Hindawi Publishing Corporation
Advances in Human-Computer Interaction
Volume 2009, Article ID 619405, 18 pages
doi:10.1155/2009/619405

Research Article

Means of Question-Answer Interaction for Collaborative Development Activity

Petr Sosnin

Department of Computer Engineering, Faculty of Information Systems and Technologies, Ulyanovsk State Technical University, Severny Venetc 32, 432027 Ulyanovsk, Russia

Correspondence should be addressed to Petr Sosnin, sosnin@ulstu.ru

Received 27 May 2008; Revised 23 October 2008; Accepted 14 January 2009

Recommended by Guadalupe Muñoz

The key problem of successful developing of the software intensive system (SIS) is adequate conceptual interactions of stakeholders at the early stages of designing. Nowadays the success of development is extremely low. It can be increased with using artificial intelligence (AI) means including models of reasoning supported by the human-computer interaction in collaborative development activity. In this paper, a number of question-answer means for modeling reasoning are suggested. Such kind of means is defined and implemented in order to get effects of integrating the collective reasoning for their positive influence on the intellectual activity of designers. Question-answer means are arranged as a specialized processor opening the possibility to question-answer programming of the tasks on the conceptual stage of designing. Suggested and investigated means can be used for solving any complicated task.

1. Introduction

Nowadays the most problematic area of computer applications is "development of software intensive systems", within the frame of which the collaborative works of developers and other stakeholders are being fulfilled in corporate networks. "*A software intensive system is a system where software represents a significant segment in any of the following points: system functionality, system cost, system development risk, development time*. Examples are numerous: an ECU (Electronic Control Unit) in a modern car, processing engine for digital or mobile TV" [1].

The significant number of SIS developments (about 65 percent) either is being stopped, or is exceeding planned time and/or finance, or reach the end in the poorer version [2]. Investigation results of the successfulness problem (which are regularly published by Standish Group [3]) are presented in Figure 1. These results show that developers have not received very important means for successful developing of the SIS.

The usage of collaborative development environments (CDEs) is estimated as a promising way for increasing the level of a success in the named subject area. Such type of environments was defined by Booch and Brown in [4] where the place and role of communication means for the collaborative work were underlined and generally specified. Details of CDEs and their current state are presented in [4] and the most part of details is connected with communicative interactions of stakeholders in real time solving the tasks which are common for them.

It is possible to use a set of typical specifications of the CDE as a template for comparing the different technologies for the development of the SIS and for finding the ways for their evolving. In our opinion, one of such ways is "a real-time integrating the intellectual resources" which can help the developers in solving the complicated tasks. The intellectual potential of any developer is limited and developer often needs the help for the work with appointed tasks from useful intellectual sources. The base of such help is the reasoning of colleagues.

This article concerns the problem solving and decision making of the complicated tasks the work with which is being fulfilled in the CDE-system supporting the question-answer modeling of collective reasoning. The CDE-system is

Figure 1: Statistics of successfulness of the developing the SIS.

implemented as processor in corporate network which gives the possibility for question-answer programming of human-computer interaction with design objects and the design process.

2. Integration of Intellectual Activities

Many different kinds of complexity measures of tasks are used in the system engineering. For example, one can measure the complexity of the task by the number of subordinated tasks.

This version of measure is adequate for the development of the SIS where developers must solve a very big number of typical tasks. Over five hundred different typical tasks subordinated to the main project task can be used by developers only on the conceptual stage of developing the SIS with the help of the rational unified process [5]. Including such tasks into the design process decreases the complexity of the main project task Z^*; but the low degree of a success in designing the SIS shows that similar project tasks can be insufficient for design with the warranted success. The useful means for solving the design tasks and coordinating such processes are needed.

When the designer meets a difficult task he needs to remember that possibilities of human intelligence are limited and the additional intellectual support can help him. So there are bases to agree that in a complex situation for designers they need instrumental means which can give them a real-time access to all appropriate intellectual resources (and not only to intellectual capabilities of colleagues) of the corporate network.

This allows affirming that the integration of intellectual resources is needed for working with some complex design tasks. It is reasonable to choose useful means for the integration from a set of relevant AI means and include them into the instrument of designing the SIS. First of all the relevance must be estimated in the context of intellectual capabilities such as "consciousness" and "understanding" used for controlling the consciousness process.

When we choose or build relevant AI means we need to remember that the basic form of consciousness work revealing is the reasoning and the dialogue (the "question-answer" process in the brain structure) is a nature of the reasoning.

One way for modeling collective (collaborative) "consciousness" is to create the question-answer model (QA-model) of collaborative reasoning which is reflecting the consciousness process. However, for feedback the joint QA-model in any of its current state must be open for the interactive "visual pressure" on the brain of any member of a collective "consciousness" if it is useful for solving tasks (Figure 2).

The scheme in Figure 2 shows that collective reasoning $R(t)$ of the subject group $G(\{Sb_k\})$ in the frame of task Z transforms to the QA-model accessible to any member Sb_k of the group through operations under reasoning, its QA-model and questions $\{Q(t)\}$ and answers $\{A(t)\}$ as elements of this model. In such interaction the individual consciousness will be combined with the interactive "visual pressure" of the QA-model of collaborative reasoning (collaborative "consciousness"). The QA-model of the collective consciousness can be used not only for the interaction with the individual consciousness but also for creating the model of the collective understanding which must be useful for checking the intellectual activity.

It is necessary to notice that in creation of QA-models and their usage the specific understanding of a question will be used in this article. The essence of the question-answer activity is determined by a so-called "question". The "question" is a natural-artificial phenomenon appearing in attempts by a person to apply the own experience. In concrete active situation the "question" appears as a mismatch between the experience, which is necessary for the reaction and the existing experience of the person.

Determining "questions" as natural phenomena of a certain type it is necessary to specificity how such phenomena are found out, identified and described (coded). Any representation of the "question" in concrete language is its language model which fulfils required functions within the framework of reasoning. Language forms of reasoning (texts and speech), used by a person (or a team) during activity are capable to serve as information sources for constructions of sign models of "questions". The "questions" leave obvious and implicit "traces" in texts and speech and such "traces" can help to discover questions and define their attributes.

It is useful to find the "question" as a natural-artificial phenomenon appearing in specific conditions, to identify, to code and to use it according to its functionality. Any "question" as a definite mismatch in experience will activate the process as a result of which the definite "answer" will be constructed. The "answers" to "questions" develop an experience and can enrich the semantics of used language.

In accordance with understanding the "question" and "answer" the names "question" and "answer" will be used in what follows when speaking about phenomenon. Such names without quotes will be used in accordance with the context.

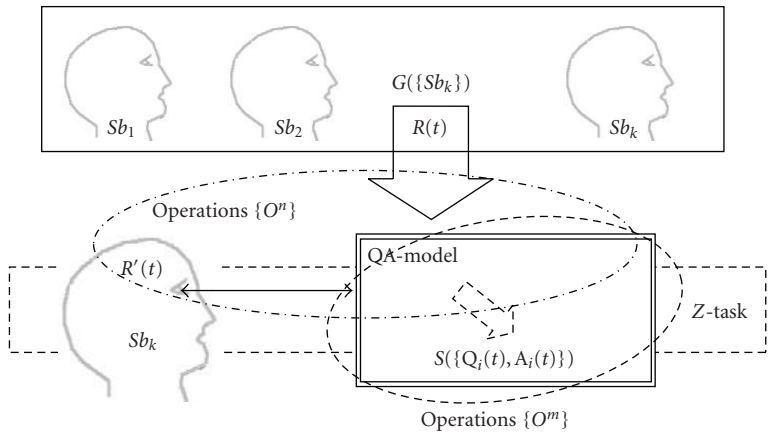

Figure 2: Scheme of question-answer base for interaction.

3. Related Works

The problem of rational reasoning in the development process of the SIS is well known in this subject area. This problem has been investigated for more than 10 years in the Software Engineering Institute (SEI) of Carnegie Mellon University [6]. However, the question-answer approach is not used and the problem of "a real-time integration of intellectual efforts" is not indicated in interests of the SEI to the schemes of reasoning and their formalizing.

Artificial intelligence means are not used for supporting reasoning of developers in such well-known technology as Rational Unified Process (RUP) and in other similar technologies, for example, in Microsoft Solution Framework and Eclipse.

It is very interesting because there are many types of reasoning which are investigated and modeled in AI. For example, the Programs of the European Conferences on AI (ECAI) include about 20 topics connected with modeling reasoning (analogical reasoning, case-based reasoning, common-sense reasoning, reasoning about actions, etc.).

We have the answer to the question "Why AI means is not used in technologies for developing the SIS?"

Adequate AI means which can increase the successfulness of designing the SIS are absent till now because problem-solving and decision-making based on the real-time integration of intellectual resources are investigated in AI only partially (different kinds of models for reasoning which are useful in definite classes of design situations, first of all case-based reasoning models).

We are convinced that investigation of question-answer reasoning is a perspective way for finding the AI means which can give the positive results helping to solve complicated tasks and not only in designing the SIS [7].

In the number of relative works using "questions and answers" (or QA), for example, we can mention reasoning in the "inquiry cycle" [8] for working with requirements, "inquiry wheel" [9] for scientific decisions and "inquiry map" [10] for education. Similar ideas are used in the special question-answer system which supports development of the SIS [11]. The typical schemes of reasoning for the SIS development are presented in [6], while in [12] reasoning is presented on seven levels of its application together with the used knowledge, and in [13] model-based reasoning is presented as useful means for software engineering.

However, in all publications referred to above, the issue [14] and the special report [15] the task of real time integration of the intellectual resources in processes of problem-solving and decision-making is not mentioned.

4. Question-Answer Models

In developing the SIS for each task of any degree of complexity the concrete developer is appointed for decision. Developer colleagues are entered to the solving process in that case when the intellectual help is required for the developer. Externally the general intellectual activity of the developer and its colleagues is observed in the form of their reasoning which can be registered by the useful way.

We suggest using the QA-model (mentioned before) for simulating collaborative reasoning in any state of the decision process for operative including such kind of models into the intellectual activity of the developer appointed to the task. The QA-model which reflects collaborative reasoning in the frame of the definite task $Z(t)$ we will name as the QA-model of this task (or shortly QA($Z(t)$)). The QA-model of the task is a model of collaborative reasoning (and integrated consciousness) in the real time process of solving the task.

The QA-model is a systematized representation of reasoning used during the solution of the task $Z(t)$ and kept in the special QA-database. Any QA-model is a set of interactive objects such as "question", "answer" and "task" with the certain attributes and operations.

Therefore specifications of the QA-models will be presented from the interactive system viewpoint or another words as specifications of a specialized software intensive system SIS^{QA}. Such position gives the possibility to use the experience of the SIS to the SIS^{QA} first of all the experience of the architectural description. We defined and investigated the QA-model of the task which is architecturally presented in Figure 3.

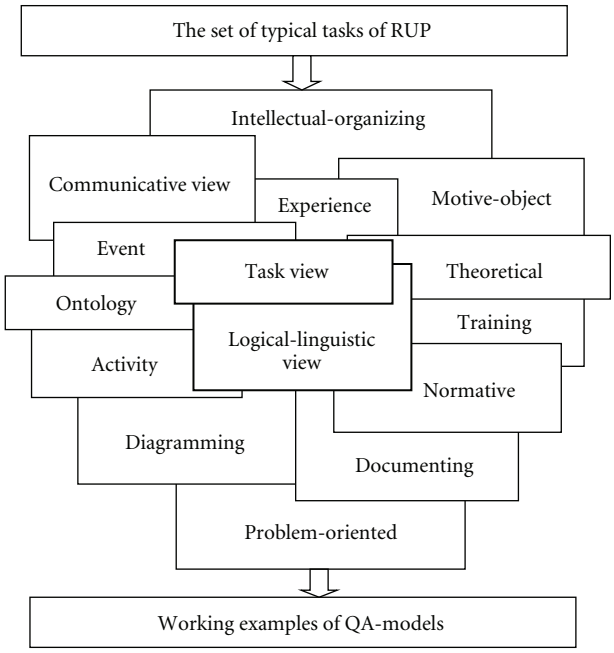

Figure 3: Architectural description of QA-model.

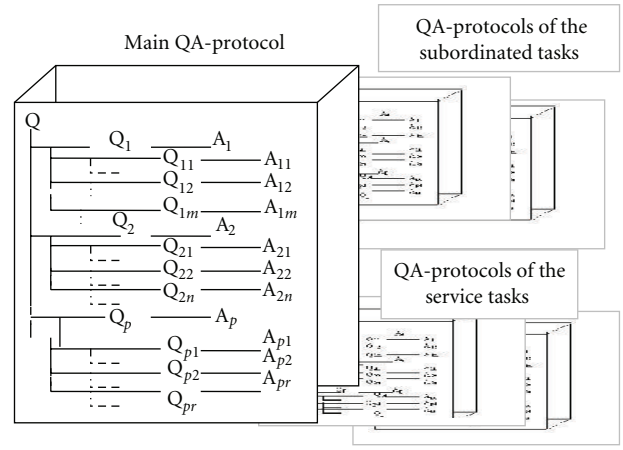

Figure 4: System of QA-protocols.

Question-answer models, as well as any other models, are created "for extraction of useful answers to the questions enclosed in the model". Any QA-model is adjusted to solving the corresponding task. In general case the QA-model must support the useful answering process for solving the main task of conceptual designing the SIS. Therefore the typical structure and content of the QA-model were being chosen and defined for general case of the task the role of which fulfills the set of typical RUP-tasks used by designers on the conceptual stage of the RUP-application. Such decision is being explained that the RUP is the richest source of typical tasks the utility of which is being confirmed by practice. Conceptual means of the RUP help to build and express the conceptual solution of any task in development of the SIS.

The typical QA-model is used as a template and as a complex of means for creating the example of the QA-model for any task in designing the SIS. The typical QA-model (or in other words the specialized SIS^{QA}) is constructed as the CDE-system which supports the collective work with tasks in the frames of following architectural views.

The logical-linguistic view presents $QA(Z(t))$ within frames of logic and linguistics of questions and answers. The visual representation of the view (Figure 4) includes a system of QA-protocols corresponding to the task tree of $Z(t)$. Each QA-protocol is a tree of questions and answers (QA-tree) which presents the reasoning used in the decision process of the corresponding task. Any question or answer in any QA-protocol is a result of translating the definite volume of natural reasoning about "question" or "answer". In general case any task Z can include subordinate tasks.

The logical part of the view describes the hierarchical tree of questions and answers. Each "question" or "answer" is being modeled as an interactive object with the unique name and rich visual presentation. The QA-tree as a whole consists of the main QA-tree (corresponding to the task $Z(t)$) and a set of subordinated QA-trees for subordinated tasks.

The linguistic part of the view includes a system of texts presented as the informational content of questions and answers. The text of each unit of the view is a result of the QA-formalization which is applied to the definite volume of reasoning at the natural language. The technique of formalization is presented below in Section 7. Such result can be interpreted and implemented as a translation of reasoning from the natural language to the QA-language.

Dynamics of the view reflects the history of reasoning registered by step-by-step as the history of each unit of the view. Therefore the logical-linguistic view was named as the "QA-protocol".

The QA protocol as a base of the logical-linguistic view has a number of useful interpretations such as

(1) it reflects the definite volume of the design process as a data of "the research experiment" representing "the primary measuring information" about design process and about the used control facilities;

(2) the content of the protocol reflects a real reasoning, which can be investigated to increase the knowledge about a "phenomenon of reasoning and questions";

(3) each of registered questions or answers admits its interpretation as an event which is essential for reasoning and design process, that allows understanding the protocol as "a network of events" ordered in time;

(4) the protocol is a data structure (QA-structure) with its practically useful set of operations. Such interpretation of the QA structure corresponds to the experience of Computer Science in the area of data structuring for adequate presentation of reasoning.

The task view opens the model $QA(Z(t))$ as an interactive task tree (Figure 5) including the interactive model of $Z(t)$ with models of all subordinated tasks. "The task" is a type of "the question" and, therefore, it is subordinated to the used understanding of "questions". However, the interactive object

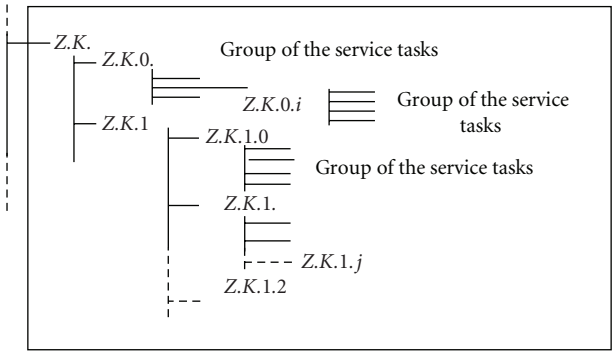

Figure 5: Fragment of the task tree for $Z = Z.K$: K-index name, symbol "0" indicates service tasks, other names indicate the problem-oriented tasks.

which presents the task in SIS^{QA} is opened for the interaction through the special set of commands one of which is "To open task as its QA-model". In general case the task tree is a tree of all project tasks combined in the task Z^* of designing the SIS. The task is a based unit for structuring the process of designing and for question-answer modeling (QA-modeling). In such process a problem-oriented and service types of tasks are differentiated. As shown in Figure 5 the service project tasks can be used for solving the problem-oriented tasks.

The example of the visual integration for the *task view* and the *logic-linguistic view* is shown as the real screenshot in Figure 6 where any label of the task or the question or the answer can be used for the interactive access to the other parts of the chosen interactive object. The example also visualizes the structure of the QA-protocol connected with the chosen task in the task tree. Here is shown that the picture can be attached to any unit of the task tree and the QA-protocol.

The ontological view indicates all concepts used in QA($Z(t)$). It is acceptable as an interactive list of concepts with references to the project ontology which is being constructed and used during the life cycle of the SIS. The inclusion of the ontological view into the QA-model helps to the developer to express and to check the definite volume of the own understanding on the level of used notions. The special subsystem is included to the SIS^{QA} for supporting the creation and usage of the "ontology of the project" in real time.

The theory view presents the system of QA-protocols as a theory of the content-evolutionary type. It is being accessible after transforming of the current state of QA-protocols to the text form (DOC or HTML). The theory view is constructed from systematically combined text units in the step by step process of collaborative designing. It helps the designer to "transfer" operatively from the QA-structure of reasoning to its textual form reflecting the evolution of reasoning from the causal point of view.

The intelligence-organizing view gives an interactive access to the organizational structure of designers and other stakeholders involved in the solution process of $Z(t)$. It can help to find interactive answers not only about any person involved to the solution process but about the person contribution to the project also.

The communicative view opens all communicative relations used between solvers of the task $Z(t)$. A special set of service tasks (special e-mail, meeting, brainstorming, estimation, etc.) is opened (in the QA-processor) for including the necessary communication into the solution process

The experience view presents a list of all models of experience used in the solution of the task and units of experience which have been built during this process. The special open library of experience units (models) is included in the CDE-system used for the development of the SIS.

The motive-aim view registers and demonstrates visually a semantic net of motives and aims with task specifications the implementing of which supports motives and aims. It implements the principle according to which for each motive or aim it is necessary to indicate how it is materialized in the life cycle of the SIS.

The event view is an interactive net for the access to "questions" and "answers" each of which is presented (with the help of the needed attributes and their visualizing) as event in the designing.

The activity view registers "questions" and "answers" of any types as objects of activity. This view has two versions one of which is a Gant diagram and the second is a PERT-diagram.

The problem-oriented view is presented as a number of interactive semantic schemes describing the important aspects of the task solving. Each scheme is visualized as the useful (for designing) graphic picture. The special graphic processor is included to the SIS^{QA} for supporting the creation and usage of the "block and line" diagrams.

The diagramming view demonstrates a set of the normative diagrams (UML-diagrams or others "block and line" diagrams) each of which presents the declarative or procedural aspect of the task solving. Each of the diagram is built with the help of the QA-model for the corresponding service task subordinated to $Z(t)$.

The documenting view opens the process and results of the solving task through the normative set of documents for the used technology.

The normative view presents an interactive list of standards and frameworks used during the task solving.

The training view combines all QA-means of the named views, which are useful for the training purposes. Such possibility helps newcomers of the stakeholder group or designers to study the project units.

Representation of the QA-model and interaction with it as with the specialized automated system SIS^{QA} based on a set of presented architectural views requires answering the question about systematization of all views. The following interconnected components are used for such aim:

(i) declarative-visual integration, within the frame of which all artifacts of the QA-model are presented and interactively visualized;

(ii) procedural integration as the technique system for QA-modeling.

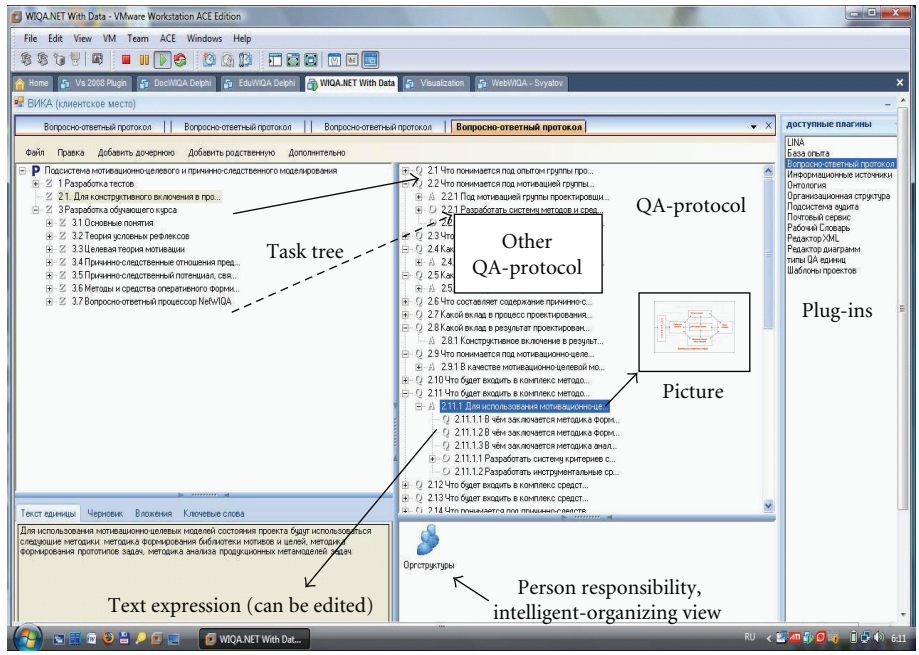

Figure 6: Relation between logic-linguistic view and task view.

Declarative-visual integration is responsible for coding, structuring, ordering, storage, and delivery by inquiries of those artifacts which are parts of the QA-model. The function of the integration of such type we will assign to the question-answer database of the SIS project. Such solution allows saving all QA-models at the file-server accessible for any workplace of the corporate network. It opens the possibility to include in architecture of the SIS^{QA} two styles—repository and client-server styles. Visualization of artifacts is another problem for the decision of which it is rational to use architectural style Model-View-Controller (MVC) widespread in practice of the development of the SIS [16]. Architectural style MVC is the style oriented specially on the human-computer interaction.

Procedural integration is being implemented in the form of a question-answer programming in according to which creation of any QA-model and interaction with it are fulfilled as a special type of programming in an instrumental environment of a processor type. Such type of programming will be presented in what follows.

5. Question-Answer Means

The QA-model of the task $Z(t)$ with its system of views are defined as the problem-oriented base of the software intensive system SIS^{QA} designated for supporting the design of the SIS. In the general case the task $Z(t)$ is a task $Z^*(t)$ solving of which is conceptual designing the SIS. Therefore the SIS^{QA} must be built as a complex of QA-means designated for conceptual designing any SIS.

The system of QA-means named as QA-processor Working In Question-Answers (WIQA) has been implemented in several versions. Developments of two last versions were based on architectural views of the QA-model and usage of repository, MVC, client-server and interpreter architectural styles. Moreover in developing the versions have been used object-oriented, component-oriented and service-oriented architectural paradigms. One of the last versions named as NetWIQA has been programmed on Delphi 6.0 and the second version (named as WIQA.Net) has been created on C# at the platform of Microsoft.NET 2.0.

The system WIQA.Net has been developed for its usage as the kernel of the product line each unit of which is an application based on this kernel. The product line includes applications "conceptual design", "system of documenting", "decision-making system," and "training system" adjusted for their usage in the corporate network. The NetWIQA was used as a reach source of assets for developing the kernel [17]. Server possibilities of WIQA.Net are opened for thin clients in the corporate network and for Web-clients via the Web-shell programmed with using means of ASP.NET.

The component structure of WIQA.Net is implemented as the open system of plug-ins supported by the implementation of all architectural views of the QA-model. It is generally (without means reflected and supported web access to the QA-means) presented in Figure 7.

All components are allocated on following tiers:

(1) *The database tier* is implemented with use of the technology ADO.NET. It is used as a repository of all data.

(2) *The server plug-ins tier* encapsulates all function for interaction with the database. A number of plug-ins in this tier is specialized. The tier is formed dynamically during starting the server with using the technology of a dynamic reflection of types which is supported by a platform.NET. Interfaces of the tier (server plug-ins) contain the functions intended for direct adding, updating and removing of data.

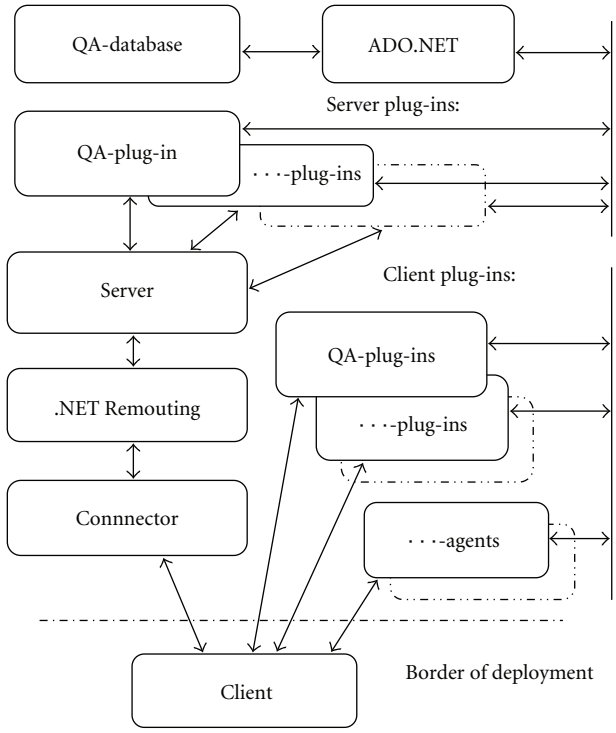

Figure 7: Component structure of the WIQA.Net.

These functions are called only by a client tier (client plug-ins).

(3) *The server tier* is implemented using the technology.NET remouting, that allows addressing the server from the remote client workplace. When any client workplace is starting the server carries out registration of the client, giving out to the client a unique key which is used for checking the access rights to various functions of system. The server registers also starting of any component of a client plug-ins tier. The server is conducting also logging the most important events of system (i.e., formation of magazine of events).

(4) *The connector tier* (an intermediate tier). The usage of the technology .NET Remouting demands the certainty for all functions of the remote object for access to it. The connector hides the implementation of the server from the client. Therefore the interface of the server is being stored in the connector.

(5) *The client tier* is an application with which begins the work of the user on the client workplace. This tier is responsible for the program encapsulation of the work with the client plug-ins tier and also for encapsulation of the remote interaction with a server. In the case of an allocation of the server and the client on one computer, through a configuration file it is possible to disconnect the usage of the technology .NET remouting. It will raise the speed of the work of the program. The client is also responsible for the reception of the name and the password of the user.

(6) *The client plug-ins tier* consists of two parts. One part supports the active interaction with the user and other part is adjusted on the processing of the certain events in the system.

The first part gives the access for the user to the functionalities of the implemented architectural views. Each plug-in of this tier contains a unique key which gives the possibility to distinguish plug-ins of the tier for controlling the access of the user to the appointed plug-ins only. The QA-plug-in of this tier provides the access to commands for the work with the logic-linguistic view (with QA-protocols). Each plug-in of this part of the tier supports the interaction with objects of the corresponding architectural view. The second part of the tier supports the automatic work with a definite set of events during QA-modeling. It is provided by the units programmed as agents.

(7) *The tier of continuous developing* is activated as a real-time access for the developer of additional plug-in (or agent) from the client workplace to the .Net 2.0 means in context of the current state of the QA-processor. Such additional units are shown in Figure 7 by chain lines.

Only QA-plug-ins for the work with QA-protocols have names in Figure 7. Other plug-ins for the work with other interactive units of QA-models are presented as classes of plug-ins. System of means included to the WIQA.Net supports the (collaborative) solution of any task in conceptual designing the SIS. Such possibility is stipulated by the potential of the QA-model which is defined on the base of analogy with the RUP tasks. It is necessary to notice that WIQA.Net can be qualified not only as CDE-system for conceptual designing the SIS but also as a processor for the creation (and execution) of QA-programs for complicated tasks.

Web-access in WIQA.Net is arranged as the one-page site (asp.net) with dynamic additional loading the data, executed on the technology AJAX . At initial loading the user registers the URL address of a resource on which site is placed. It is unique GET-request at work with the system. The further reception and sending of data is carried out by means of asynchronous POST-requests which are carried out by methods of a client part of libraries Microsoft ASP.NET AJAX extensions and Ext JavaScript.

In Figure 8 one of the screenshots of the Web-shell is presented. This version of the shell (named as EduWIQA) is the Web-shell of WIQA.Net adjusted to the educational problem.

The screenshot demonstrates the current state of the visualization for a number of answer-units chosen for interaction from the QA-model. These units are located on the screen by user and change each other in front according to the chosen tempo. In the QA-processor there are several versions for the dynamic allocation of visualized units on the monitor screen.

6. Question-Answer Modeling

Question-answer models, as well as any other models, are created "for extraction of answers to the questions enclosed in the model". Moreover, the model is a very important form of representation of questions, answers on which are generated during interaction with the model.

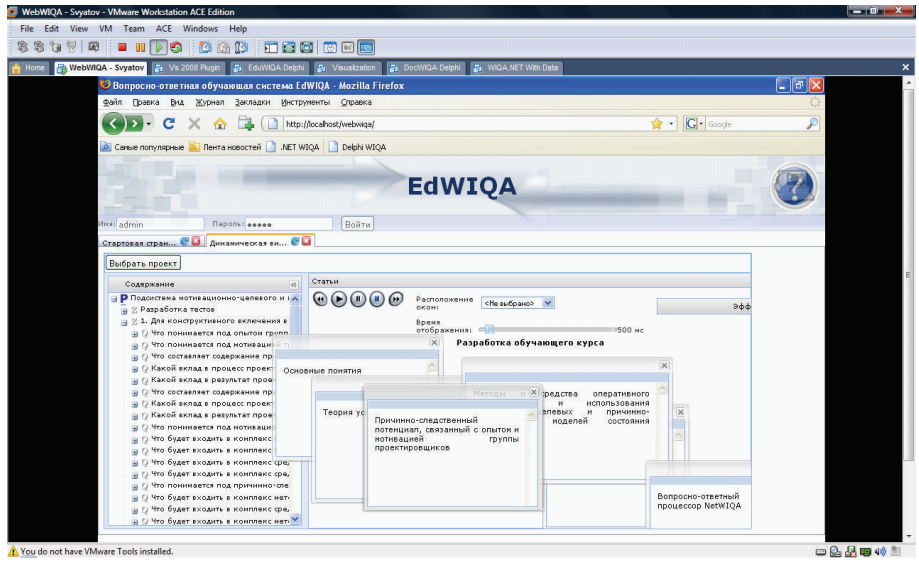

Figure 8: Web-view of visualization for QA-model.

The essence of QA-modeling is an interaction of stakeholders with artifacts included to the QA-model in their current state. For such interaction the developer can use the special set of commands (QA-commands), their sequences and a set of plug-ins combining with QA-commands. Such work is similar to programming (QA-programming) on the base of means of the special processor (QA-processor) the role of which fulfils the SIS^{QA}.

In order to get the definite positive effect from the concrete QA-model of the definite task the developer will need to program the definite volume of the developer interaction with this model and execute the created QA-program.

There is a number of expected positive effects for each of which the QA-program (or a set of alternative QA-programs) must be rationally built. It has given the possibility to create a library of typical QA-programs accessible to the developer in real time.

The main subset of positive effects of QA-modeling is connected with real time integrating of intellectual resources and this subset includes:

(i) controlling and testing the reasoning of the developer with the help of "integrated reasoning" and "integrated understanding" included into the QA-models;

(ii) correcting the understanding of designer with the help of comparing it with "integrated understanding";

(iii) combining the models of collective experience with individual experience for increasing the intellectual potential of the designer on the definite working place;

(iv) including individual experience of the developer in accordance with the request on the other working places in the corporate network.

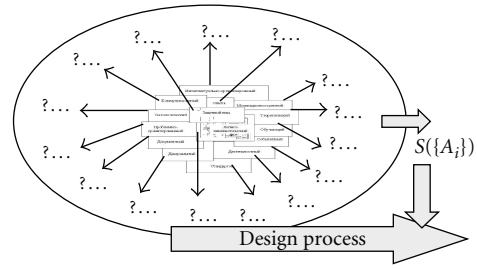

Figure 9: QA-model as a source of answers.

Any developer can get any programmed positive effect with the help of QA-modeling as "answer" on question actually or potentially included in the QA-model (Figure 9).

As it is shown in this scheme any view is the source of answers accessible for the developer as results of the developer interactions with the QA-model. At the same time the potentialof the QA-model is not limited by the questions planned at defining and creating the QA-model. Another source of useful effects of QA-modeling is an additional combinatorial "visual pressure" of questions and answers which is caused by influence on brain processes of their contact with componentsof QA-models. There are different forms for building answers with the help of QA-modeling, not only linguistic forms. But in any case the specificity of QA-modeling is defined by the inclusion of additional interacting with "question-answer objects" into dynamics of the integrated consciousness and understanding (into natural intellectual activity).

The developer achieves the positive interaction in the work with problem-oriented tasks and service tasks (including a subset of technological tasks) which are being defined and solved in the designing process with the help of QA-modeling.

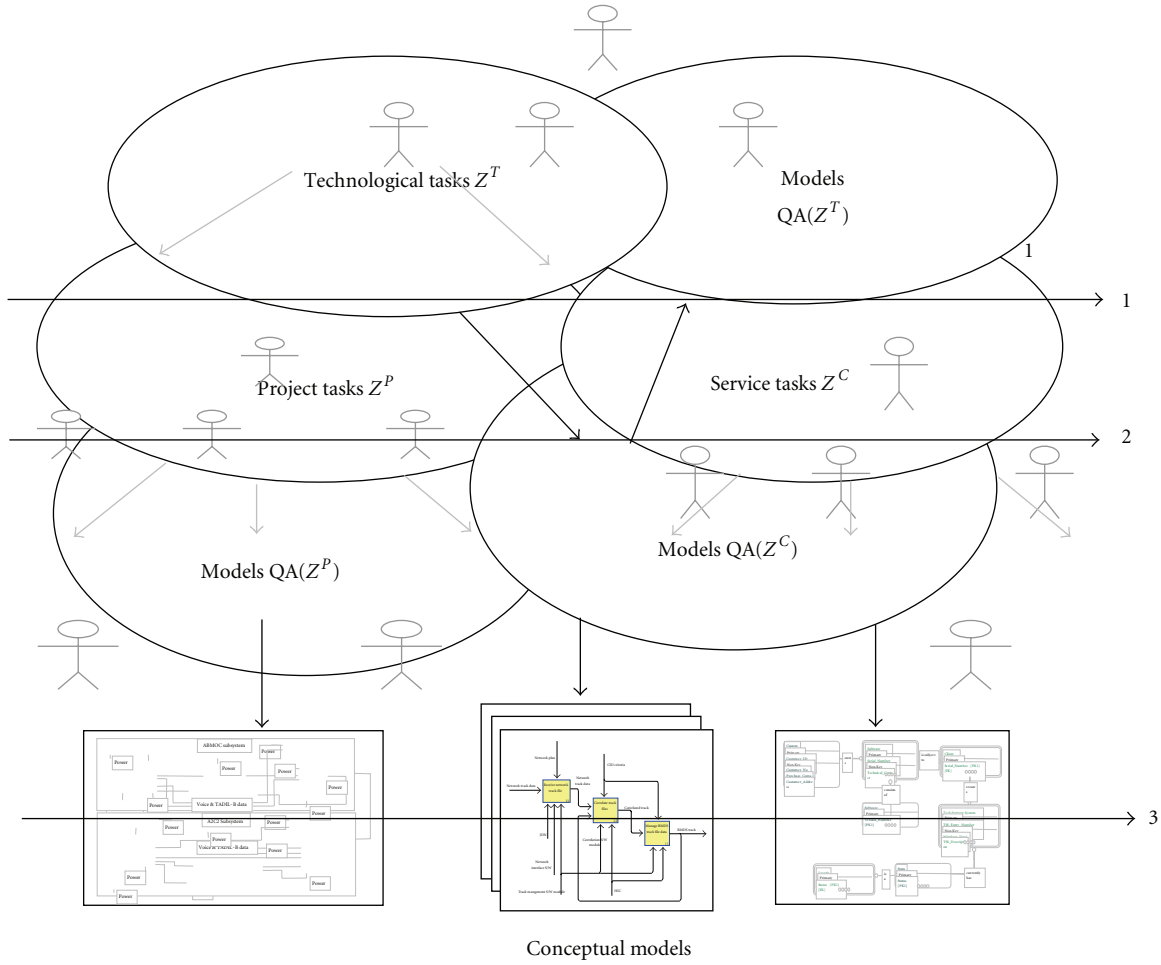

Figure 10: Relations between tasks.

All problem-oriented tasks $Z^P = \{Z_r^P\}$ are derived from the analysis of the subject area of the SIS, from requirements to the SIS and from the design process. Any task Z_r^P is a question qualified by stakeholders as a task-question answer to which can be found only through the decision process.

Any service task Z_m^C has its typical QA-model which is extracted from the special library. Such pattern (QA-procedure) helps to build the model QA(Z^C) for the definite conceptual artifact. Service tasks are defined and implemented for creating documents and visual diagrams, for supporting the typical schemes of communicating and training.

The work with questions, answers and other conceptual artifacts is executed with the help of technological tasks $Z^T = \{Z_n^T\}$ solving each of which uses a scenario form. The solving of the definite technological task is an executing of the corresponding technique programmed for the definite application of the QA-processor.

The main application of the QA-processor is a "Conceptual designing" the process of which is based on workflows "Interactions with Experience" [7]. Each workflow is implemented with the help of definite technological tasks. Nowadays about 70 technological tasks have been developed. They are accessible to designers through the special shell of a client workstation. Technological tasks are used for the conceptual decision of all problem-oriented tasks Z^P and other service tasks Z^C. Relations between tasks Z^T, Z^P and Z^C are schematically shown in Figure 10.

It is possible for developers to work at two lines of activity. The line 1 is a line of strict executing of QA-method of designing with the help of the technological tasks only. The line 2 is a line of actions on the base of QA-commands and plug-ins of the QA-processor. The developer can switch between lines if it can give more effective result.

7. Question-Answer Programming

The potential opportunity to create useful samples of interactive objects such types as "problem", "question" and "answer" and existence of means (QA-processor) for their combination and linkage has led to the decision to implement the creation of such objects, their assembling and using in the form of programming.

For such decision there are sufficient bases which include the following arguments:

(1) the named objects are implemented in the QA-processor as a set of specialized types of data with corresponding operations, and all these means support the QA-coding of reasoning used for the problem-solving;

(2) the decision of the task Z presented as a system $S(\{Mi\})$ of conceptual models $\{Mi\}$, which has been built on the basis of the necessary reasoning $R(Z)$, is one of the conceptual solving the task Z;

(3) QA-coding of reasoning $R(Z)$, aimed at construction of QA-model of task Z, leads to QA-codes $\mathrm{QA}(R(Z))$ which can be transformed to the system of models $S(M)$ and such work is supported in the QA-processor (*diagramming and documenting views*).

The decision of task Z materialized in the form of $\mathrm{QA}(R(Z))$, is open for its useful application in the environment of the QA-processor, including reuses.

All of named arguments allow qualifying the construction $\mathrm{QA}(R(Z))$ as the program, as the QA-program. The introduction of such type of programming allows separating the creation of QA-models from their usage in QA-modeling.

Interpretation of $\mathrm{QA}(R(Z))$ like program can be extended naturally in the case of the task which has subtasks. In this case the corresponding QA-program should be built for each subtask and its QA-code should be opened for the call from the main task Z and its subtasks, including the recursive calls.

Such possibilities (typical for traditional programming) are supported by the QA-processor. Moreover, the QA-processor supports the work with the library of QA-programs.

Introduction of a special kind of programming leads to questions about the formal definition of language for QA-programming, transformations of QA-programs and their execution.

Answering such questions it is necessary to take into account that QA-programs of tasks are classified as specific kinds of programs not only because of specificity of types of data but the execution of each of them is possible also in any state of its creating.

The QA-processor supports the work with QA-programs written on the QA-language with grammar which uses the following BNF-rules:

⟨QA-program⟩ ::= ⟨Heading⟩⟨QA-code⟩ |
| ⟨QA-program⟩⟨Heading⟩⟨QA-code⟩.
⟨Heading⟩⟩ ::= ⟨Name⟩⟨Statement of a problem⟩.
⟨QA-code⟩ ::= ⟨QA-group⟩ | ⟨QA-code⟩⟨QA-group⟩.
⟨QA-group⟩ ::= ⟨Question⟩ | ⟨Answer⟩ |
| ⟨QA-group⟩⟨Question⟩ | ⟨QA-group⟩
⟨Answer⟩.
⟨Question⟩ ::= ⟨Problem⟩ | ⟨Task⟩ | ⟨Inquiry⟩.
⟨Question⟩ ::= ⟨Text⟩ | ⟨Interactive object⟩.
⟨Task⟩ ::= ⟨Subtask⟩ | ⟨Task⟩⟨Subtask⟩.
⟨Answer⟩ ::= ⟨Project⟩ | ⟨Decision⟩ |
⟨Idea⟩ | ⟨Hypothesis⟩ | ⟨The answer to Inquiry⟩ |
| ⟨Specification⟩ | ⟨Motive⟩ | ⟨Purpose⟩.
⟨Answer⟩ ::= ⟨Text⟩ | ⟨Interactive object⟩.
⟨Object⟩ ::= ⟨“Project”⟩ | ⟨“Task”⟩ |
| ⟨“Answer”⟩ | ⟨“QA-group”⟩ |
⟨Function from object⟩.
⟨“Task”⟩ ::= ⟨QA-model⟩.

It is necessary to notice, that the grammar description can be used as the definition of the QA-program. A number of versions is used for execution of QA-programs each of which is being fulfilled as one of the following forms of QA-modeling:

(1) the real-time interaction of the person solving the task Z (or similar inquiries of any other persons) with any fragment of the QA-program in any state of its readiness;

(2) the interaction with the program $\mathrm{QA}(R(Z))$ controlled by a technique chosen from the library of techniques where each of them is materialized in the form of the QA-program;

(3) the human-computer execution of the technique chosen from the library mentioned above.

In any version of interaction its acts are caused by activity context which is subordinated to the basic purpose—to solve the task Z. For achievement of such purpose the subject solving the task interacts with its QA-model allowing him to use (in construction of the decision) combination with objects of the types “task”, “question” and “answer”, selecting suitable objects and connecting them in “the conceptual decision of the task Z”.

The addressee of effects of interaction with the QA-program is the person interpreting its codes and using results of interpretation in the following basic purposes:

(i) to understand the object (on the monitor screen) with which the person interacts and to use the effect of understanding;

(ii) to use the result of interaction in the communicative purposes for promoting the decision of the task to the final result;

(iii) to analyze object of interaction;

(iv) to use effects of interaction for estimation;

(v) to promote “forward” the decision of task Z, first of all due to decision-making.

The QA-processor is implemented so, that its interfaces promote the real time work to the named purposes. So, for example, the subsystem of dynamic visualization of question-answer units is included to the QA-processor in order to support the process of understanding. The set of visualized units is being selected by inquiry of the user. There is a special subsystem in QA-processor which supports

the preliminary preparation of the visualizing information from the QA-database. Any prepared block of questions and answers can be visualized in suitable tempo by switching special windows on the screen or using effects of the 25th card. Such version of interaction performs also the role of the interactive "visual pressure" of the QA-model on the intellectual activity of the designer.

Let's present one more technique of the support which is included in the QA-processor for promoting the creation the QA-program (and promoting the decision of task Z also).

The essence of the technique consists in extraction the potential set of questions $\{Q_k\}$ from the current state of program $QA(R(Z(t)))$ for the subsequent creation of answers $\{A_k\}$. Step by step the technique promotes the creation of program $QA(R(Z(t)))$ from the initial state of the program $QA(R(Z(t_0)))$ to the final result.

This technique has been developed for the solution of tasks in conceptual designing the SIS but it can be applied to the decision of any (complex) task.

The execution of the technique begins with an initial state of the program $QA(R(Z(t_0)))$ for which the generalized statement of task Z has been formulated with the usage of the sample consisting of three clauses. The first clause reflects the main purpose of the task. It opens the access to the questioner about structure and content of the typical Use-Case diagram [16] which should be adjusted to the specificity of the task. The second clause defines the dynamic behavior corresponding to the Use-Case diagram for the task. It provides construction of the basic diagram of business-objects of UML. The third clause defines technology of implementation of a system under design. Information of this block is applied in conceptual design as context information.

Analysis of text T_0 of the general statement of a task and its translation to PROLOG-like language are used for extraction of questions. More detail it is based on step by step registering of questions and answers in QA-program (the visual presentation of which is QA-protocol) in accordance with following points of the technique.

(1) The set of questions $\{Qi\}$ is extracted from the text T_0 and coded by adequate texts $T(Qi)$ for each Q_i.

(2) Actions of the item (1) are executed for each text $T(Qi)$, therefore the set of questions $\{Qij\}$ and their codes $\{T(Qij)\}$ is being formed also. Actions of item (2) are being used to control the correctness of question codes.

(3) The subset of questions $\{Q_k\}$ which will be used for the next step of stepwise refinement, is being chosen from the united set $\{Qi\} \cup \{Qi_j\}$. Other questions of the set $\{Qi\} \cup \{Qi_j\}$ are being recorded for their usage in the subsequent steps of QA-programming.

(4) Set of answers $\{A_k\}$ and their codes $\{T(A_k)\}$ is being formed and registered in the current state of the QA-program.

(5) Each text $T(A_k)$ is being processed as the text T_0.

(6) The cycle (1)–(5) is being repeated until the solution process comes to the end.

In working with QA-programs, it is necessary to discriminate their declarative parts (question-answer blocks), the procedural components (techniques of interaction with blocks, each of which is visualized in a question-answer form) and interpretation (execution of QA-programs).

Interpretation is used in two versions, one of which operates the interaction with the model during the solution of tasks, and another provides dynamic visualization of question-answer blocks for their influence on the intellectual activity of the designer. Dynamic visualization of question-answer blocks by inquiry of the designer is the base form of the developer real-time access to the integrated intellectual resources.

It is possible to use two versions of QA-programming. The first version is similar to event-driven programming when the state of the QA-model is interpreted as the description of the situation used for the next steps of designing. If any question or answer of the QA-model is under constructing then such unit is the object of the potential work. Analyzing the state of the QA-model (more truly "*event view*" of the model), we can choose its more priority question or answer for continuing or finishing the work with such interactive object ("*activity view*" of the QA-model).

In any state of any question or answer the developer can activate the useful interaction with the chosen object using accessible QA-commands, plug-ins or techniques. Such actions are similar to the work of the QA-processor as "an interpreter".

The second version is used for simulating the technique steps in the QA-form where "questions" are used for coding steps of the technique and "answers" are used for registering facts of executing steps of the technique. Any step of the technique as a definite volume of the work is being presented as "question" for which is needed to build "the answer".

The QA-processor suggests to the designer many different types of the QA-commands, plug-ins and techniques used in designing with the help of QA-modeling. Any designer has the possibility for using a set of normative QA-techniques for designing, decision-making, communicating, documenting and training.

A set of especially important actions (operations, commands, plug-ins mechanisms, techniques) of the QA processor includes:

(1) for questions: detection of obvious questions (on their indicators), predication (through translation on Prolog-like language), identification (on patterns), concrete definition (for types), assignment of meanings to attributes (as to the phenomenon of event type), argumentation of question;

(2) for answers: creation, assignment of a type, change of a type, registration of a condition, editing of the contents, assignment of meanings to attributes (as to the phenomenon of event type), argumentation;

(3) for QA groups: transformation to the node, expansion into the QA-structure, transformation to the event net, visualization of a network, analysis of

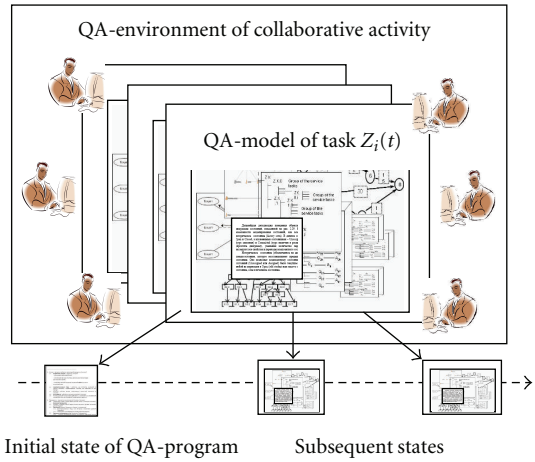

Figure 11: Form of existing of the programmed task.

a condition, choice of a direction of development, scrolling of dynamics (on inquiries);

(4) for texts: creation, transformation, grammar analysis, semantic analysis, transformation to the semantic graph, supporting of a phenomenon of attention.

Any program and QA-program also is a model of the programmed task. The introduction of QA-programming allows separating the creation of QA-models from their usage. The relation between QA-model and QA-program is presented in Figure 11. Any QA-model of the corresponding task Z_i is being created and existed as an interactive complex combining the chosen artifacts of appropriate views. The needed views are included to the model of Z_i in accordance with the task type.

The role of the initial state of the QA-program fulfils the typical QA-procedure if it is existed. The execution of the QA-program performs in the form of QA-modeling.

8. Question-Answer Designing

First of all QA-modeling is used for solving the design tasks. Moreover this kind of modeling helps to build the conceptual solution for any task during the design process of the SIS. We suggest rational method of conceptual solution for the system of design tasks being executed in the corporate network.

In the most general case the application of the method begins with the first step of question-answer modeling the initial statement (Figure 12) of a development task $Z^*(t_0)$. In special cases of the method application the initial statement of a task is included in a task tree of the design technology with which it will be fulfilled.

The essence of the method consists in the following.

(1) The system of tasks of conceptual designing is being formed and solved according to the method of stepwise refinement.

(2) The initial state of the stepwise refinement is being defined with the system of normative tasks of "life cycle the SIS" which is included the main project task Z^*. The basic version of normative tasks corresponds to standard ISO/IEC 12207.

(3) Implementation of the method begins with the formulation of the main task statement in the form, allowing start constructing of the prime conceptual models.

(4) Detailed elaboration in the system of tasks join not only the project tasks connected with specificity the SIS but also service tasks, each of which is aimed at creation of the corresponding conceptual diagram or document. The strategy of the collective work with project and service tasks is presented in Figure 13.

(5) For each service task its question-answer model is created on the base of the definite question-answer pattern from the special library.

(6) During conceptual solution of any task, included in the task tree of the SIS project, additional tasks can be discovered and included into the system of tasks.

(7) General conceptual solution integrates all conceptual decision of all task included in a task tree of the project.

(8) Conceptual solution is estimated as the completed solution if its state is enough for the successful work at the subsequent development stages of the SIS. The degree of sufficiency is obviously and implicitly checked. Useful changes are added to the more adequate conceptual representation of the SIS.

As a forementioned, above general conceptual decision integrates all conceptual decision of all tasks included in a task tree of the project. So, conceptual solution of the project tasks is defined as the system of conceptual diagrams with their descriptions on the notion languages the content of which are sufficient for successful programming of the solution of the task. Which conceptual diagrams are included into the solution depends on the technology used for developing the SIS.

Potentials of QA-means are sufficient for conceptual designing the SIS. The success and quality of the QA-designing essentially depend on that which complex of technological tasks and service tasks is combined in the technology used by designers.

All tasks of named types are open for their QA-programming. For creating the technology, it is necessary to choose the appropriate conceptual artifacts and techniques for transforming them (with the help of editing only) to the QA-forms and loading such forms in the library of the QA-templates. After that designers can activate the execution of the method for QA-designing.

In Algorithm 1, the fragment of the QA-program for one of the service tasks is presented. This is a task helps the designer to create the Use-Case diagram. The fragment reflects the initial state of the program where answers expound to the designer their meaning. These answers are necessary to fill by the definite content.

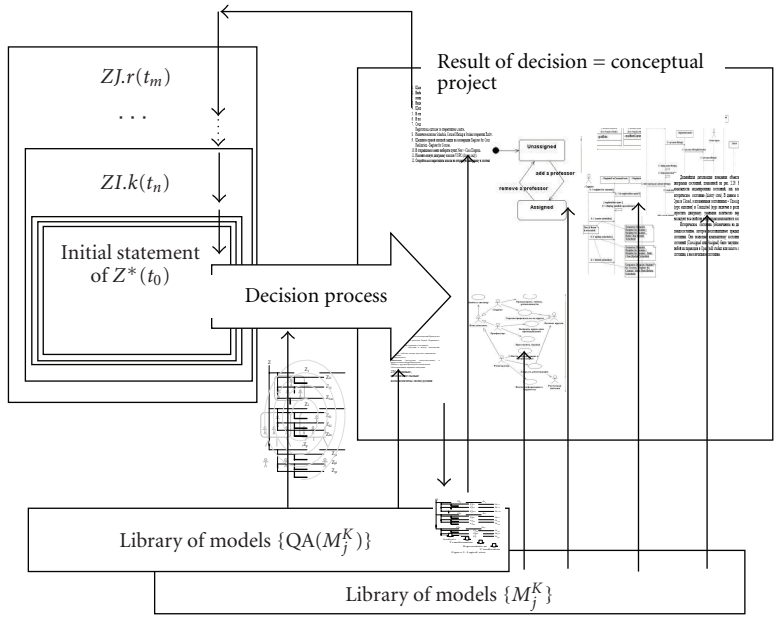

Figure 12: Scheme of method.

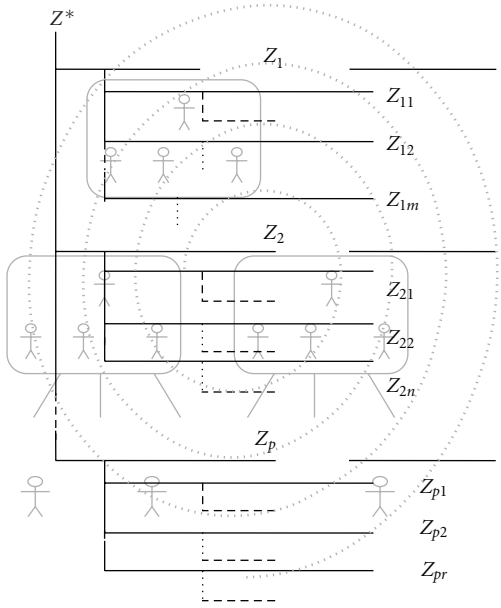

Figure 13: Task's tree of the development process (strategy of a collective activity).

The method has received the name "question-answer method for conceptual decision of the project tasks". In the name of this method the attribute "conceptual" indicates that base actions of the decision process are "conceptual actions" of stakeholders, first of all question-answer reasoning used in the design work. Such conceptual actions are needed to build an adequate conceptual representation of the SIS.

The application of the method supposes that any its technique can be interrupted in any time. The special subsystem for controlling the interruptions of techniques is included in the QA-processor.

9. Question-Answer Decision-Making

QA-programming of the technological tasks and service tasks will be demonstrated on the examples of decision-making (DM).

There are many different types of the DM methods. Choosing methods suitable for designing the SIS we need to take into account the specificity of such kind of activity. Therefore for certainty we are being limited below the interactive decision-making, decision-making in group, decision-making by experts, and also analysis models of situation and alternatives. In all these methods the human-computer interaction is important (as in conceptual designing also).

In order to use the possibilities of the QA-processor the typical task must be defined for each DM method suitable for designing the SIS. Moreover, each typical task must be presented as the corresponding technique of the DM coded in the form of the QA-program.

QA-programming of the chosen DM tasks showed that decision-making by the technique of expert system has specificity in implementing and executing. At first we present the general version of QA-programming and then QA-programming for the expert DM task. For example, the QA-code of the technique for the Plus-Minus-Interesting analysis (or PMI-analysis) [18] is presented in Algorithm 2.

The answers in the template are absent till its using. After loading this template in the task tree it will receive the status of "the QA-model" for the new subordinated PMI task and for each question of this model the answer will be opened. When the point of technique of the PMI task has been executed the symbol "$*$" is added to the answers for questions which have been fulfilled as points of technique.

Usage of value "$*$" in answers gives the possibility for breaking the technique in any position and returning to the next point of the technique in suitable time. Moreover it gives

Q2. Actors?
A2. The external persons and systems cooperating with the system being designed.
2.1 Primary Actors?
A2.1. Persons and systems who and which will initiate interaction, will use functions of system for the decision of the task and will receive valuable result (useful effect) from this interaction.
Q2.2. Interested persons?
A2.2. Persons and systems, whose interests consider, protect and satisfy in system being designed.
Q2.3. External events initiating any processes in the system being designed.
Q2.4. Assistants?
A2.4. Persons who are used by the system.
Q2.5. Why the given actor cooperates with the system?
A2.5. Under own initiative or how the intermediary, the representative of another.
Q3. Precedents—variants of use of the system by actors?
Q3.1. What task is solved by the actor with help of the system?
A3.1. The name of the first precedent.
Q3.2. What is the higher system for which the system is being designed?
Q3.3. Participants (besides the Primary Actor—the initiator)?
Q3.3.1. Interested persons, whose interests are mentioned with the given precedent?
Q3.3.2. The assistants participating in realization of precedent?
Q3.4. Frequency rate of association between the actor and precedent?
Q3.4.1. How many "copies" of the actor (concrete subjects) can (simultaneously) take part in realization of precedent (the concrete script)?
Q3.4.2. How many copies of precedent can (simultaneously) be realized by one actor (the concrete subject)?

Algorithm 1: Fragment of QA-program for creation of Use-Case.

Question-Answer Template File
"The scheme of PMI management"
Q1. Specifying a problem of decision-making?
Q2. Creating a subtask of decision-making (PMI)?
Q3. Creating a subtask of documenting of decision-making?
Q4. Loading a pattern of method PMI?
Q5. Identifying of problems and the purposes in a pattern?
Q6. Forming the list and an estimation of pluses?
Q7. Forming the list and an estimation of minuses?
Q8. Forming the list and an estimation of interests?
Q9. Loading of the module of gathering and calculation of estimations?
Q10. Calculation of the general estimation?
Q11. Analyzing result and decision-making?
Q12. Creation documents

Algorithm 2: QA-model of the technique.

QJ.1. What is the first plus from the decision?
AJ.1. Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.
QJ.1.1. What importance of the first plus (+1 till +10)?
AJ.1.1. +6.
QJ.2. What is the second plus from the decision?
AJ.2. Yyyyyyyyyyyyyyyyyyyyyyyyyyy.
QJ.2.1. What importance of the second plus?
AJ.2.1. +8.
QK.1. What is the first minus of decision (−1 till −10)?
AK.1. −5.

Algorithm 3: Fragment of the estimation results.

Table 1: Table structure for PMI technique.

Positive		Negative		Interesting	
Plus	Value	Minus	Value	Interest	Value

the possibility for tracing the way of executing the technique with any executable structure.

Definite documents are used in the DM tasks and in presented PMI task also. For working with documents the QA-processor allows using the special class of service tasks. Each task of such class also uses the QA-template and the corresponding QA-model. Such possibility is reflected in Figure 3 as "*Documenting view*."

For the PMI task its base document has the table structure (see Table 1).

The document table structure is being transformed to the corresponding QA-protocol the potential fragment of which is presented in Algorithm 3.

Externally simple structure of the two QA-models hides the rich possibilities of their (useful for designing) transforming, processing, and visualizing which are caused by the system of their architectural views. Moreover the PMI task and the corresponding task of its documenting are included into the task tree of designing the SIS and all tasks of this tree have similar QA-models and are controlled by unified rules.

Different kinds of relations between tasks are used in the task tree of designing the SIS. Relations being possible for the DM tasks are presented in Figure 14 where tasks of the definite type are marked by labels.

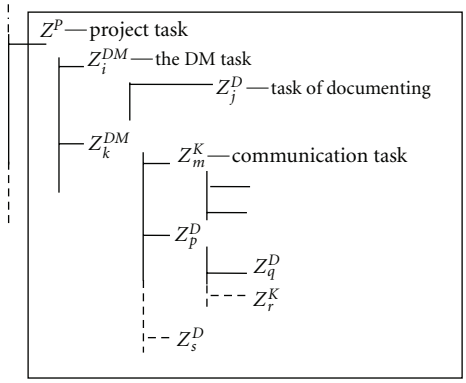

Figure 14: Fragment of the task tree reflecting the relations between tasks for decision making.

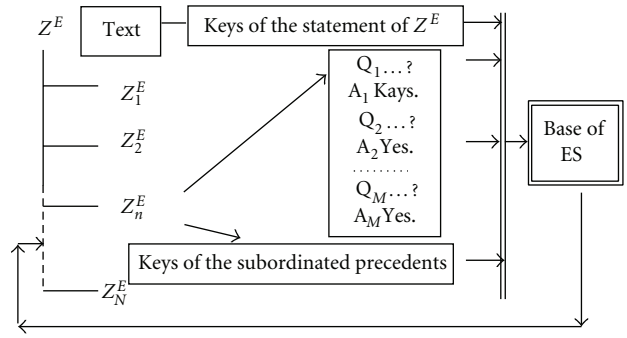

Figure 15: The work with keys of choice.

In accordance with circumstances the developer can open for solving:

(i) the DM task Z_i^{DM} subordinated to the project task Z^P;

(ii) the task of documenting Z_j^D subordinated to the DM task Z_i^{DM};

(iii) the communication task Z_m^K subordinated to the DM task Z_k^{DM};

(iv) the task of documenting Z_q^D subordinated other task of documenting Z_p^D;

(v) the communication task Z_r^K subordinated to the task of documenting Z_p^{DM}.

Let us notice that communicative tasks in the WIQA.Net support "*communicative view*" by the specialized e-mail, meeting, estimation and argumentation forms of the interaction between its users. In the ES its communicative tasks are used for supporting the substantiation and estimation functions.

The developer can include to the any point of the task tree any task of any type if it is necessary for him. The necessity of decision-making can be requested in any time of designing the SIS. If so, the appropriate method of decision-making as a special task will be loaded in the task tree of designing the SIS.

If it is be useful for the developer he can include to the needed point of the task tree the suitable communication task supported by the QA-processor (special e-mail, the task of collective estimating, the task of argumentation or the brainstorming task). Estimations into the DM tasks are fulfilled with the help of the communication tasks. For all tasks of communication the QA-templates are created and loaded in the common library of QA-templates. A set of communication tasks supports "*communicative view*" of the QA-model.

Presented approach to QA-programming the PMI technique was used for QA-programming a set of decision-making techniques which includes force-field analysis, paired-comparison analysis, impact analysis, GRID analysis, and SWOT analysis. All typical QA-programs of named techniques are combined and included to the special section of the library with QA-templates. All QA-models of the corresponding documents are included to this library also. For working with the DM tasks in the frame of designing the SIS the special plug-in of the QA-processor was developed.

10. Expert Decision-Making

Into the complex of QA-means for decision-making the "expert system" (ES) is included. Such system is implemented on the base of the plug-ins of the QA-processor for supporting "*experience view*" of any QA-model.

"The knowledge base" of the ES gives the possibility to keep precedents of two types - declarative and procedural precedents each of which includes production rule combined with the QA-model of this rule. The fact of storage together with precedent of its QA-model opens additional (to known) means for checking the adequacy of precedent and specification of its version for using the precedent in a situation of a choice. Such QA-model can be interpreted as a DM task positive solution of which confirms the relevance of the corresponding precedent. Moreover, a set of keys can be appointed to the result of solving the DM task for potential working with precedents composing other precedents.

For this purpose with any potential precedent (entering into a number of precedents selected on a set of keys of for situation of choice for the task Z^E), it is necessary to connect subordinated task of choice Z_i^E. Process of the decision of task Z_i^E consists in confirmation or not for answers A_m on questions Q_m of its model QA (Z_i^E). After positive confirmation of all answers the normative key for this precedent is being added to the set of keys of the expert task Z^E.

In scheme presented in Figure 15 the inclusion of the subordinated tasks connected with the analysis of relevance of precedents, is reflected.

For working with the expert DM tasks in the WIQA.net following sequence of action can be used.

(1) The statement of the project task (e.g., task Z^E in Figure 15) is transformed to the list of keys for accessing the knowledge base.

(2) The list of keys is used for interacting with the expert system and forming the list of potentially relevant precedents.

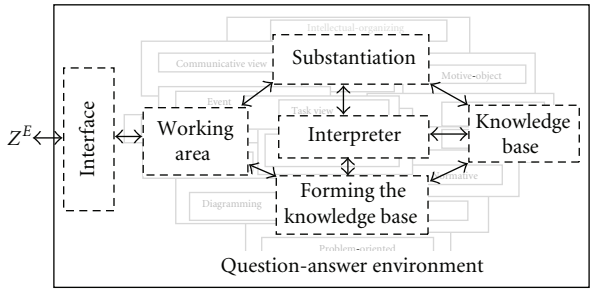

Figure 16: Immersing the ES to SIS^{QA}.

(3) The expert DM task (e.g., task Z_n^E) is created for each potentially relevant precedent and this task is loaded in the task tree.

(4) If the task Z_n^E has a compound type its subordinated task are loaded in the task tree also.

The work with the uploaded task starts with subordinated tasks (if they are defined). If the solution of the subordinated task (e.g., task Z_m^E) confirms relevancy then keys of this task (answer A1) are included to the common list of keys. Changing the list of keys is a cause for the next access to the knowledge base. The process of the expert decision-making is being finished when all loaded DM tasks are solved.

The general rules of working with the expert DM task after its loading to the task tree of designing the SIS is fulfilled as working with the others tasks in this tree. The expert DM task can be interrupted in any time with returning to the solution process in suitable time. The useful service or technological task (any the DM task even) can be subordinated to the expert DM task.

One of traditional components of decision-making with the help of the expert systems is estimation of reliability in the form of answers on questions about attributes used for a choice of precedents, and about each of precedents as a whole. One of approaches to estimations of reliability is the estimation by expert or the estimation by a group of experts. For the estimation, accessible through "*communicative view*" of QA-models, it is enough to include the next task of estimation to the chosen "point" of a task tree.

Another questions which must be solved within the frame of expert systems, is the question about a substantiation of accepted decisions. To the main task for generating the substantiations it is possible to include a subtask of argumentation the typical version of which is included to the set of service tasks.

Adaptation of the expert DM-task to its decision in the SIS^{QA} leads to "*problem-oriented view*" of the QA-model presented in Figure 16. This "*block and line*" view is chosen specially, so that it is corresponded to the typical scheme of the Expert System.

In Figure 16 the structure of the ES is presented against background of the QA-model to emphasize functional character of immersing the ES to the SIS^{QA}. Factually all functions of the ES are being programmed (are being QA-programmed) with using means of the processor WIQA.Net.

```
Z10. Precedent 10.
Q1. System is not uploading?
A1. Yes.
Q2. Is a sound signal about disrepair?
A2. Yes.
Q3. Sound signal corresponds to the *****?
A3. Yes.
Q4. What is a reason of the disrepair?
R4. Processor is broken.
```

Algorithm 4: QA-program of confirmation.

```
Z10.1. Precedent 46
Q1. What is a reason of the disrepair?
A1. The skew of the adapter plate
Q2. How it can be repaired?
A2. At the switched off computer take out the adapter
from the slot and then insert adapter in the slot.
```

Algorithm 5: QA-program of reaction.

The fact of uniform programming and reprogramming the base functions of the ES is the sufficient basis to recognize the version of the SIS^{QA} adapted to the expert DM-tasks as a new kind of expert systems.

Moreover, each of the functions of the ES is being activated in accordance with the plan (or situationally) with using the uniform interface of the QA-model. In other words, through the interface of the QA-model the user can get access to any component of the ES shown in Figure 16, including the interface of the ES. It can be understood that the interface is distributed within structure of the ES.

The high level of QA-programming the task Z^E is beingimplemented as its describing through subordinated tasks till the required depth. One group of subordinated tasks includes service tasks which support the decision-making process. Other subordinated tasks $\{Z_i^E\}$ are similar to the task Z^E because they reflect the work with subordinated precedents. The typical set of such subordinated tasks include "documenting task", "adjusting task", "confirmation task", "communicative task" and "reacting task".

Adjusting tasks support the searching, testing and adopting the appropriate precedents. QA-programs of adjusting tasks reflect the logical part of "activity reflexes". Executions of such QA-programs help to adjust the chosen precedents to their using for decision-making the task of Z^E-type.

QA-programming of the confirmation task and reacting task will be demonstrated for the ES which helps the user in the work with the disrepair of the computer. In the developed ES its knowledge base includes the following precedent "If (system is not being uploaded) and (there is a sound signal about disrepair) and (the form of sound signal is ****) then (processor is broken)".

For confirmation the user must confirm the set of questions presented in Algorithm 4.

In the knowledge base there are twelve subordinated precedents which correspond to the confirmation. One of

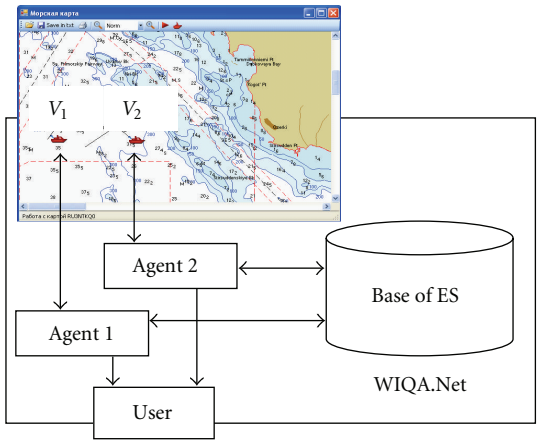

Figure 17: System for expert monitoring.

the potential reactions for the confirmed condition (reacting task) is presented as QA-program in Algorithm 5.

Described means were used for solving the task of "expert monitoring of vessel surrounding" also. The general scheme of the developed system is presented in Figure 17.

Each vessel in the task is presented as an agent. All agents live in the common subject area which is materialized in QA-base as the working area of the ES. States of the working area reflects dynamically the movement of vessels which must be subordinated to normative rules described in a guide [19] and coded in the knowledge base (*experience view*).

Any agent predicts the own trajectory and location on the definite time ahead. Such information from agents is used for building the predictable state of the working area. This state is accessible to the user of the ES and to agents. Any state is checked in the frame of "the collision avoidance rules". Each action of any agent is registered in its QA-protocol which is interpreted as a QA-program of the specialized type (similar "black box").

In the described ES the cartography subsystem and means of the agent representation have been included additionally to the base means of WIQA.Net. The radar station and its imitator are used as base sources of cartography data about the vessel surrounding.

11. Conclusion

This paper presents the system of means for question-answer interaction in collaborative development activity. The content of such activity is connected with conceptual designing the SIS the successfulness of which is extremely low. In order to increase it the AI means can be used and first of all means for using the integration of intellectual resources for solving the complex tasks must be experienced.

For such aim the system of QA-means similar to the CDE-system is suggested. This system (named WIQA) is investigated and implemented as the QA-processor in a number of versions. The QA-processor supports creating the QA-programs for the system of design tasks and their useful investigation as special kind of conceptual models.

The QA-programming is a technique of conceptual solving the task. This technique helps the designer to build the decision of tasks from the units of such types as "question" and "answer with the help of actions with the experience and/or their models. The QA-programming gives the possibility to program the human-computer interaction in real time with task being solved. Step by step the designer or their group creates new "points" of interaction on the visualized model of the task promoting the solution of the task to the final result.

The QA-program of any task is its conceptual decision representing the combination of "questions" and "answers" which are created and/or selected (and are adapted if necessary) in acts of interaction with accessible experience and/or accessible models of experience according to the specificity of the task.

The QA-model of the task is a conceptual model the functions of which are being performed by the QA-program in its useful investigation in forms of QA-modeling.

The method of conceptual decision is developed for QA-programming the main task of the conceptual design. Means of the method are organized as a set of workflows called "interaction with experience". Such means can be open for stakeholders in Internet through the defended Web-access.

Methods and means offered, investigated and implemented by the author open some new possibilities for real time integrating the intellectual activities of designers. The number of positive effects also includes: effective monitoring of a QA-process; analysis of opportunities of parallel coordination of work in the design team (with the purpose of distribution of work between designers); demonstration (at a suitable speed) of solution events (at the certain time interval of automated design); demonstration of the current condition of automated design (at the certain time); training the typical decisions (CAD samples) and developing design skills; as well as personification of events for subsequent definition of authorship and contribution of members of the design team.

Proposed means have confirmed the practical usefulness in development of a number of the SIS, including "automated system for planning of cargo transportation", "expert monitoring of vessel surrounding," and "automated system for management of distance education".

References

[1] "Software intensive systems in the future," Final Report, IDATE, Montpellier, France, January 2006, http://www.itea2.org/attachments/150/ITEA_SIS_in__the_future__Final_Report.pdf.

[2] R. N. Charette, "Why software fails," *IEEE Spectrum*, vol. 42, no. 9, pp. 36–43, 2005.

[3] The Standish Group, http://www.standishgroup.com.

[4] G. Booch and A. W. Brown, "Collaborative Development Environments," http://www.booch.com/architecture/blog/artifacts/CDE.pdf.

[5] P. Kroll and Ph. Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Addison-Wesley, Reading, Mass, USA, 2003.

[6] L. Bass, J. Ivers, M. Klein, and P. Merson, "Reasoning frameworks," Tech. Rep. CMU/SEI-2005-TR-007, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa, USA, July 2005.

[7] P. Sosnin, "Question-answer means for collaborative development of software intensive systems," in *Complex Systems Concurrent Engineering, Part 3*, pp. 151–158, Springer, London, UK, 2007.

[8] C. Potts, K. Takahashi, and A. I. Antón, "Inquiry-based requirements analysis," *IEEE Software*, vol. 11, no. 2, pp. 21–32, 1994.

[9] R. Reiff, W. Harwood, and T. Phillipson, "A scientific method based upon research scientists' conceptions of scientific inquiry," in *Proceedings of the Annual International Conference of the Association for the Education of Teachers in Science*, pp. 546–556, Charlotte, NC, USA, January 2002.

[10] D. J. Rosen, "How to Make Inquiry Maps," 2008, http://alri.org/pubs/im3.html..

[11] S. Henninger, "Tool support for experience-based software development methodologies," *Advances in Computers*, vol. 59, pp. 29–82, 2003.

[12] C. Rich and Y. A. Feldman, "Seven layers of knowledge representation and reasoning in support of software development," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 451–469, 1992.

[13] M. H. Lee, "Model-based reasoning: a principled approach for software engineering," *Software—Concepts and Tools*, vol. 19, no. 4, pp. 179–189, 2000.

[14] J. Burger, C. Cardie, V. Chaudhri, et al., "Issues, tasks and program structures to roadmap research in question & answering (Q&A)," Tech. Rep., NIST, Gaithersburg, Md, USA, 2001.

[15] L. Hirschman and R. Gaizauskas, "Natural language question answering: the view from here," *Natural Language Engineering*, vol. 7, no. 4, pp. 275–300, 2001.

[16] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Unified Approach*, Addison-Wesley, Reading, Mass, USA, 1999.

[17] P. Sosnin, "Question-answer processor for cooperative work in human-computer environment," in *Proceedings of the 2nd International IEEE Conference on Intelligent Systems (IS '04)*, vol. 2, pp. 452–456, Varna, Bulgaria, June 2004.

[18] Mind Tools, "Decision Making Techniques and Decision Making Skills," http://www.mindtools.com/pages/main/newMN_TED.htm.

[19] A. N. Cockcroft and J. N. F. Lameijer, *A Guide to the Collision Avoidance Rules*, Elsevier, Amsterdam, The Netherlands, 2003.