different disciplines, the relationships between them can not be completely defined, which makes it difficult to formulate both disciplinary and interdisciplinary goals and may be one of the reasons of incorrectly interdisciplinary tasks formulation.

These problems determine the relevance of the task of creating the technology and the appropriate software complex ensuring the teamwork of a specialists (experts) from various disciplines for solving interdisciplinary problems related to the design of complex technical systems. The software complex should meet the following requirements:

- common space for information exchange including mathematical, methodological and software areas;
- interaction between specialists from the various disciplines;
- formulation of the one cooperative view point for the multiple of the expert groups;
- processing of heterogeneous information, along with data of unknown or uncertain at design time structure model;
- the possibility of using heterogeneous software (open source code, third-party programs, own programs, hybrid systems, etc.);
- the user requirements for skills in programming and knowledge representation languages are limited to interaction with the software complex interface in domain terms;
- decision support at all stages of problem solving;
- high level of the automation during implementation of the new problem solving methods and adaptation of existing ones to any changes in the initial data.
- Implementation of these requirements is based on the following approaches:
- common space for information exchange by ontological modeling,
- interaction between specialists from the various disciplines by the methods of group decision-making,
- use of heterogeneous information and software by the separation of data representation and processing methods according to model-oriented and component-based software development approaches,
- decision making support by applying knowledge processing methods (for example, expert systems),
- high level of the automation by applying self-organizing algorithms.

Currently, there are methods and models of artificial self-organization for the formation of coordinated solutions and control of complex objects [1, 2, 3]. As follows from works [3, 4], a system can be considered self-organizing if it acquires some spatial, temporal or functional structure without specific external influence. So the artificial self-organization can be represented as a process of the automatic modification (adaptation) of the action plan (decision-making algorithm) when changing the properties of the controlled object, the control goal, or environmental parameters are meet.

It is proposed to acquire the solution of the interdisciplinary problem as the schema of interaction between "solvers" of disciplinary and interdisciplinary problems of different competence and specialization by application of the self-organizing algorithm [3]. The mechanism of self-organization provides organization the available "solvers" into computational structure according to the parameters of the problem model and object to study like the object properties, influencing factors, etc.

The obtained computational structure can be automatically reorganized in case of any changes in initial states.

The purpose of this paper is to present the concept of a software for solving interdisciplinary problems of designing complex technical systems.

## 2 Architecture of the software complex

The conceptual architecture of a software complex for creating and supporting the self-organizing systems for solving interdisciplinary tasks is defined:

$$IS = \left( E, Dt, Knl, Ont, P, \hat{P}, Slv, Crd, R_{IS}, Ind, Pln, UI \right), \tag{1}$$

where *IS* – software complex, $\underline{E}$ – experts, the *Dt* – database, *Knl* – knowledge base, *Ont* – the ontology of domain and problem areas, $Ont \rightarrow \{P, \widehat{P}, Dt, Knl, R_{IS}\}$ *P* – tasks, $\hat{P}$– task hierarchy, *Slv* – "solvers", $R_{IS}$ – the relationships between components *IS*, in particular, between experts and tasks, between tasks, between tasks and "coordinators", between tasks and "solvers", *Crd* – "coordinators" of the task, *Ind* – state indicators IS, $Ind = \{ I^{DefP}, I^{SP}, I^{Un}, \ldots \}$ a set of indicators to display of the current state of task execution process, *Pln* – an intelligent scheduler that implements a self-organizing algorithm *SAlg* for solving an interdisciplinary task based on local rules *LRule*, *UI* a user interface.

The "solvers" of software complex can be divided to the basic ones and user ones $Slv = \{ Slv_{Base}, Slv_{User} \}$. The basic "solvers" provides the set of build-in data and knowledge processing operations [5]: data control – $Slv^{Dt}$, ontological modeling – $Slv^{Ont}$, rule-based reasoning – $Slv^{Knl}$, organization of two-way data exchange – $Slv^{Com}$, dialog interaction with the user – $\mathrm{Slv}^{\mathrm{UI}}$, specification of operations based on visual workflow notation – $Slv^{Op}$.

$Slv^{Ont}$ provides the creation of an ontological (conceptual) model for the selected domain with a specified level of detail.

$Slv^{Knl}$ provides creation of knowledge bases by visual construction of rules and formation of initial conditions using obtained domain ontology as initial data. In current implementation Drools system is used as rule-based reasoning engine so the code generation function results are in format of Drools knowledge representation language.

The *Slv*$^{Com}$ use WebSocket protocol to provide the ability to fast, asynchronous data exchange between users and/or "solvers" of the complex, as well as the server to client interaction with a server as initiator.

*Slv*$^{UI}$ provides automatic creation of user interface based on meta descriptions using graphical controls to display the data in a simple and/or nested tables, semantic networks and elements for selection one/multiple values from the list, and the standard set of elements for simple data types (one/many-lines input fields, the date, time, datetime controls, checkbox for Boolean type, etc.).

Slv$^{Op}$ has two main functions: the creation formal specification of actions plan for solving a certain domain problem using visual workflow notation, and the second is the implementation of the obtained specification using the other "solvers" of software complex and functionality of the available external systems.

*Pln* [3] provides an actions plan of the solving domain task process as the interaction schema between "solvers" that displays both the control flow and data flow according to model of the task. During the task execution process Pln monitors the state of the task and in case of necessity adjusts actions plan by results of applying a local rules to the current state according to the values of the indicators.

## 3 Stages of self-organization algorithm

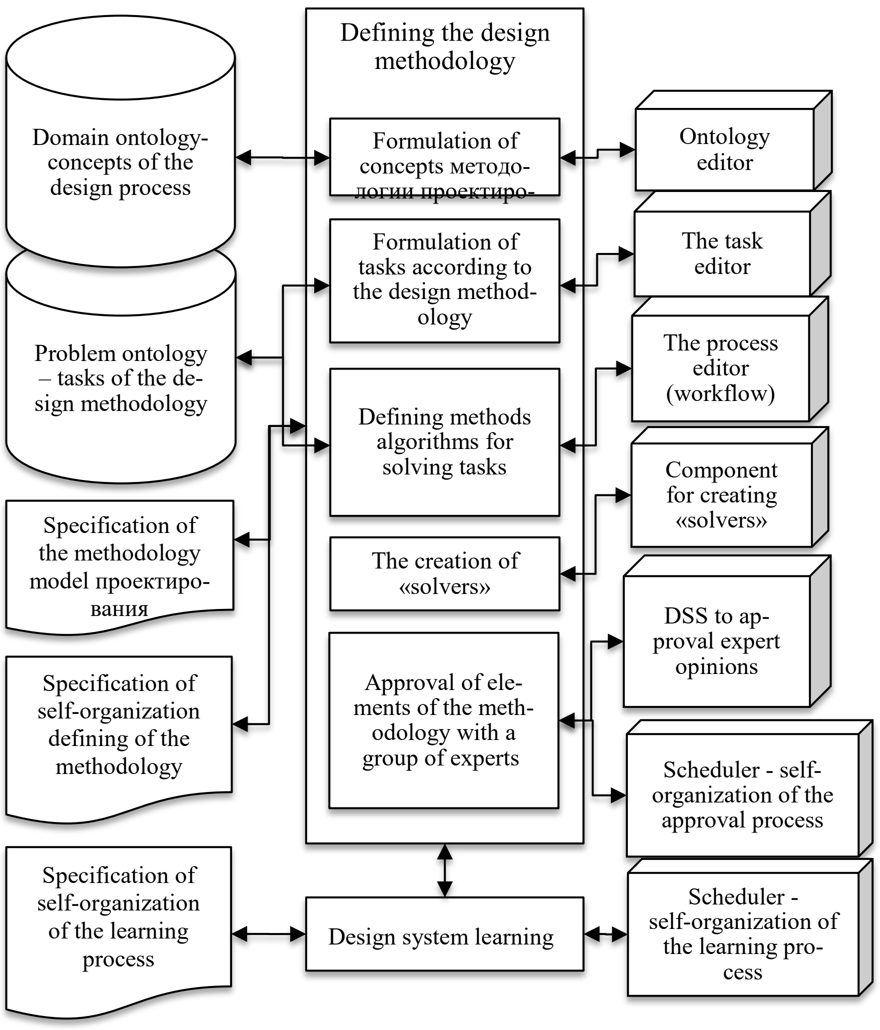
Let's consider the stages of the proposed algorithm for self-organization of the software for solving an interdisciplinary problems of designing technical systems. The algorithm has four main stages:

1. Self-organization at the stage of creating a software system (see Fig. 1).

   This stage implements the following:

- Definition of the design methodology. By defining the methodology, we mean describing the stages and tasks of the design process for an object or a certain class of objects. Design stages and tasks are formalized in the task ontology as a hierarchy of conceptual tasks [6-8], where the hierarchy shows the part-whole relationships. The actions plan of abstract kind is formed using visual workflow notation based on the obtained task hierarchy.
- Detailed formulation of tasks [8], according to the design methodology. At this stage, the conceptual tasks defined in the previous stage are being specified. Such type of specification consists of related domain concepts and rules along with descriptions of the methods available for considered tasks. The obtained specification would considered by the related group of experts (domain specialists) to retrieve the common view.
- Definition of methods for solving tasks. The algorithms of methods for solving tasks that been described in the task ontology [8] are defined using visual workflow notation. A mandatory part of this stage is the description of actions in the case of obtaining unsatisfactory result for one or several subtasks. Moreover, the alternative algorithms that based on other implementations or provides a different

accuracy, and so on can be added to the method description. The method definition is also have be adjusted to the common opinion by the group of experts.



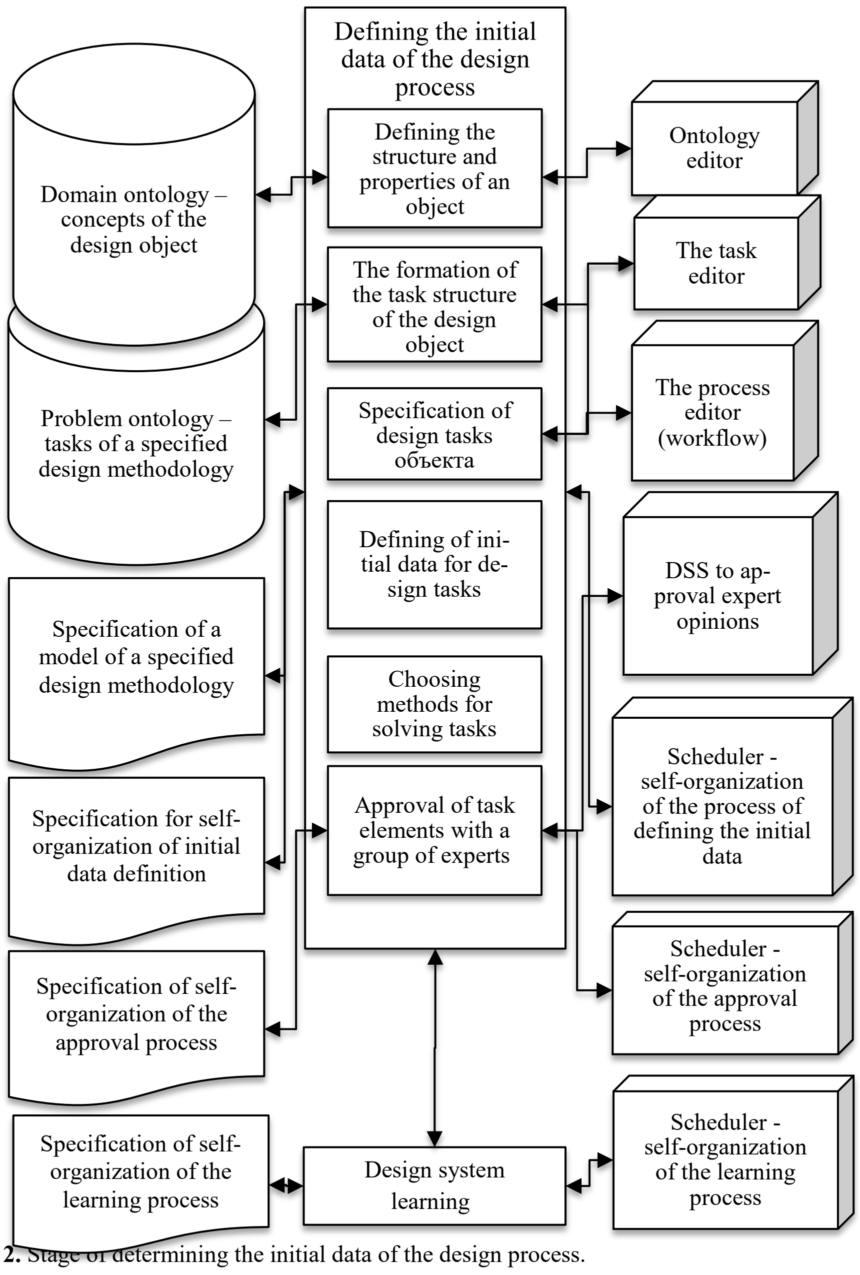
Defining the design methodology
Domain ontology-concepts of the design process
Formulation of concepts методологии проектиро-
Ontology editor
Formulation of tasks according to the design methodology
The task editor
Problem ontology – tasks of the design methodology
The process editor (workflow)
Defining methods algorithms for solving tasks
Component for creating «solvers»
Specification of the methodology model проектирования
The creation of «solvers»
DSS to approval expert opinions
Specification of self-organization defining of the methodology
Approval of elements of the methodology with a group of experts
Scheduler - self-organization of the approval process
Specification of self-organization of the learning process
Design system learning
Scheduler - self-organization of the learning process

**Fig. 1.** Stage of determining the design methodology.

– Creating "solvers" - software components that implement the proposed methods for solving tasks. This stage is important in case of the methods for processing weakly structured information based on artificial intelligence methods, such as expert systems [9, 10, 11].

Within the first stage, the model of the design methodology is formed while the self-organization algorithm is utilized for user support during the describing concepts, tasks, methods, “solvers” and adjusting of descriptions [12, 13, 14].

2. Self-organization at the stage of setting up a software system (see Fig. 2). This stage includes:



Defining the initial data of the design process
Defining the structure and properties of an object
The formation of the task structure of the design object
Specification of design tasks объекта
Defining of initial data for design tasks
Choosing methods for solving tasks
Approval of task elements with a group of experts
Domain ontology – concepts of the design object
Problem ontology – tasks of a specified design methodology
Specification of a model of a specified design methodology
Specification for self-organization of initial data definition
Specification of self-organization of the approval process
Specification of self-organization of the learning process
Design system learning
Ontology editor
The task editor
The process editor (workflow)
DSS to approval expert opinions
Scheduler - self-organization of the process of defining the initial data
Scheduler - self-organization of the approval process
Scheduler - self-organization of the learning process

**Fig. 2.** Stage of determining the initial data of the design process.

- Input of initial data, including a description of the structure and properties of the object under design.

- Defining the specific structure of tasks / subtasks according to the specified structure and properties of the object under design. Self-organization at this stage is the user support during analysis of the description of the object structure in the ontology and the formation of a sequence of tasks for each element of the hierarchy according to structure of the object under design and the description of the methodology.
- The obtained task structure are expanding by concrete data (instances of the domain ontology concepts) about the object under design. Also the correspondence between the domain concepts and the concepts from the task description is established.
- Determination the runnable methods of solving tasks and most effective ones from them based on the available initial data. The intelligent scheduler is used to check whether the initial data is available to evaluate whether the method can be launch, and whether the methods are runnable with the current input parameters. For example, with the domain ontology, knowledge bases for expert systems are formulated, methods are ranked by efficiency and accuracy, and the selection of the runnable methods is performed.

  As a result of this stage, a models of the specific tasks is formed for a given object.

3. Self-organization at the stage of exploitation of the software system (see Fig. 3). This stage includes:

- Adjustment of task statements based on the results of their executions and expert suggestions,
- Changing methods for solving tasks based on the results of the executions and expert suggestions: reducing performance requirements, addition a new method or a new algorithm for implementing the method.
- Adjusting the actions plan for task solving based on the results of the executions and expert suggestions.

The process is controlled with set of indicators: the progress indicator, risk indicator, the indicator formulation of the task etc. [3].

The result of this stage is the adjusted model for solving the specific task.

4. Self-organization during the learning phase of a software system:

- Learning of the system as a result of analysis, extraction and accumulation of information (rules, precedents) at each stage of the software system operation.
- Use of knowledge at all stages of self-organization to improve efficiency.

## 4 Knowledge structure of the scheduler

The main component of the software complex that implements self-organization algorithms is the intelligent scheduler. Self-organization is based on the analysis the states of set of indicators and the local rules.

The structure of knowledge base containing the local rules corresponds to the stages of self-organization, in particular, consists of following parts:

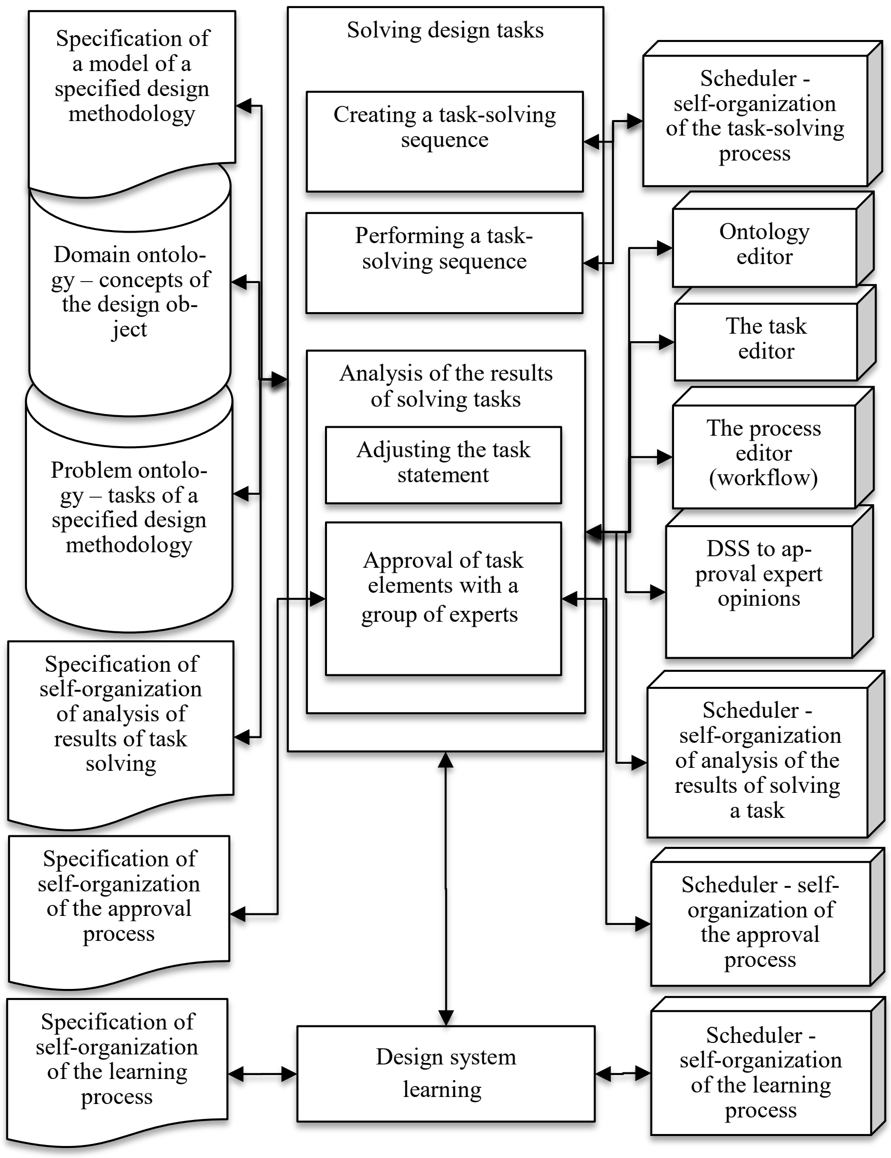

Specification of a model of a specified design methodology
Domain ontology – concepts of the design object
Problem ontology – tasks of a specified design methodology
Specification of self-organization of analysis of results of task solving
Specification of self-organization of the approval process
Specification of self-organization of the learning process
Solving design tasks
Creating a task-solving sequence
Performing a task-solving sequence
Analysis of the results of solving tasks
Adjusting the task statement
Approval of task elements with a group of experts
Design system learning
Scheduler - self-organization of the task-solving process
Ontology editor
The task editor
The process editor (workflow)
DSS to approval expert opinions
Scheduler - self-organization of analysis of the results of solving a task
Scheduler - self-organization of the approval process
Scheduler - self-organization of the learning process

**Fig. 3.** Stage of execution of the design process.

- self-organization in the process of determining the design methodology,
- self-organization in the process of defining the initial data,

- self-organization in the process of description the task solving algorithm,
- self-organization in the process of knowledge adjustment by expert and group of experts,
- self-organization in the process of the system learning.

Here are examples of local rules.
Rules for self-organization of the process of determining the design methodology:

```
IF the stages of the methodology are not defined, THEN
start the procedure for defining the stages;
IF the task of the methodology stage is not defined, THEN
start the task definition procedure;
IF the requirements for the object under design are not
defined, THEN launch the requirements definition proce-
dure;
and others.
```

Rules for self-organizing the initial data definition process:

```
IF the initial data is not defined, THEN start the ini-
tial data definition procedure;
IF the result of the procedure for determining the ini-
tial data is not feasible, THEN start the procedure for
changing the model of the methodology;
and others.
```

Rules for self-organization of the problem solving process [3,15].

```
IF the formulation of the methodology tasks is complete
AND the definition of the initial data is complete, THEN
the procedure for solving the design problem is started;
IF the result of solving the task is unsatisfactory (for
example, it does not meet the risk criteria), THEN launch
a procedure to identify the reasons for unsatisfactory
solution of the task;
and others.
```

Rules for self-organization of the knowledge adjustment process [12]:

```
IF there is an incompleteness of the task formulation,
THEN launch a procedure to reduce (resolve) the identi-
fied incompleteness, depending on the type of incomplete
information;
IF there is an incompleteness of the source data of the
task, THEN launch the procedure to agree on the opinions
of experts on the new source data;
IF there is an incompleteness of knowledge (templates,
rules, precedents and etc.) of the task, THEN launch a
```

```
procedure to agree on the opinions of experts on new
knowledge (templates, rules, precedents and etc.) of the
problem;
IF there is an incompleteness of data on the methods of
the problem, THEN launch a procedure to agree on the
opinions of experts on new methods of the task;
and others.
```

Rules for self-organization of the learning process:

```
IF a new task formulation is being created, THEN the pro-
cedure for collecting expert opinions on the reasons for
creating a new task formulation is launched AND the case-
based knowledge base of task formulations is updated;
IF you are forming a new sequence of methods of solving
the task, THEN start the procedure of collecting expert
opinions on the reasons for the creation of a new algo-
rithm for solving the task AND funding the case-
knowledge-base algorithms (methods) of solving the task;
and others.
```

## 5 Features of software implementation

The software implementation of the software complex is proposed to be performed using the capabilities of the platform for creating knowledge-based systems [5, 11].

This software platform is developed as a web application using the "thin" client technology, in which the components [16, 17] ("solvers") of the complex are placed on the server, and a standard browser program acts as a user terminal.

In the process of organizing interaction between the software complex components along with internal client-server links of them for calling methods and receiving (sending) data, it is suggested to limit the use of the following set of protocols: HTTP, SOAP, and WebSocket. This set provides the ability to use a significant part of existing software systems and libraries.

To implement the components user interface, are used a well-established approach for creating interactive web pages using HTML, CSS, JavaScript, and popular libraries (jQuery, jQueryUI, jQueryGrid, and jsPlumb). The data exchange process is organized both on the basis of client-initiated requests over the HTTPS Protocol, and using bidirectional channels of the WebSocket Protocol. Currently, the server part includes two HTTP servers: Apache-based and Node.js, which are together with the WSS Node.js server form the external part of the server.

The HTTP-Apache server provides access to the functionality of the basic "solvers": $Slv^{Dt}$, $Slv^{Ont}$, and partially to the functionality of $Slv^{Kln}$ (knowledge base design and code generation). HTTP Node server.js provides a solution to the problems of authentication of the customer $Slv^{Com}$ and WSS server Node.js provides Slavcom with the ability to organize bidirectional data exchange, in which each client is able to interact with others in any order.

## 6 Conclusion

The article discusses the basic approaches, conceptual stages of self-organization of the process of solving an interdisciplinary task of designing complex technical systems, as well as the architecture of the software complex and features of its implementation. The proposed approach include a combination of ontological knowledge representation, group decision-making, component and model-oriented approaches that ensure the implementation of a self-organizing algorithm. The stages of self-organization include the stage of defining the design methodology, determining the initial data, description the task solving algorithm, and the system learning. Self-organization of the system is based on analyzing the state of set of indicators through a system of local rules. The chosen approach to software implementation is based on the use of modern tools and data exchange protocols and provides the tool under development with the ability to add a new one, including by integrating software from third parties, as well as adapting existing functionality in accordance with changes in the requirements of the domain.

In the future, it is planned to implement proposed software complex and utilize it to solve the tasks of designing unique mechanical and technical systems operated at hazardous industrial facilities.

## 7 Acknowledgments

The research was supported by the Program of the Fundamental Research of the Siberian Branch of the Russian Academy of Sciences, project no. IV.38.1.2 (reg. no. AAAA-A17-117032210079-1). Results are achieved using the Centre of collective usage «Integrated information network of Irkutsk scientific educational complex».

## References

1. Kaljaev, I.A., Kaljaev, A.I., Korovin, Ja.S.: Principles of organization and functioning of deserted robotic production. Mechatronics, automation, control, vol. 17, 11, 741–749 (2016).
2. Kolesnikov, A.V., Kirikov, I.A., Listopad, S.V.: Hybrid Intelligent Systems with Self-Organization: Coordination, Consistency, Dispute, 189 p. Moscow, IPI RAN Publ (2014).
3. Berman, A.F., Nikolajchuk, O.A., Pavlov, A.I.: Self-organizing solution formation algorithm to ensure the required technical condition of complex hazardous objects. In: Proceedings of the conference "System Analysis and Information Technologies", vol.1, pp. 377-384. Moscow, ISA RAN Publ (2017).
4. Haken, G.: Information and self-organization: a macroscopic approach to complex systems, Moscow, URSS: LENAND Publ. (2014).

5. Nikolaychuk, O.A., Pavlov, A.I., Stolbov, A.B.: The software platform architecture for the component-oriented development of knowledge-based systems. In: Proceedings of the 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1234-1239, Opatija, Croatia (2018).
6. Gavrilova, T.A., Kudrjavcev, D.V., Muromcev, D.I. Knowledge Engineering. Models and methods. Spb., «Lan'» Publ (2016).
7. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: Ontology of Tasks and Methods. IEEE Intelligent Systems, 14(1), 20-26 (1998).
8. Berman, A.F., Nikolaychuk, O.A., Pavlov, A.I.: The Ontology Model for Automating the Solution of Multidisciplinary Research Tasks. In: Proc. the V Intern. Workshop Critical Infrastructures: Contingency Management, Intelligent, Agent-Based, Cloud Computing and Cyber Security (IWCI 2018), vol. 158. pp. 1-6 (2018).
9. Cretu, L. G., Florin, D.: Model-Driven Engineering of Information Systems: Principles, Techniques, and Practice. Apple Academic Press (2014).
10. Yurin, A.Yu., Dorodnykh, N.O., Nikolaychuk, O.A., Grishenko, M.A.: Designing rule-based expert systems with the aid of the model-driven development approach. Expert Systems, vol. 35, 5 (2018).
11. Dorodnykh, N.O., Yurin, A.Yu.: Technology for creating rule-based expert systems using the model transformations. SB RAS, Novosibirsk (2019).
12. Gubanov, D., Korgin, N., Novikov, D., Raikov, A.: E-Expertise: Modern Collective Intelligence. Series: Studies in Computational Intelligence, vol. 558, pp. 112. Springer International Publishing (2014).
13. Berman, A.F., Nikolajchuk, O.A., Pavlov, A.I.: Method of acquiring multidisciplinary knowledge based on ontology. In: Proceedings of the conference "System Analysis and Information Technologies", vol. 1, pp. 295-302. Moscow, ISA RAN Publ (2017).
14. Berman, A.A., Nikolaychuk, O.A., Maltigueva, G.S., Yurin, A.Yu.: A Method of Experts' Knowledge Approval for Industrial Safety Expertise Task. In: Series:Advances in Intelligent Systems Research. Proceedings of the VIth International Workshop «Critical Infrastructures: Contingency Management, Intelligent, Agent-Based, Cloud Computing and Cyber Security» (IWCI 2019), vol. 169, pp. 102-107 (2019).
15. Mahutov, N.A., Berman, A.F., Nikolajchuk, O.A.: Some principles of self-organization to manage the risk of man-made disasters. Risk analysis problems, vol. 12, 4, 34–45 (2015).
16. Nikolajchuk, O.A., Pavlov, A.I.: Using the component approach to create a research automation system. Vestnik of Computer and Information Technologies, 4 (70), 23-32 (2010).
17. George, T. Heineman, William, T.: Councill. Component-Based Software Engineering: Putting the Pieces Together. Addison-Wesley Professional, Reading (2001).