We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

4

Question-Answer Shell for Personal Expert Systems

Petr Sosnin
Ulyanovsk State Technical University, Russia

1. Introduction

In the near future a ubiquitous computerization of all spheres of the modern human activity, including various forms of the collective activity, will lead to conditions of a life when all population of the Earth will be involved in interactions with computers. Therefore, in usages of computers by the person it is necessary to aspire to a naturalness of such attitudes. The naturalness should be achieved in that sense that any usage of a computer should be embedded in the activity in accordance with its essence.

Any activity is a naturally-artificial process created on the base of a definite set of precedents the samples of which are extracted from the appropriate experience and its models. Such role of precedents is explained with the help of the following definition: "précédents are actions or decisions that have already happened in the past and which can be referred to and justified as an example that can be followed when the similar situation arises" (Precedent, 2011).

Accessible samples of precedents are necessary means for the activity but in a general case such means can be insufficiently. If absent means will be found and the necessary activity will be created then the new sample of precedent can be built for the reuse of this activity. Hence, told above entitles to assert that "the creation and reuse of precedents defines the essence of the human activity."

Each unit of the fulfilled activity must be modeled by the useful way, be investigated and be coded for its reuse as the precedent. In the life all these actions are similar to creating the programs for the building of which a natural language in its algorithmic usage is applied. Moreover such programs as behavioral schemes are built for tasks which have been solved for already created units of the activity. So, any sample of the precedent can be understood as a program which is coded previously at the natural language (in its algorithmic usage) for the task aimed at the creation of the definite activity unit.

Such understanding of precedents samples allows assert, that any person is solving continuously tasks, programming them in a natural language because the human life is based on precedents. Any person has an experience of programming in a natural language in its algorithmic usage. Let's name such possibility of programming as "a natural programming of a human" (N-programming). Any human has a personal ability of the N-programming the experience of which depends on a set of precedents which have been mastered by the person in the own life.

One can count any human as an expert who owns the valuable information about personal precedents. Such information can be extracted from the human by the same human and can be used for creating the knowledge base of an expert system built by the human for the own usage. In the described case one can speak about the definite type of expert systems which will be named below as personal expert systems (or shortly be denoted as ES^P).

The definite ES^P should be created by the person who fulfills roles of the expert, developer and user of such computer assistant. Such type of expert systems should have the knowledge base containing the accumulated personal experience based on precedents. To create the own personal expert system the human should be provided simple, effective and powerful instrumental means. The Question-Answer shell (QA-shell) which is described in this chapter is a system of such means. QA-shell is built on the base of the instrumental system WIQA (Working In Questions and Answers) previously developed for conceptual designing of software intensive systems.

A very important specificity of QA-shell and ES^P is a pseudo-programming (P-programming) which is used for the creation of precedents samples and also for the work with them in the real time. The language L^{PP} of the P-programming is similar to the natural language in its algorithmic usage. Therefore the P-programming is similar to the N-programming and such similarity essentially simplifies its application in the creation of precedents samples and their use. This specificity takes into account the ordinary human who have decided to use the computer for solving own tasks based on precedents.

The next important specificity is connected with executors of P-programs. There was a time when computers have not been existed and when N-programs of precedents were being executed by certain persons (by intellectual processors or shortly by I-processors). Computer programs (or shortly K-programs) are being executed by computer processors (or shortly K-processors). Any P-program in the ES^P is being executed by I-processor and K-processor collaboratively.

The last important specificity is the "material" which is used by the human for writing data and operators of the P-programs on its "surface". This "material" consists of visualized forms for data originally intended for modeling questions and answers in processes of problem-solving. The initial orientation and features of such type of data are being inherited by data and operators of P-programs and for this reason they are declared as P-programs of the QA-type. In further text the abbreviation of QA will use frequently to emphasize the importance of question(s) and answer(s) for the construction(s) labeled by QA.

2. Question-answering and programming in subject area of expert systems

2.1 Logical framework for precedent model

The use of the precedent as a basic unit of the human interaction with own surrounding demands to choose or build adequate patterns for precedents representations. Appropriate patterns should provide the intellectual mastering of precedents and their natural using by the ordinary person.

In accordance with the author opinion the necessary model for the definite precedent can be created on the base of the following logical framework:

Name of precedent P_i:

c h o i c e

while [logical formulae (LF) for motives **M ={M_k}**]
as [LF for aims C = **{C_l}**]
if [LF for precondition U'= {U'$_n$}],
then [plan of reaction (program) **r_q**],
end so [LF for postconditions **U" = {U"$_m$}**]
---
there are alternatives **{$P_j(r_p)$}**.

This framework is a human-oriented scheme the human interaction with which activates the internal logical process on the level of the second signal system in human brains. Such logical processes have a dialog nature and for keeping the naturalness the interaction processes outside brains should keep the dialog form also.

The logical framework is used in ES^P for creating the precedents models and keeping them in the knowledge base. This fact can be used for indicating the difference between the suggested ES^P and known types of ES. It also distinguishes ES^P from systems which use case based reasoning (CBR). Measured similarity between cases and the access to them in the form of "cases recognition" are the other differences between CBR-systems and ES^P.

Let's notice that any ES is a kind of rules-based systems any of which are "software systems that applies the rules and knowledge defined by experts in a particular field to a user's data to solve a problem". Any precedent model can be understood as a rule for its owner and it opens the possibility to define the class of personal expert systems. The shell which is described below helps humans in the creation of expert systems belonged to this class.

2.2 Question-answering in creation and usage of precedents samples

There are three ways for the appearance of the precedent sample. The first way is connected with the intellectual processing of the definite behavior which was happened in the past but was estimated by the human as a potential precedent for its reuse in the future. The second way is the creation of the precedent sample in parallel with the its first performance and the third way is an extraction of the precedent model from another's experience and its models. In any of these cases if the precedent sample is being created as fitting the logical framework and filling it by the appropriate content then the human should solve the retrieval and extraction tasks of the necessary information from useful sources.

Named tasks of the retrieval and extraction should be solved in conditions of the chosen framework and the usage of diverse informational sources including different kinds of texts and reasoning. In the solving of this task the important role is intended for the mental reasoning. Taken into account all told above the question-answering has been chosen by author for retrieval and extraction of informational elements needed in the creation of precedents samples. Question-Answering (or shortly QA) is a type of "an information retrieval in which a direct answer is expected in response to a submitted query, rather than a set of references that may contain the answers"(Question, 2011) .

There were many different QA-methods and QA-systems which have been suggested, investigated and developed in practice of the informational retrieval and extraction (Hirschman, 2001). Possible ways in the evolution of this subject area were marked in the

Roadmap Research (Burger, 2001) which is actual in nowadays. This research has defined the system of concepts, classifications and basic tasks of this subject area.
Applying concepts of the Roadmap Research we can assert that QA-means which are necessary for working with precedents samples should provide the use of "interactive QA" and "advanced reasoning for QA" (Question, 2011). In interactive QA "the questioner might want not only to reformulate the question, but (s)he might want to have a dialogue with the system". The advanced reasoning is used by questioner who „expects answers which are outside the scope of written texts or structured databases" (Question, 2011). Let's remind, that one of informational sources for the creation of precedents samples is mental reasoning in dialog forms.
QA-means are effective and handy instruments not only for the creation of the precedents samples but for their use also. Sequences of questions and answers which had been used in the creation stage of the precedent can be used for the choice of the necessary precedent sample.

2.3 Programming in the work with precedents samples

The important component of logical framework is a reaction plan of the human behavior which should be coded in the precedent sample for the future reuse. Before the appearance of computers and frequently nowadays the ordinary human used and uses the textual forms for registering plans of reactions. If the plan includes conditions and-or cycles then, its text is better to write in pseudo-code language similar to the natural language in its algorithmic use. In this case the reaction plan will have the form of P-program.
The reaction plan in the form of P-program is being created as a technique for solving the major task of the corresponding precedent. The other important task is connected with the search of the suitable sample including its choice in a set of alternatives.
In ES^P both of these tasks should be solved and P-programmed by the human for their reuse in the future with the help of computer by the same human. Hence, a set of effective and handy means should be included to ES^P for writing and fulfilling QA-programs supporting the work of the human with precedents samples.
There is a feature of P-programs oriented on the work of the human with precedents and their samples. As told above any P-program in ES^P is being executed by I-processor and K-processor collaboratively where the role of I-processor is fulfilled by the human. The idea of the human model as I-processor is inherited by the author from a set of publications (Card, 1983; Crystal, 2004) where described the model human processor (MH-processor) as an engineering model of the human performance in solving the different tasks in real time.
The known application of the MH-processor is Executive Process-Interactive Control (EPIC) described detailly in (Kieras, 1997). Means of EPIC support the programming of the human interaction with the computerized system in the specialized pseudo-language Keystrok Level Model (KLM). A set of basic KLM actions includes the following operators: **K -** key press and release (keyboard),**P -** Point the mouse to an object on screen, **B -** button press or release (mouse), **H -** hand from keyboard to mouse or vice versa and others commands. Means of I-processor should support QA-interactions of the human with the precedent reuse process. The major part of such interactions consists of the execution of P-programs embedded to the current precedent sample. The main executor of P-programs is the human who fulfills the role of I-processor.

2.4 Co-ordination of I-processor and K-processor

MH-processor is defined (Card, 1983) as a system of specialized processors which solve the common task collaboratively. One of these processors is a cognitive processor providing mental reasoning the basic form of which is an implicit dialog (question-answer reasoning, QA-reasoning). Let's count that I-processor is similar to MH-processor and includes the cognitive component with its named natural functions.

It is easy to agree that for saving the naturalness the implicit QA-reasoning as a natural form of the cognitive processes inside I-processor should "be translated" and transferred to K-processor as an obvious QA-reasoning. Hence, K-processor should include the embedded QA-processor supporting the work with obvious QA-reasoning (or the work with question and answers). Such combining of processors provide their natural coordination in the collaborative work managed by the human reasoning.

Combining of processors is schematically presented in Fig. 1 which is inherited and adapted from Fig. 1 of the ACM SIGCHI Circulium for Human-Computer Interaction (Hewett, 2002).

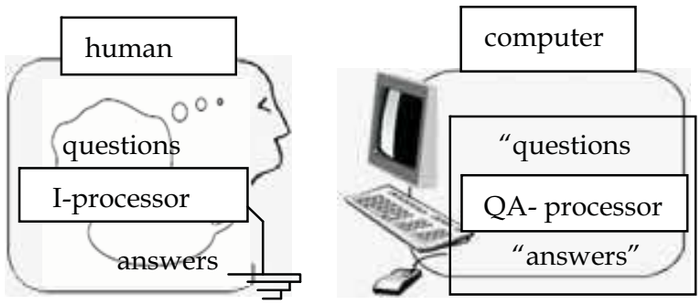


Fig. 1. General question-answer scheme of CHI

In scheme the question is understood by the author as the natural phenomenon which appears at the definite situation when the human interacts with the own experience (own precedents). In this case the „question" is a symbolic (sign) model of the appropriate question. Used understanding helps to explain the necessity of fitting the „question" in QA-processes. Implicit questions and answers exist in reality while „questions" and „answers" present them as sign models.

3. QA-processor and its applications

3.1 Conceptual solution of project tasks

The system named WIQA has been developed previously as QA-processor for the conceptual designing of the Software Intensive System (SIS) by the method of conceptual solving the project tasks.

In most general case the application of a method begins with the first step of QA-analyzing the initial statement of a development task $Z^*(t_0)$. In special cases of its application the initial statement of a task is included in a task tree corresponded to the design technology with which it will be used. The dynamics of the method is presented schematically in Fig.2.

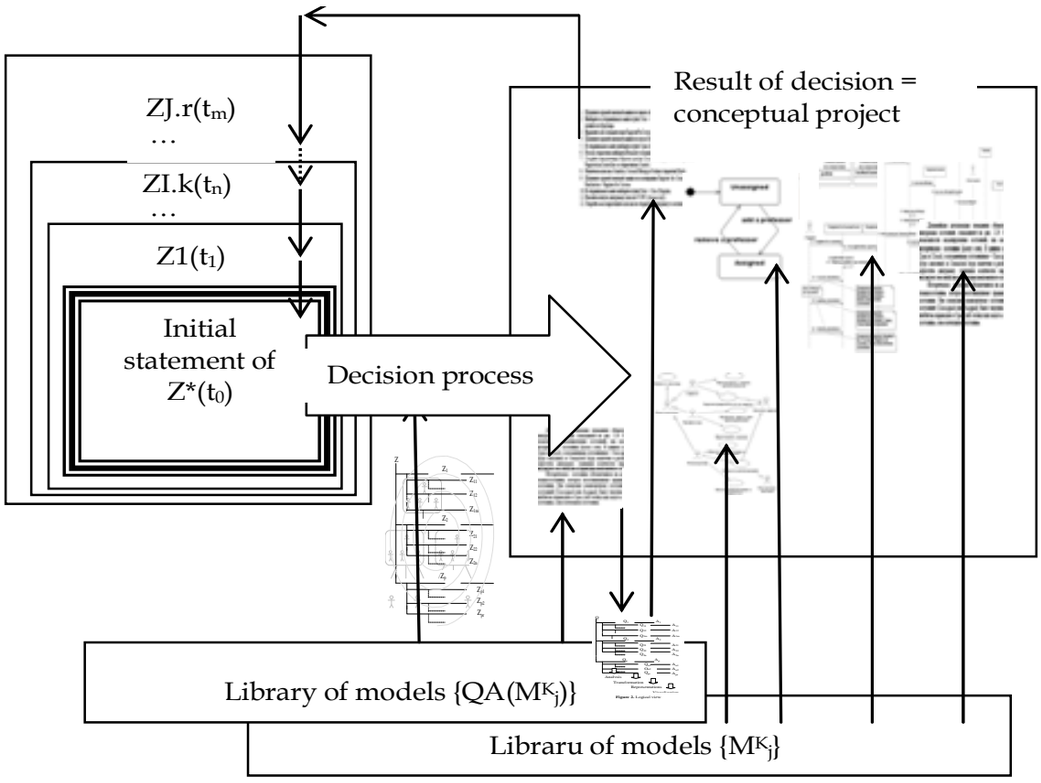


Fig. 2. Dynamics of conceptual solving the project task

The system of tasks of conceptual designing the SIS is being formed and solved according to a method of the stepwise refinement. The initial state of the stepwise refinement is defined by the system of normative tasks of the life cycle of SIS which includes the main project task $Z^*(t_0)$. The base version of normative tasks corresponds to standard ISO/IEC 12207.

The realization of the method begins with the formulation of the main task statement in the form which allows starting the creation of the prime conceptual models. The initial statement of the main task formulates as the text $Z^*(t_0)$ which reflects the essence of the created SIS without details. Details of SIS are being formed with the help of QA-analysis of $Z^*(t_0)$ which evolves the informational content of the designing and includes subordinated project tasks ($Z1(t_1)$, …, $ZI,k(t_n)$, …, $ZJ,r(t_m)$) in the decision of the main task.

The detailed elaboration of SIS forms the system of tasks which includes not only the project tasks connected with the specificity of SIS, but also service tasks, each of which is aimed at the creation of the corresponding conceptual diagram or document. The solutions of project and service tasks are chosen from libraries of normative conceptual models $\{M^k\}$ and service QA-techniques $\{QA(M^k_i)\}$.

During conceptual decision of any task (included in a tasks tree of the SIS project) additional tasks can be discovered and included to the system of tasks as it shown in Fig. 3. The tasks tree is a dynamic system which is evolved iteratively by the group of designers. The step-wise refinement is used by any designer who fulfils QA-analysis and QA-modeling of the each solved task. General conceptual decision integrates all conceptual decision of all tasks included in a tasks tree of the project.

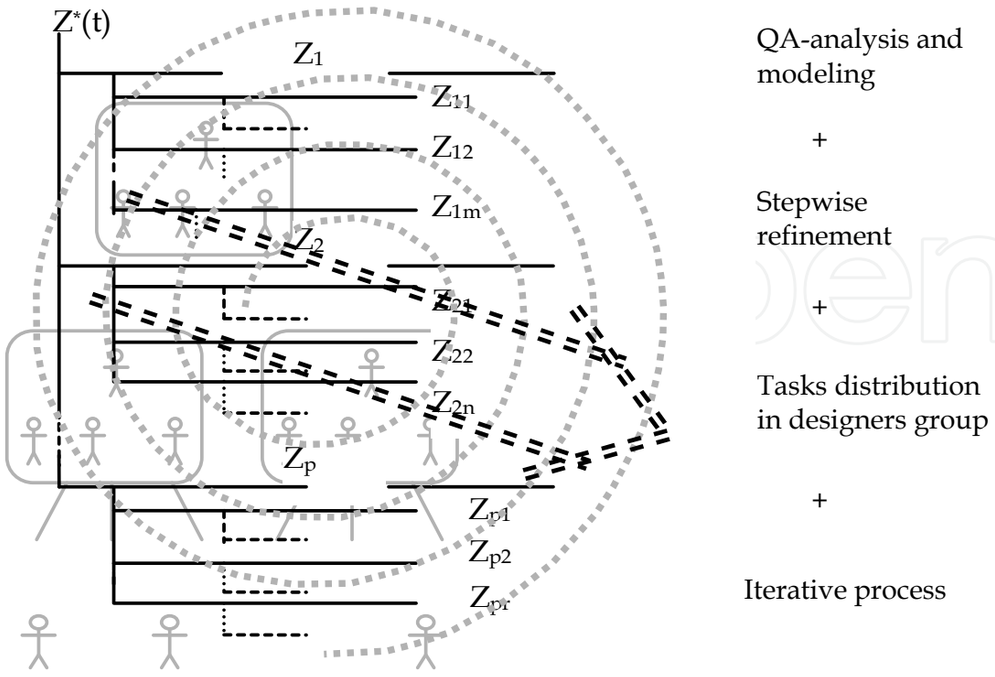


Fig. 3. Task tree of development process

The conceptual solution is estimated as the completed decision if its state is sufficient for the successful work at the subsequent development stages of SIS. The degree of the sufficiency is obviously and implicitly checked. Useful changes are being added for achieving the more adequate conceptual representation of SIS.

Thus, the conceptual solution of the main project task is defined as a system of conceptual diagrams with their accompanied descriptions at the concept language the content of which are sufficient for successful coding of the task solution. Which conceptual diagrams are included to the solution depends on the technology used for developing the SIS.

As a related works which are touched QA-reasoning, we can mention the reasoning in the "inquiry cycle" (Potts, 1994) for working with requirements, "inquiry wheel" (Reiff, 2002) for scientific decisions and "inquiry map" (Rosen, 2008) used for the education aims. Similar ideas are used in the special question-answer system which supports the development of SIS (Henninger, 2003). The typical schemes of reasoning for SIS development are presented in (Bass, 2005), in (Yang, 2003) reasoning is presented on seven levels of its application together with the used knowledge and in (Lee, 2000) model-based reasoning is presented as useful means for the software engineering.

3.2 Question-answering in WIQA

The conceptual solution of any project task is based on QA-analysis and QA-modeling. QA-analysis provides the extraction of questions from the task statement and searching and formulating the answers on them. QA-modeling helps to combine questions and answers in QA-model of the task and its parts and for checking them on the correctness and conformity.

Named QA-actions are fulfilled by designer who translates internal QA-reasoning and registers them in QA-database of WIQA. All these works are implemented with using the visual forms presented in Fig. 4. This form fulfils the role of an inter-mediator between I-processor and QA-processor. The language of WIQA is Russian therefore fields of the screenshot are marked by labels.

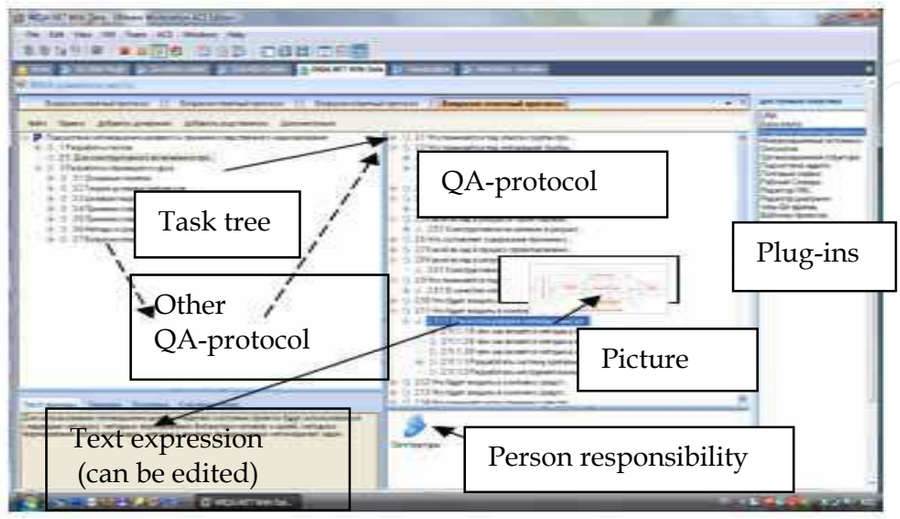


Fig. 4. The main form of QA-processor.

The responsibility for evolving the tasks tree, defining tasks statements and building for them adequate QA-models is laid on designers. For this work they use any informational sources not only mental reasoning. One of these sources is a current content of tasks tree and the current state of QA-model for each task. Therefore a set of commands are accessible to designers for interactions with tasks, questions and answers which are visualized in the main form. The additional commands are accessible via plug-ins of WIQA.

The usage of QA-model of task is a specificity of WIQA as a Question-Answering system. Any QA-model is being formed as an example of QA-sample which is defined as a set of architectural views on the materialization of the model. This set includes, for example, the task view, logical-linguistic view, ontological view and views of other types each of which is being opened for designers with the help of specialized plug-ins.

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 the interaction with the model. Any designer can get any programmed positive effect with the help of the access to the "answer" on the chosen question actually or potentially included in the appropriate view of QA-model (Fig. 5).

The definite set of questions and answers are available to the designer via visual "side" of QA-model named as QA-protocol the structure of which is presented in Fig. 6.

The field of QA-protocol is marked in the screenshot presented above. The designer can use any visual task for the access to the corresponding QA-protocol. Further the designer can use any question Q_i or answer A_j for the access to the content of the corresponding QA-model. One can interprets labels of Z-, Q- and A-elements at the main interface form as visual addresses of corresponding Z-, Q- and A-objects.

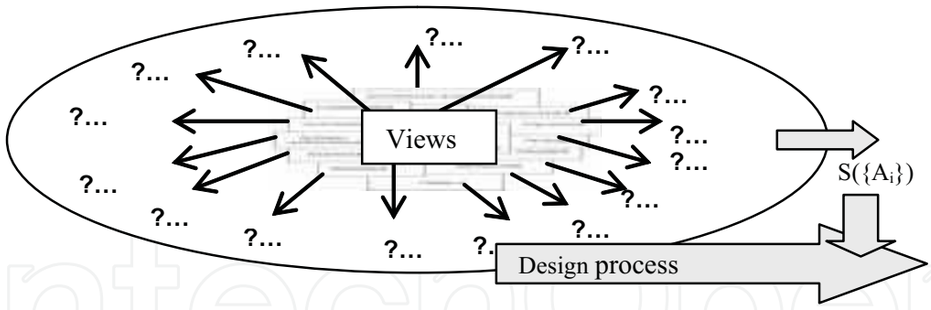


Fig. 5. QA-model of the task

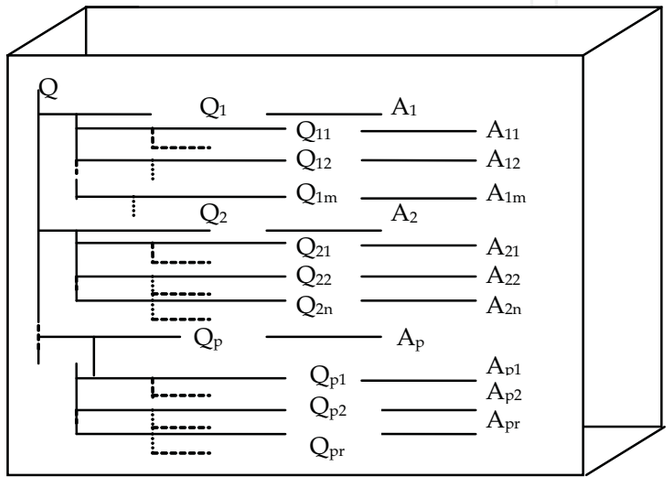


Fig. 6. QA-protocol of QA-model

Any label has a unique code which includes a capital letter (Z, Q, A, or other) and its index appointed automatically. Any capital letter is presented by the icon and indicates the type or subtype of the visualized object. In WIQA there are means for creating the new icons.
The content of such interactive objects are not limited only their textual and graphical expressions which are accessible to the designer via the main interface form. Other "sides" of any QA-model and any interactive object of Z- or Q- or A-type are accessible via plug-ins of WIQA.

3.3 Applications of WIQA

QA processor WIQA has been implemented in several versions. Elaborations of two last versions were based on architectural views of QA-model and the usage of repository, MVC, client-server and interpreter architectural styles. Moreover in created 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 3.5.
The structure of WIQA, its functional possibilities and positive effects are described in a set of publications of the author. The features of WIQA are reflected by its general components structure presented in Fig 7 on the background of QA-model to emphasize that components are working with the common QA-database.

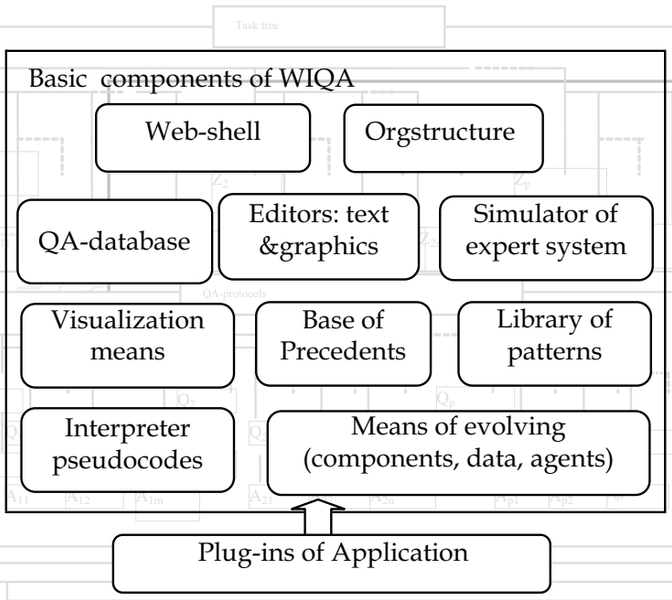


Fig. 7. Components structure of WIQA

As told above WIQA has been created for designing the SIS. The practice of this activity has shown that WIQA can be used as a shell for the creation of some applications. By present time on the basis of this shell, for example, the following applications have been elaborated: DocWIQA for the creation and manage of *living documents*, EduWIQA for the automated teaching, TechWIQA for technological preparation for production and EmWIQA for the expert monitorng of the sea vessel surrounding.

The last application of WIQA is QA-shell for personal expert systems which is being described in this chapter. This QA-shell inherits basic means of WIQA and evolves them by necessary plug-ins supporting the activity based on precedents. Some inheritances were described above and consequently some features of ES^P are already presented.

4. Elaboration of expert system on the base of WIQA

4.1 Question-answer modeling the basic tasks of expert system

The description of ES^P will be continued in the form of its elaboration in WIQA with the inheritance basic means of WIQA, and also their necessary modifying and evolving. First question is about QA-modeling the typical tasks of ES without their orientation to ES^P. The answer this question is connected with immersing the ES into WIQA which is schematically presented in Fig. 8.

The "Block and line" view in Fig 8 is chosen specially, so that it corresponds to the typical scheme of the ES. The structure of the ES is presented on the background of QA-model and also as early for emphasizing the functional style of immersing the ES to its model of QA-type.

The corresponding task should be defined and programmed for each block of ES in its chosen immersing. The tasks structure and the definition of each necessary task can be presented in WIQA in the form of the tasks tree. Each task of this tree can be solved conceptually by the step-wise refinement method. After that each built solution should be distributed between I-processor and QA-processor and necessary computer components should be programmed. In such approach to the elaboration of ES one can assert that possibilities of WIQA means are used for the emulation of ES in WIQA as into the instrumental shell.

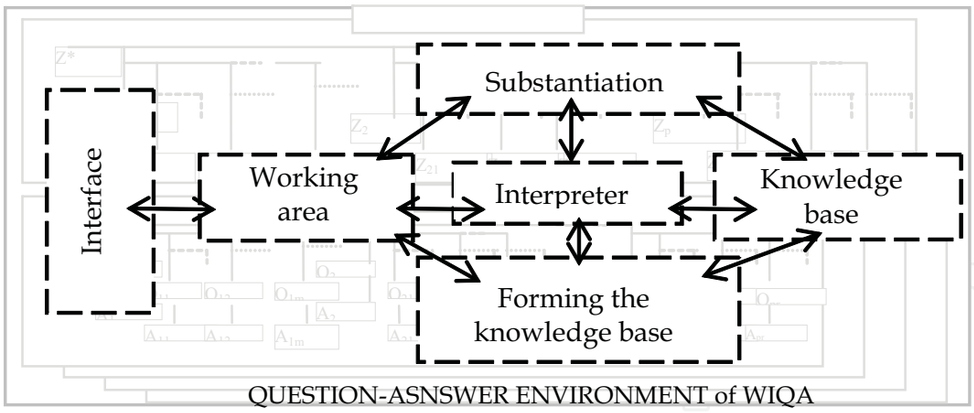


Fig. 8. Emulation of ES in WIQA

First all works named above have been fulfilled for the specialized ES with knowledge base oriented on its filling by samples of precedents extracted from international rules for collision avoidance at sea (COLREG-72) (Cockcroft, 2003). After that the work was repeated creatively and QA-shell for ES^P has been elaborated. Thus the elaboration of the own ES^P is implemented as creating the SIS of the ES^P type.

The usage of Question-Answering is the main specificity of both elaborations which opens for the human the right QA-access not only to the knowledge base (precedents base). The human has the direct access to any task of the tasks tree of ES or ES^P and therefore to any QA-protocol or QA-model in any its state. The human can use such uniform access for the analysis of solution processes in any interval of time and for modeling the evolving the events in ES or ES^P.

4.2 Composite structure of precedent samples

The creation of the new precedent sample P_i is a specially important for the human who elaborates and uses the own ES^P. Such creation is being implemented technologically as the elaboration of SIS also but SIS of the precedent type. This point of view opens the possibility for registering a set of elaboration states in life cycle of precedent (Fig. 9)

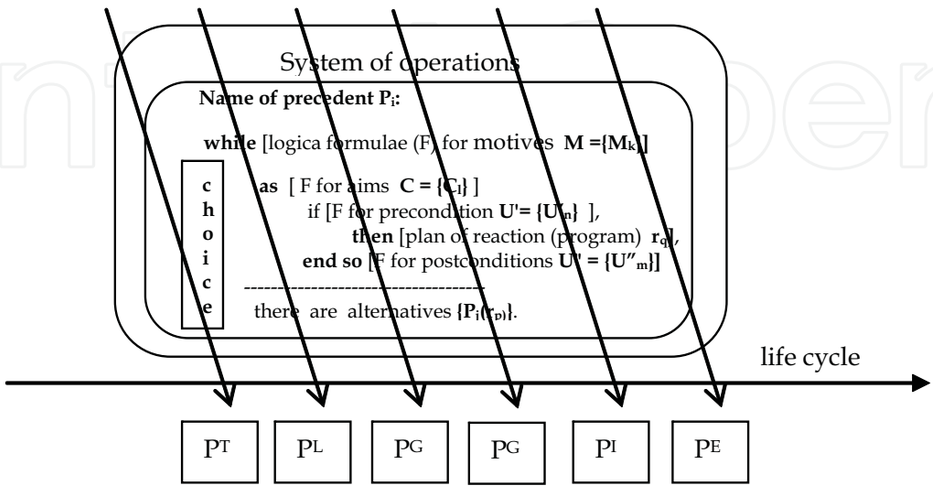


Fig. 9. Presentations of precedent models on the line of its life cycle

This set includes the following useful precedent models: P^T - textual precedent description, P^L - logical (predicate) model, P^G - graphical (diagrammatic) model, P^{QA} - question-answer model, P^I - source program code and P^E - executed code. All of these models are included to the typical materialization of the precedent sample in the knowledge base (precedets base).
The composite structure of the precedent sample and the specificity of its production units were chosen for their usage by I-processor firstly and for the usage by K-processor secondly. The first version of the typical precedent sample which was used for coding the rules of COLREG'72 is presented in Fig. 10. This version is included to QA-shell of ES^P

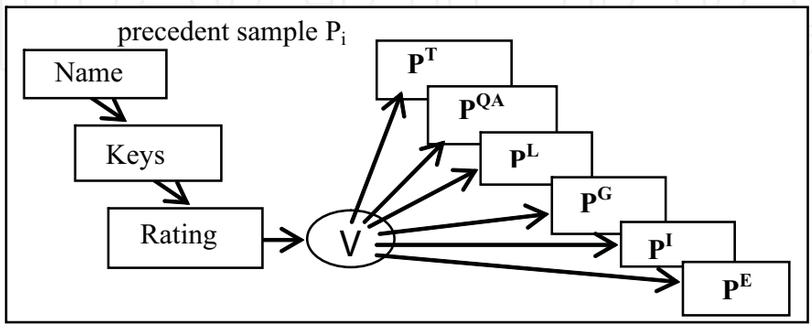


Fig. 10. Structure of the typical precedent sample in the knowledge base of EmWIQA

Precedents used in EmWIQA are accessible as for the user (sailor on duty) so for software agents which are presenting the vessels in the definite sea area. The usage of the automatic access of the vessel agent to the precedents sample in EmWIQA has led the author to the second version of precedents samples which uses P-programming for the work with conditions and reactions in samples of precedents in the form of software agents (Fig. 11).

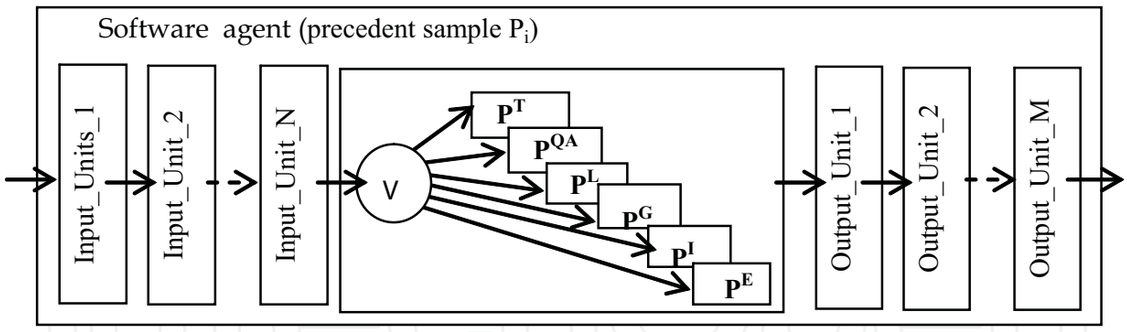


Fig. 11. Precedent sample as a sotware agent

In the second version any precedent sample is presented as an autonomous software unit the access to which is being processed in accordance with conditions of the precedent usage. It is supposed that conditions are defined and described by the person (human) in text form in the natural language (from this point of text we will use the word „person" instead the word „human" to emphasize the context of the personal expert system).
The input text is being processed step by step by a set of input units (morfologic analyzer, ontological filter,key words filter, compiler of condition). If the precedent sample has been chosen and the corresponding precedent has been fulfilled then a set of output units can be activate automated by the person and automatically for registering post-conditions (events on blackboard, output data). The second version is included to QA-shell of ES^P partially.

5. Pseudo-programming in WIQA

5.1 QA-approach to P-programming

The ordinary person in own ES^P should have the possibility for programming the behavior embedded to the precedent sample. As told above the best way for fulfilling such work is the use of P-programming which is supported by handy automated means included to WIQA.

Any P-program is better for understanding as the code of interactions of the person with the corresponding precedent. In WIQA the normative way for interactions is QA-reasoning. Hence is better to adapt the means of QA-reasoning for their use in P-programming. For such adaptation it is necessary to find the ways for emulations (wuith the help of QA-reasoning) data and operators of the appropriate language of P-programming.

Expressions of data and operators of P-programs by means of QA-reasoning is only one part of QA-approach to P-programming. This part should be expanded by the interpreter which transforms any written P-programs in collaborative actions of the person and computer.

Both named parts of QA-approach to P-programming are defined and implemented with their orientation on the ordinary person. To distinguish P-programs of such type from other P-programs they have been named QA-programs.

The type of QA-data has been defined for expressions of data and operators by means of QA- reasoning. Features of this type D will be opened on the example of its simple subtype which consists of a "question" Q_i and appropriate "answer" A_i which haven't the subordinated "questions" and "answers". In this case the "name" and "value" of the definite data D_i are written in attributes of Q_i and A_i which are intended for the textual expression of Q_i and A_i in QA-database. All other attributes Q_i and A_i are inherited by D_i .

The attributes structure of D_i is presented in Fig.12 where not only attributes of QA-database are indicated but additional attributes which are defined by the user also. In general case QA-data are an association of simple data each of which is based on the corresponding pair of Q_i and A_i.

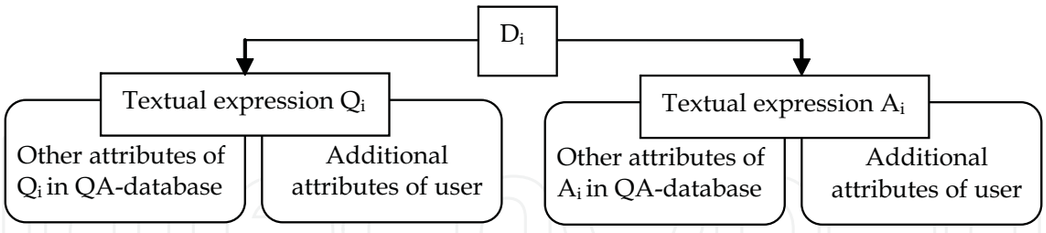


Fig. 12. Attributes stricture of the simple QA-data

Means of additional attributes (AA) are embedded to WIQA for simplifying the elaboration of new plug-ins. The mechanism of AA implements the function of the object-relational mapping of QA-data to programs objects with planned characteristics. One version of such objects is classes in C#. The other version is fitted for pseudo-code programming. The scheme which is used in WIQA for the object-relational mapping is presented in Fig. 13.

The usage of the AA is supported by the specialized plug-ins embedded in WIQA. This plug-ins helps the user to declare the necessary attribute or a group of attributes for definite Z-, Q- and A-elements. In any time the user can view declared attributes for the chosen element. Other actions with the AA must be programmed in C# or in the pseudo-code language supported by WIQA.

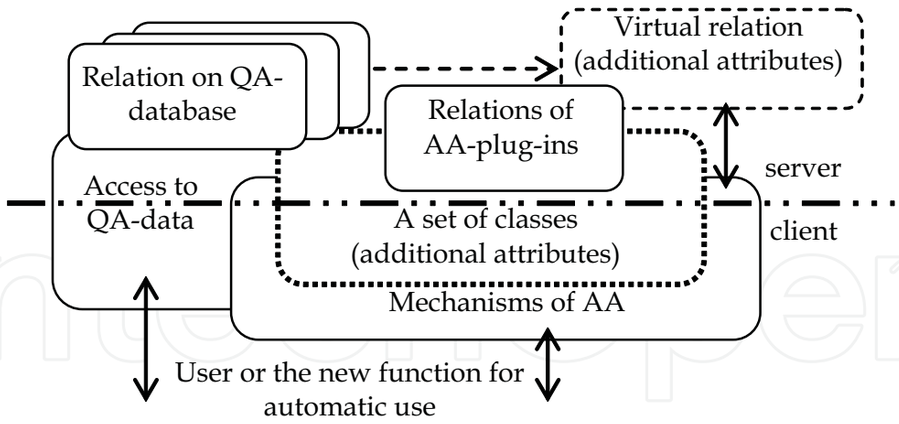


Fig. 13. Creation of additional attributes

Thus in D_i the field for the textual expression of Q_i can be used for writing the declaration of the necessary element of data or operator of P-program. In this case the corresponding field for the textual expression of Ai will be used for coding the "value" of data or the result of the operator execution.

Hence, any line of any P-program is possible to write on the "surface" of the corresponding Q-element which can be interpreted as a "material for writing" with useful properties. This "material" consists of visualized forms for writing the string of symbols. The initial orientation and features of such type of strings are being inherited by data and operators of P-programs and for this reason they are declared as P-programs of QA-type. In order to separate this type of P-programs from P-programs of the others types, they will be named as QA-programs. Such name of P-programs is rightful as the pseudo-code text of any line can be qualified as a "question" on which the interpreter of QA-program builds the corresponding "answer".

5.2 Emulation of pseudo-code data

There are two types of lines of the source pseudo-code one of which intends for the data emulation and another for the operator emulation. Let's begin to describe the emulation of QA-data.

First of all the AA-mechanism was used for the creation a subset of objects imitated the typical data (such as scalars of traditional types, array, record, set and list) in the forms of packed classes (Fig. 14).

For the declaration of variables the constructor of QA-data has been developed. This constructor gives the possibilities to name QA-variable, to choose its type and to appoint the initial value of the variable. The constructor can be used as the self-dependent utility or can be embedded to the translator of pseudo-programs which is implemented as a compiler and an interpreter (in two versions).

Let's remember that any unit of QA-data is created for its use by I-processor firstly and for the computer processor secondly. The visualized declaration of QA-data of the necessary type and the touchable appointment of the necessary visual value take into account the interactions possibilities of I-processor. But any declared QA-variable is accessible automatically for the appropriate programs executed by the computer processor also.

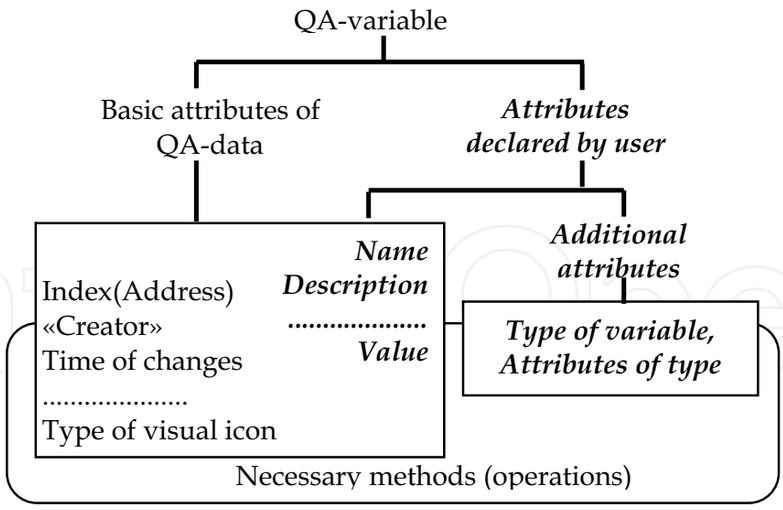


Fig. 14. Imitation of variable

As told above there is a pssibility to create and use the icon for the necessary types or subtypes for Z-, Q- and A-objects. QA-variables can be qualified as a definite type of Q- and A-objects. For this type the icons for letters D and V instead of icons for letters Q and A are created and used.
An example of keeping the array with elements of the integer type is presented in Fig. 8 where a set of additional attributes are used for translating the array declaration to computer codes.

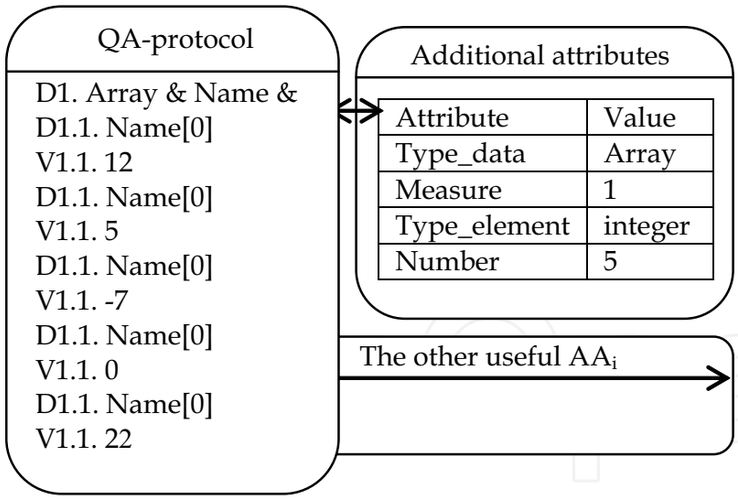


Fig. 15. Declaration of array

Attributes which are assigned for the array are visually accessible for the person at any time and can be used not only for translating. The person can add useful attributes to the set of array attributes for example for describing its semantic features which will be checked in creating and executing QA-program.
Let's open some features of additional attributes for data declarations. For the chosen Q-element the person can appoint not only the definite attribute AA_m but the type T_k of AA_m with characteristics of type T_k and also a set of subordinated attributes $\{AA_{mn}\}$ with the

appropriate type T_n for each of which. All these attributes and types with their values can be used by the person in the creation of QA-programs. Such possibilities help the person in P-programming the work with semantics of QA-variables. The named effects can be used in P-programming the planned or real time work with pseudo-code operators also.

5.3 Emulation of pseudo-code operators

The second type of pseudo-code lines are intended for writing the operators. As it was for QA-data we can define for operators the next interpretations:

- "question" is " a symbolic presentation of an operator";
- "answer" indicates by the special marker about "the fact that the operator was fulfilled".

In other words, the string of symbols for the "question" can be used for writing (in this place) the operator in the pseudo-code form. The fact or the result of the operator execution will be marked or registered in the string of the symbol for the "answer". Such version of emulating the operator has been named as QA-operator. The expression of any QA-operator can be understood as the „question" about the action which is coded. The execution af QA-operator builds the „answer" this „question".

The next step in the emulation of operators is connected with taking into account types of operators. For simulating the basic pseudo-program operators the next constructions were chosen:

- **Appoint**: "question" → "name of variable" and "answer" → "appoint the value;
- **Goto:**"question" → "condition" and "answer" → "go to the definite operator of QA-program;
- **If:** «question» → «condition» **Then** «answer» → «**Execute** the definite operator»;
- **Command**: "question" →" the command of QA-processor" and "answer" → "execute the command";
- **Function**: "question" → "definition of function" and "answer" → "compute the value";
- **Procedure**: "question" → "definition of procedure" and "answer" → "execute the procedure".
- **End**: "question" → "end of program" and "answer" → "finish the work with QA-program".

In named operators the following definitions of functions and procedures are used:

- any function is defined as the expression written in the P-language;
- any procedure is a typical sequence of actions which are accessible in QA-processor for the execution by the person.

The set of basic operators includes traditional pseudo-code operators but each of which inherits the feature of the appropriate QA-unit also. Hence, the basic attributes of QA-unit and necessary additional attributes can be taken into account in processing the operator and not only in its translation. In order to underline the specificity of operators emulation they will be indicated as QA-operators.

In pseudo-programming languages a set of basic operators is being expanded usually. In the described case the expansion includes cycle-operators such as «**for**», "**while-do**" and «**do-until**». Emulations of QA-data and QA-operators are implemented in WIQA and provide the creation of pseudo-code programs for different tasks.

As for QA-variables the special icons for letters „O" (for operator) and „E" (is executed) have been created and used instead icons for letters „Q" and „A". The person can defined

and labeled subtypes of QA-operators. The person can appoint additional attributes for any QA-operator and such attributes can be used obviously in the text of QA-program, for example, for operations with comments included to QA-program lines.

6. Specimens of QA-programs

6.1 Types of QA-programs

Any QA-program creates for the division of the problem-solving process among the person an computer. In this case the division is presented in the form of the source pseudo-code the interactions with which are used as the person so the computer. The definite task of human-computer interactions can be solved with the help of its QA-programming.

But interactions on the base of QA-programs have the additional features. These features are implemented in interactions of persons with Z-, Q- and A-objects which are used for registering the lines of pseudo-code source of QA-programs. As told above such interactive objects open very useful positive effects for persons.

Both named features define the essence of QA-programming for I-processors firstly and for computer processors secondly. The basic aim of the interaction is the access to the person experience in the precedents forms for its inclusion to the problem-solving processes.

The structure of any precedent includes a condition part and a part of a reaction each of which should be QA-programmed. The value "truth" in the estimation of the conditional part opens the access to the execution of the appropriate reaction. Therefore QA-programs for estimating the conditions of precedents and QA-programs for executing the reaction part of precedents are two basic types of QA-programs.

But as told above, some QA-programs can be written for their translating and executing as computer programs. Some of such QA-programs can be created for supporting the work with "precedents" in the definite application. The system of QA-programs was created by author for the collision avoidance expert system of the sea vessel.

QA-programs, which are oriented on the computer execution, are useful in cases when the direct access to the visualized data is profitable for example for developers of SISs or for their users (documenting, decision-making, expert estimating and other tasks). Such programs are suitable when the library of QA-templates (not precedents samples) can be created for a set of typical tasks solving in SISs. The possibility of working with QA-templates and the library of templates are included to WIQA.

For the real time working of I-processor with precedents the following QA-program scheme is useful:

```
QA-PROGRAM_1(condition for the access to the precedent):
D1. Variable V_1 / Comment_1?
V1.Value of V_1.
D2. Variable V_2 / Comment_2?
V2. Value of V_2.
……
DN. Variable V_M / Comment_M?
VN. Value of V_M.
OJ.  F = Logical expression (V_1, V_2, …, V_M)?
AJ. Value of Expression.
End.
```

It is necessary to notice that the person can build or to modify or to fulfill (step by step) the definite example of this program in the real time work with the corresponding precedent which, it may be, the person creates. In presented typical scheme the logical expression is defined for the function F.

The next typical scheme reflects the work with techniques programmed as QA-procedures:

QA-PROGRAM_2 (technique for the typical task):
P1.K_i, K_j, …, PL_k ?
E1. *
P2. K_m, QA-P_n, …, K_q?
E2.*
……
PN. K_s, Pl_t, …, QA-P_v?
EN. #
End.

The program text includes the symbolic names K_x and Pl-y for the Command and Plug-ins of WIQA and QA-P_z for QA-program written by means of WIQA. It is necessary to notice that all names of the types K_x, Pl-y and QA-P_z are indicated positions on the monitor screen for initiating the actions by touch of the person. In this typical scheme the symbols "*" and "#" (as "yes" and "no") indicate the facts of the execution for operators.

The following fragment of the Outlook reset actions demonstrates (without E-units) one type of QA-procedures:

P1. Quit all programs.
P2. **Start** On the menu **Run**, click.
P3. **Open** In the box **regedit**, type, and then **OK** the click.
P4. Move to and select the following key:
HKEY_CURRENT_USER/Software/Microsoft/Office/9.0/Outlook/
P5. In the Name list, **FirstRunDialog** select.
P6. If you want to enable only the **Welcome to Microsoft Outlook** greeting, on the Edit menu **Modify**, click the type **True** in the Value Data box, and then **OK** the click.
P7. If you also want to re-create all sample welcome items, move to and select the following key:
HKEY_CURRENT_USER/Software/Microsoft/Office/9.0/Outlook/Setup
P8. In the **Name** list, select and delete the following keys: **CreateWelcome First-Run**
P9. In the **Confirm Value Delete** dialog box click **Yes** , for each entry.
P.10. On the **Registry** menu, click **Exit**,.
P11. End.

This type provides the work of the person with service techniquea of the definite application. WIQA and QA-shell are examples of such application. About three hundred typical techniques are implemented as QA-programs for designing the SISs with instruments of WIQA. A half of these QA-programs are the guide type. To remember such (or more) quantity of QA-programs are impossile. Therefore all typical QA-programs

are kept in the special library. Any QA-program of this library is kept in the special area of QA-database and registered in its catalog which is visually accessible to the person. Let's notice that the greater part of WIQA techniques are being inherited by QA-shell for ES^P.

If the person needs to use the typical QA-program (needs to solve the typical task with QA-model implemented as QA-program) the person extracts the typical QA-program from the library, creates the new task, includes the task to the tasks tree and after such actions the person can start to solve the task (to execute the corresponding QA-program).

The reality of the person activity is a parallel work with many tasks at the same time. Therefore the special interpreter for executing QA-procedures and the system of interruption are included into WIQA. It gives the possibility to interrupt any QA-procedure (if it is necessary) for working with other QA-programs. The interruption system supports the return to any interrupted QA-program to its point of the interruption.

6.2 Example of QA-functions

As told above WIQA was used for elaboration the application EmWIQA provided the expert monitoring of the sea vessel surrounding. This application uses the base of precedents and means of QA-programming. The behavior of users in EmWIQA can be qualified as the potential behaviour of the person in ES^P. Therefore QA-programs in EmWIQA can be used as examples of QA-programs in ES^P.

One of such QA-programs is QA-function supports the access to the precedent sample which presents the 15th rule of the International Rules for Preventing Collisions at Sea (Cockcroft, 2003):

QA-PROGRAM_3 (conditional access to the precedent).
D1. Velocity V_1 of the power driven vessel V_1?
V1.Value of V_1.
D2. Bear_B_1 of the vessel V_1?
V2.Value of B_1.
D3. Place of the vessel V_1?
V3. Coordinates of the place_1.
D4. Velocity V_2 of the power driven vessel V_2?
V4.Value of V_2.
D5. Bear_B_2 of the vessel V_2?
V5.Value of B_2.
D6. Place of the vessel V_2?
V6. Coordinates of the place_2.
O7.CPA = expression for computing the Closest Point of Approach (CPA)?
E7. Value of CPA.
O8. Cond = (V_1, "keep out of the way")&
& ($|$Bear_1 - Bear_2$|$ > 11, 5^o) &
& (CPA-D^{DA}- $\Delta D_1 \leq 0$)?
E8. Manoeuvre_M_i / Call of the appropriate QA-procedure.
O9. End.

This QA-function is shown with demonstrated aims only and therefore without explaining the variables and expressions. This function is kept in the knowledge base (with embedded

precedents) into the EmWIQA and function is accessible for program agents (automatically) and for the sailor on duty (in the automated regime). The knowledge base of the EmWIQA consists of 155 units each of which includes QA-function for choosing the precedent and QA-procedure for its executing.

7. Means for development and usage of personal expert systems

7.1 Additional means of WIQA

As told above AS-shell of ES^P inherits the basic means of WIQA presented in Fig. 7. These means include the simulator of expert system elaborated previously for EmWIQA, base of precedents with their coding in the first version and the interpreter which uses the means of the dynamic compilation of Microsoft.Net 3.5. After estimation all of these means from the point of view of ES^P the WIQA has been evolved with the orientation on the ordinary person.

The additional technological QA-programs have been added to the specialized system of QA-programs simulating the expert system. The first version of coding the precedent sample is modified by the inclusion to it the possibility of QA-programming the conditional access to the sample (morphologic analysis of key words and compilation of QA-functions). The language of P-programming has been modified by the inclusion to its grammar the description of additional attributes.

Following components have been developed and included in the ES-shell additionally:

- a set of translators (compilers and interpreters) of QA-programs;
- a specialized generator of interface units for helping the person to combine QA-programs and executed codes of other types;
- a set of means for simplifying the work of the person aimed at the creation of precedent samples, their inclusion to the precedents base, access to the necessary sample and its use.

7.2 Translators of QA-programs

Translation means for the pseudo-programming are evolved step by step from one kind of QA-programs to the other kind. Two compilers and two interpreters are embedded in QA-shell for ES^P.

The first compiler provides the processing of QA-programs which describe the conditional parts of precedents. Copies of such compiler can be embedded by the person to the precedent samples implemented as agents. The second compiler supports the translation of QA-programs in the executed codes (.dll-forms).

Both interpreters are intended for I-processors. There are the following differences between interpreters - the first interpreter can work with cycle operators and the second interpreter uses the mechanism of the dynamic compilation for the current line of QA-program which is being executed.

Let's present some details for the first interpreter. As other translators embedded in WIQA this interpreter is worked with the L^P-language. The lexicon of the created QA-program can be chosen by the programmer (by the person). For the declaration of QA-data the specialized utility program is developed. This utility program supports the work with data of traditional algorithmic types. The main window of the interpreter is presented in Fig. 16 with commentary labels.

Interfaces of the main form help to control as executing QA-program so its debugging. The person who is fulfilling the role of I-processor can interrupt I-process on any operator of QA-program with the possibility of returning to the point of the interruption.
In the set of named translators for indicating the types of operators the following variants has been used and checked:

- inclusion the key words into the symbolic presentation of operators;
- selection the type of the operator from the emerging menu;
- appointment the type with the help of additional attributes (as for QA-data).

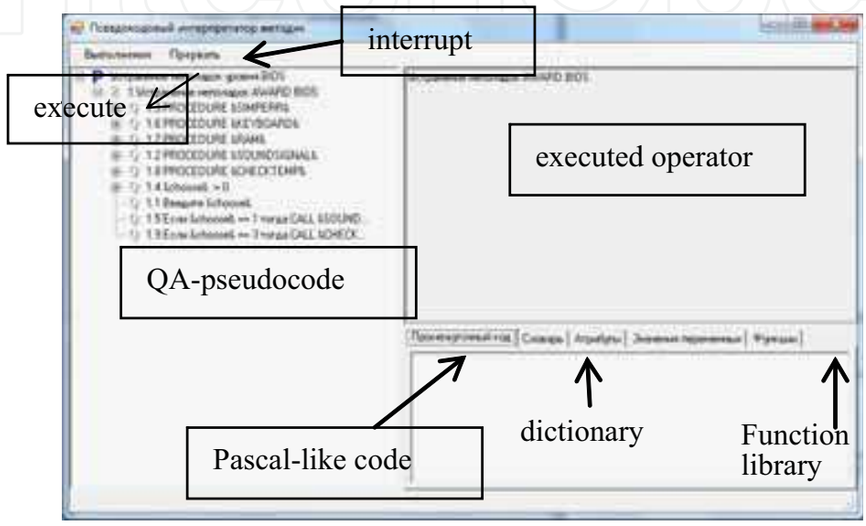


Fig. 16. Screenshot of interpreter

In accordance with told above, the usage of the potential of Z-, Q- and A-objects for emulating the typical data and simulating the basic program operators opens the possibility to create QA-programs which can be translated for their executing by computer processors also.
Pseudo-code texts of QA-programs can be written and executed (in the real time) by the person working in the corporate network. The person interacts with QA-programs as with inter-mediators between the person and computers and it gives the arguments to qualify their as new type of means for human-computer interactions. Moreover, such inter-mediators can be translated (in WIQA) firstly to the C# source code and then to the executed code.

7.3 Generator of interface units

The practice of QA-programming has shown that visual forms of WIQA presented in Fig. 4 are unsufficient for the usability of QA-programs created by the person in ES^P. Therefore the plug-ins „Generator of interface units“ has been created and embedded to QA-shell.
The necessary interface unit is being generated from the drawn interface diagram which is being translated to the scheme of the corresponding QA-program. After that the scheme of QA-program is filling by the chosen interfaces precedents.

Any interface precedent is coded the corresponding metrics of usability. A set of usability metrics includes a subset of metrics which are defined in the standard ISO/ MEK–9126. Other metrics were chosen from other useful sources. Any metrics included to the library are defined as an appropriate task which is solved in QA-shell.

7.4 Creation and usage of precedent sample

Any precedent sample is coded as a composite QA-program the integrity of which is provided by its interface shell. The special plug-ins of WIQA which was named „Elaboration of precedent sample" has been created for writing the codes of sample parts and assembling them as a whole. This plug-ins is similar to the elaboration means of traditional programs but it fits on QA-programming.

The graphic editor embedded to plug-ins helps the person to assemble the current sample by filling its typical graphic form which is a copy of scheme presented in Fig. 11. When assembling is finished the precedent sample is uploaded to the corresponding section of QA-program library.

Any precedent sample is an autonomous software unit which is QA-programmed and can be qualified as the software agent. One of the advantages of the agent of such type is the possibility for its easy reprogramming in the real time.

If a number of precedent samples are necessary for the person who are solving the current task they should be extracted from the precedent base (with using the techniques of ES^P) and uploaded into the active tasks tree.

8. Conclusion

Told above contains sufficient arguments to assert that the described QA-shell helps to create the Expert Systems of the new type. This type of ES is intended for the ordinary person who has decided to create the ES which will be filled by the valuable information about personal precedents. In creation of own ES^P the person fulfills roles of the expert, developer and user of such computer assistant.

The main specificity of the elaborated QA-shell for ES^P defines Question Answering which is fitted to pseudo-programming of precedents samples. Accessible means of Question Answering are coordinated with the dialogue nature of consciousness that simplifies transition from internal reasoning of the person to their models in the computer environment. Therefore the owner of ES^P can apply real time P-programming of I-processor and K-processor for solving own tasks on the base of precedents the samples of which are kept in ES^P.

Accessible means of P-programming is similar to N-programming and their power (types of data, additional attributes and system of P-programming) open the possibility for the ordinary person to write non-trivial programs of the own activity. QA-programs manage accustomed (habitual) semi-automatic actions when QA-programs (as techniques of the guide type) show to the person the sequence of actions which the person must execute. Moreover, QA-programs can be translated in the form which can be executed by the computer processors.

QA-shell is elaborated on the base of the sufficient experince of Question Answering applied to the development of SIS and other applications including applied systems with ES

subsystem based on precedents. For example, QA-samples of precedents were embedded in system for Expert Monitoring of Environment of the Sea Vessel. QA-samples of precedents also have been used in the solution of following tasks: Creation of Interface Prototypes in context of ISO standard 9126; Information Safety of SIS in the context of ISO standard 15408; Predicative Ontological Testing of Project Solutions.

9. References

Bass, L.; Ivers J. & Klein, M. & Merson, P. (2005). *Reasoning Frameworks*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2005-TR-007.

Burger, J. et al. (2001). *Issues, Tasks and Program Structures to Roadmap Research in Question & Answering (Q&A)*, Tech. Rep. NIST.

Card S.K.; Thomas, T.P. & Newell, A. (1983). *The Psychology of Human-Computer Interaction*, London: Lawrence Erbaum Associates.

Cockcroft, A.N. (2003). *Guide to the Collision Avoidance Rules: International Regulations for Preventing Collisions at Sea*, Butterworth-Heinemann, 2003.

Crystal, A. & Ellington, B. (2004). *Task analysis and human-computer interaction: approaches, techniques, and levels of analysis*. In proceedings of the Tenth Americas Conference on Information Systems, New York, New York, pp 1-9.

Henninger, S. (2003). *Tool Support for Experience-Based Software Development Methodologies*, Advances in Computers, vol. 59, pp. 29-82.

Hewett, T.; Baecker , R., Card , St., Carey , T., Gasen , J., Mantei, M., Perlman , G., Strong, G., & Verplank, W. (2002). *ACM SIGCHI Curricula for Human-Computer Interaction*. ACM Technical Report. P. 162.

Hirschman, L. & Gaizauskas, R. (2001). *Natural Language Question Answering: The View from Here*. Natural Language Engineering, vol. 7, pp. 67-87.

Karray, F.; Alemzadeh, M., Saleh, J. A. & Arab, M. N. (2008). *Human-Computer Interaction: Overview on State of the Art* Smart sensing and intelligent systems, vol. 1, No. 1(Mar), pp 138-159, 2008.

Kieras, D. & Meyer , D.E. (1997). *An overview of the EPIC architecture for cognition and performance with application to human-computer interaction*. Human-Computer Interaction, 12, 1997, 391-438.

Lee, M.H. (2000). *Model-Based Reasoning: A Principled Approach for Software Engineering*, Software - Concepts and Tools, vol.19, #4, pp. 179-189.

Potts, C.; Takahashi, A. & Anton, K. (1994) *Inquiry-based Requirements Analysis*, IEEE Software, vol.11, #2, pp. 21-32.

Precedent. Available from http://dictionary.reference.com/browse/precedent.

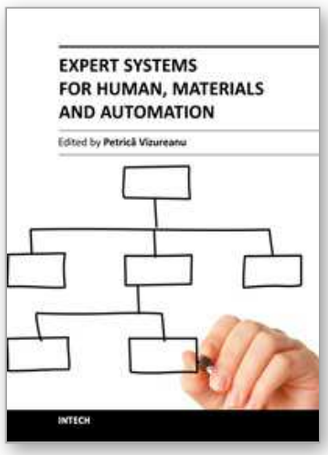
Question-Answering. Available from http://www.wordiq.com/definition/Question_ answering.

Reiff, R.; Harwood, W. & Phillipson, T. *A (*2002*) Scientific Method Based Upon Research Scientists' Conceptions of Scientific Inquiry*, In Proc.2002 Annual International Conference of the Association for the Education of Teachers in Science, pp 546–556.

Rich, C. & Feldman, Y. (1992). *Seven Layers of Knowledge Representation and Reasoning in Support of Software Development*, IEEE Transactions on Software Engineering, vol, 8, # 6, pp.451-469.

Rosen, D. J.; (2008) *How to Make Inquiry Maps*. Available from: http://alri.org/pubs/im3.html.

Yang, F.; Shen, R. & Han, P. (2003). *Adaptive Question and Answering Engine Base on Case Based and Reasoning Technology*, Journal of Computer Engineering, vol.29, #11, pp. 27-28.



Expert Systems for Human, Materials and Automation

Edited by Prof. PetricÄƒ Vizureanu

ISBN 978-953-307-334-7
Hard cover, 392 pages
Publisher InTech
Published online 10, October, 2011
Published in print edition October, 2011

The ability to create intelligent machines has intrigued humans since ancient times, and today with the advent of the computer and 50 years of research into AI programming techniques, the dream of smart machines is becoming a reality. The concept of human-computer interfaces has been undergoing changes over the years. In carrying out the most important tasks is the lack of formalized application methods, mathematical models and advanced computer support. The evolution of biological systems to adapt to their environment has fascinated and challenged scientists to increase their level of understanding of the functional characteristics of such systems. This book has 19 chapters and explain that the expert systems are products of the artificial intelligence, branch of computer science that seeks to develop intelligent programs for human, materials and automation.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Petr Sosnin (2011). Question-Answer Shell for Personal Expert Systems, Expert Systems for Human, Materials and Automation, Prof. PetricÄƒ Vizureanu (Ed.), ISBN: 978-953-307-334-7, InTech, Available from: http://www.intechopen.com/books/expert-systems-for-human-materials-and-automation/question-answer-shell-for-personal-expert-systems

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen