International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 3 (2011) pp. 444-452

Programming of Human-Computer Interactions in Development of Software Intensive Systems

Petr Sosnin

Computer Department, Ulyanovsk State Technical University,
Severny Venetc, Ulyaniovsk 432027, Russia
sosnin@ulstu.ru

***Abstract*: The paper presents a question-answer approach to programming of human-computer interactions (HCI) during a collaborative development of software intensive systems. Efficiency of the general work can be essentially increased if the human part of the work will be fulfilled as an execution of a special kind of programs by "human processors" which use models of question-answer) reasoning. Such approach was investigated and evolved till an instrumental system providing the pseudocode programming of human processors combined with computer processors. Pseudocode programs of the question-answer type are useful means of HCI. Such programs and their corresponding instrumental means can be combined easily with traditional means of HCI.**

***Keywords*:** pseudocode programming, human-computer interaction, question-answer reasoning.

I. Introduction

Reality of HCI as a subject area includes a definite subset of HCI tasks which reflect this discipline (and phenomenon) from the side of collaborative actions of a human and a computer.

Rational human-computer actions have a special importance in processes of the collaborative problems-solving, when "combined" consciousness, mutual understanding and other intellectual human abilities and their computer models are being included obviously and constructively into the coordinated activity.

The integration of named intellectual abilities is especially necessary at designing of software intensive systems (SISs) the complicated tasks of which can be solved by developers only collaboratively with the use of rational reasoning.

It is known that the rational reasoning is a natural form for the inclusion of intellectual (human) actions into the common work and models of reasoning are used as intermediaries between human reasoning and automatic actions of computer assistants.

In this article the question-answer approach to the designing and using of HCI is presented. The approach is aimed at increasing the intellectuality of HCI. We can mark the following features of such approach:

- using the question-answer reasoning (QA-reasoning) and their models in HCI for the rational connection of human and computer actions in their collaborative activity for hierarchical representing the tasks and also for their modeling, analyzing and programming;
- using the QA-reasoning for pseudo-programming of the human "processor" (H-processor), which is executing the human actions, similarly the work being executed by the computer processor (K-processor);
- using the aspect-oriented designing for creation of the interfaces embedded into the software intensive systems;
- using the knowledge database for keeping metrics of usability in the form of precedents for the access to them in the designing process.

As a source of requirements for materializing the named features the theory and practice of Collaborative Development Environments [2] were used. Such type of instrumental systems supports the development of Software Intensive Systems (SISs) during which a group of developers are solving collaboratively the enormous quantity of normative technological tasks and original project tasks. It is necessary to indicate that interests of the article are limited by the human-computer work with tasks at the conceptual designing of the SISs.

II. Related Works

The problem of rational reasoning in the development process of SIS is well known. This problem has been investigated for more than 10 years in the Software Engineering Institute (SEI) of Carnegie Mellon University [1]. But the question-answer approach is not used and the problem of "a real time integration of intellectual efforts" is not indicated in interests of 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) [14] and in other similar technologies, for example, in Microsoft Solution Framework and Eclipse.

It is a 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 and others types of reasoning).

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 situations in designing, first of all case-based reasoning models).

We are convinced that the 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 [23].

In the number of relative works using "questions and answers" (or QA), for example, we can mention reasoning in the "inquiry cycle" [16] for working with requirements and "inquiry wheel" [18] for scientific decisions. Similar ideas are used in the special question-answer system which supports the development of SIS [8]. The typical schemes of reasoning for SIS development are presented in [1]. In paper [19] reasoning is presented on seven levels of its application together with the used knowledge and in [15] model-based reasoning is presented as useful means for the software engineering.

But in all publications referred to above, the issue [3] and the special report [10] the task of the real time integration of intellectual resources in processes of the problem-solving and decision-making is not mentioned.

The specificity of suggested means is schematically presented in Fig. 1 which is inherited and adapted from Fig. 1 of the ACM SIGCHI Curriculum for Human-Computer Interaction [9].

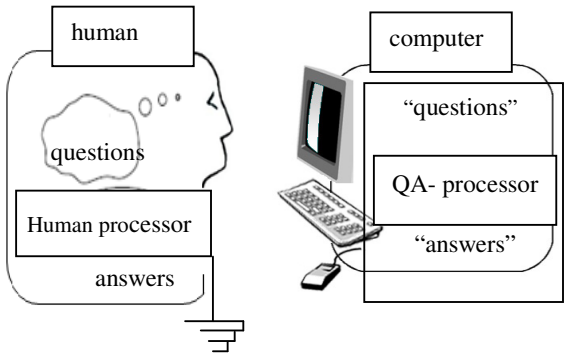

Figure 1. General question-answer scheme of CHI

The phrase "human as a processor" and partially its understanding are inherited also from the publication [9]. The used understanding of the H-processor is mostly aimed at processing of questions and answers in the work with an experience of a human and such understanding has differences with its traditional understanding (for example) applied in the informational source [6] and [17].

It is necessary to note that the suggested means can be classified as intelligent and adaptive means in accordance with the CHI overview [11].

It is necessary to mark a set of publications which are presented the theory and practice of the H-processor information model first introduced by Card, Moran and Newel in [4]. The EPIC version of this model [12] with its KLM means is similar to the H-processor model which is described in this article.

The special search of related publications was fulfilled for keys "programming, human activity". There are many publications with explaining the versions of such relations corresponding with the conjunction "as" (programming as a human activity). Here we can name authors E. Dijkstra – "programming as a human activity" [7] and D. Knuth – "programming as an art" [13].

The other type of rational relations is defined by the conjunction "of". The Internet-search of publications with key words "programming of a human activity" has remained without interesting results. Such type of relations will be defined below and implemented constructively for the processors of the human type.

III. Question-Answer Model of the Task

In accordance with the previous content the interests of this article are limited by rational HCI means when a group of designers must solve enormous quantity different tasks at the conceptual stage of designing the SISs. The HCI scheme is materialized by the author as a specialized instrumental system [20] which has the SIS type. Such client-server system is based on the usage of question-answer reasoning of designers in processes of the problem-solving what was a cause to name the system as WIQA (Working In Questions and Answers).

The main screenshot of WIQA, which demonstrates the possibility of interactions of designers with the current state of the development process of SIS, is presented in the Fig. 2 with commentary labels (because this system was used in real projects only in Russia and it has interfaces in Russian).

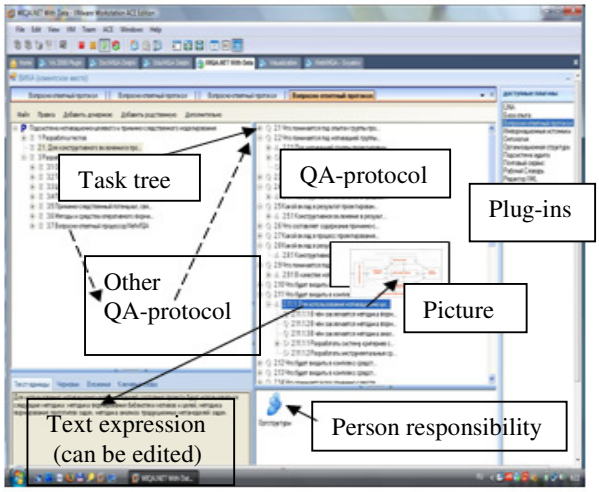

Figure 2. The main interface of WIQA

Any project task of the developing SIS is a unit for finding and registering its current solution in WIQA with the help of the question-answer reasoning. The result of such work is being materialized as a question-answer model (QA-model) of the corresponding task. WIQA is the instrumental system which supports the development process presented as a tasks tree any task of which is existed as its QA-model.

In the screenshot is shown that for the chosen task Z_i of the tasks tree its QA-model is accessible through the Question-Answer protocol (QA-protocol) registering question-answer reasoning, any unit of which (question Q_{ij} or answer A_{ij}) has a textual expression with necessary pictures (for example, with UML-diagrams or "block and line schemes"). Units of the QA-protocol which are accessible for designers on the monitor screen are presented in Fig. 3.

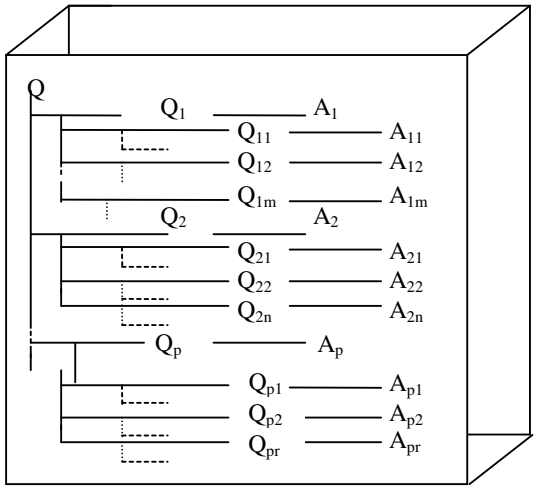

Figure 3. View of QA-model

In more details any unit of the Z-, Q- or A-type is an interactive object the properties of which are being opened when the special plug-ins are used. One of such plug-ins registers and indicates the responsibility (the assignment of the tasks) in the designer group.

The similar form of the tasks tree for the reflection of the development process of SIS is used in the RUP. Moreover, the set of RUP tasks and the generic framework [22] was used by the author [21] as a source of requirements for defining and implementing the normative QA-model of the task which can be adjusted to the definite tasks being solved by designers.

IV. Interactive Potential of QA-Units

A. Specificity of Programming for H-processor

The investigation of a number of SISs in designing of which QA-models of tasks were used, has led the author to the decision "to estimate and evolve the interactive potential of QA-units for its usage in the programming".

If we want to find the way for creating the program for the human processor we must choose firstly the model of the H-processor. In such choosing we must be oriented on the reproduction of reactions which must be similar in the reuse. It is known that the stable reactions of the human on conditions in surrounding are based on precedents as "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" (such definition of the precedent is used in many dictionaries).

Therefore, such units of the experience (as a system of precedents) we have suggested to use as a base for human reactions on data and operators of H-programs. In such solution the analog of the expert system with embedded mechanisms of the case-based reasoning and reusing the chosen precedents will be the useful model of the H-processor.

Any typical unit of the experience base (knowledge base) is implemented as the model of the definite precedent. The model of precedent ("precedent") has the original (productions) structure presented in Fig. 3 where P^T – textual precedent description, P^{QA} – question-answer model, P^L – logical (predicate) model, P^G – graphical model, P^I – source program code and P^E – executing code.

The composite structure of the "precedent" and the specificity of its production units were chosen for their usage by H-processor firstly and for the usage by K-processor secondly. We investigate "precedents" which will have to be programmed for the human activity with the usage of computer instruments.

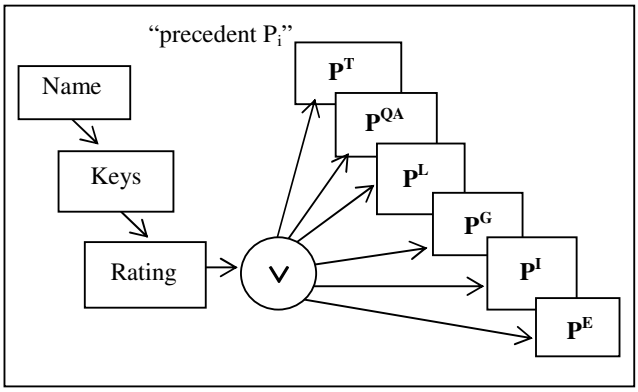

Figure 4. Structure of "precedent"

In programming the H-processor we shall distinguish between precedents as units of the experience and "precedents" as computer models of precedents. It is necessary to notice that there are two variants of mastering the precedents one of which corresponds to the skill (execution with the usage of reasoning) and the second corresponds to the habit (execution with the usage of the automatic access to typical actions).

The next specificity is connected with computer assistants which can help the H-processor during the execution of H-programs based on precedents. Any forms of the computer help in the human work with the needed precedent must be used. There are several variants of such help:

- modeling the precedent as "precedent" or modeling the useful aspects of their existence;
- controlling the real-time actions during the work with the precedent;
- keeping the attention of a human near the definite aspect of the precedent with the help of visualized data and/or operators;
- checking the condition of the fitness for the precedent;
- estimating the adequateness of the chosen precedent;
- adjusting the proper precedent to the new conditions of its usage;
- creating the new precedent and its mastering.

It is necessary to notice that all named variants of the computer help must be programmed in WIQA, first of all, for the work with models of precedents typical scheme of which is shown in Fig.3.

The chosen model of the H-processor and the orientation on the interaction with the traditional expert system are important arguments to the use of QA-reasoning in programming of the H-processor.

There are two additional ways for the adaptation of QA-reasoning to programming. The first way is to provide the expression of the basic constructions of programming with the help of elements of QA-reasoning. The second way is to fill such constructions by the adequate content extracted from QA-reasoning. But both ways of the adaptation are bound with the presentation of QA-models from the data point of view.

B. QA-model of Data

As told above, originally the QA-Model of data had been suggested and developed for the real-time work with such interactive objects as “Tasks”, “Questions” and “Answers” which were kept in the specialized database (QA-database) and used by designers in the corporate network. It is necessary to notice that “Task” is a type of a question and “Solution of the Task” is an answer to such question.

On the logical level the QA-model of data can be interpreted as the specialized hierarchical model of data emulated by means of the relational model of data. Two hierarchical trees of data the units of which are connected as questions and answers is one of specificities of the QA-model of data. The general version of the QA-model of data (presented in Fig. 5) includes the dynamic tasks tree the units of which are united with a system of QA-models for corresponding tasks.

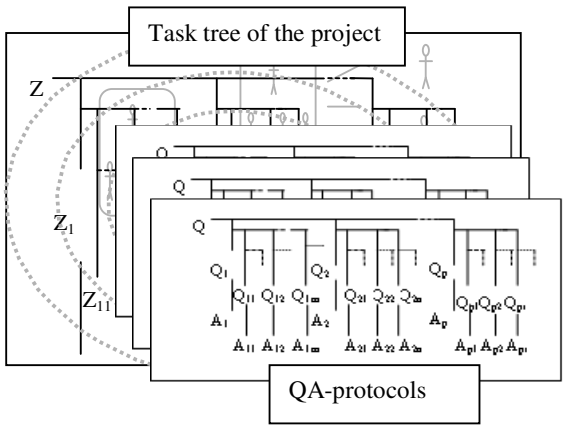

Figure 5. General structure of QA-model of data

Let’s remember that any unit of such model is the interactive object the unique name and symbolic expression of which are visually accessible to designers in the tasks tree or in the corresponding QA-protocol. Other characteristics (for example such basic attributes as name of creator, time attributes, indicator of changes, attribute of inheritances) are being discovered and used in different planned actions with the data unit.

The QA-database which is built on the base of the QA-model of data, has the following useful characteristics:

- allocation on the server with the client access to the content of data in the corporate network with an opportunity of the access from the Internet;
- visualization on the monitor screen with the possibility of the interactive access to corresponding objects;
- personification of Z-, Q- or A-units as the registration of the responsible designer and the group of "support“;
- textual definition Z-, Q- and A-units with an opportunity of the transformation to the language of the logic of predicates;
- transformation of the text for the each unit to the xml-version with positive effects which are being achieved from such form of data.

Enumerated positive characteristics are only a part of a value belonging the QA-databases which can be used not only in the development of SIS. Moreover, it is possible to expand in the interpretation of the connected pair of QA-units by following ways:

- “question” → ”cause” and “answer” → “effect”;
- “question” → ”condition” and “answer” → “reaction”.

Named interpretations and their materializations open new approaches for programming the Expert systems and systems which are based on rules. Therefore the complex of the specialized means have been developed for supporting the work with the QA-database and for programming the applications with such database.

Let’s continue to present the other versions for useful interpretation of the QA-model of data:

- “question” → ”name of the variable for the simple type of data” and “answer” → its “value”;
- “definite composition of questions” → ”typical data” (for example array, record, set, array of records or table, stack, queue and others types of composite data) and “corresponding composition of answers” → its “value”.

So the QA-model of data can be used for emulating the data of many known types. Let’s continue to develop the emulation potential of the QA-model of data. Below, the results of such emulations will be named as QA-data.

C. Means of Additional attributes

Any unit of data is defined by a set of its characteristics which help to code and keep the unit in the computer memory. Any unit of QA-data is accessible through its characteristics also. A set of such characteristics inherits all basic attributes of the corresponding QA-data but in WIQA there is a special mechanism for assigning the necessary characteristics to the definite unit of QA-data. It is the mechanism of additional attributes (AA) which gives the possibility to expand the set of basic attributes for any Z-, Q- or A-object keeping in the QA-database.

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 pseudocode programming. The scheme which is used in WIQA for the object-relational mapping is presented in Fig. 6.

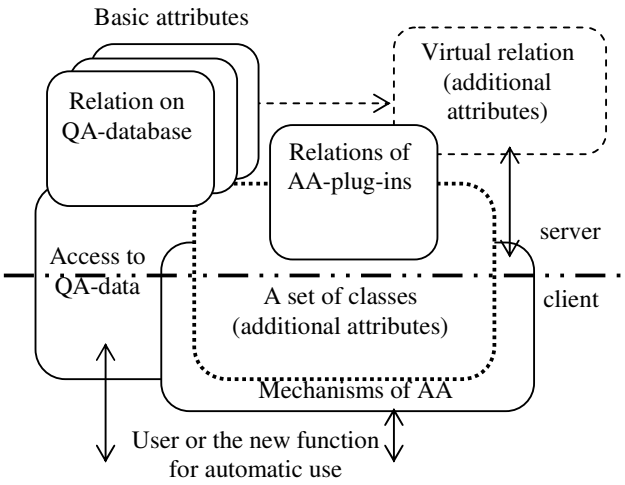

Figure 6. Creation of additional attributes

The usage of the AA is supported by the specialized plug-ins embedded in WIQA. This plug-ins helps the designer to declare the necessary attribute or a group of attributes for definite QA-units. In any time the designer can view declared attributes for the chosen unit. Other actions with the AA must be programmed in C# or in the pseudocode language supported by WIQA.

The built and used pseudocode language (L^P) as other languages of such type is similar to the natural language in its algorithmic usage. The natural language includes universal means for the creation of H-programs executed by H-processors. But this type of algorithmic means is not fitted for K-processors. The language L^P helps to build H-programs which are being executed by the H-processor and K-processor collaboratively.

Any H-program which is written in the L^P-language describes the plan of HCI for the corresponding unit of the behavioral activity of the precedent type. In the process of the H-program execution two types of processors are being included to the collaborative work.

V. QUESTION-ANSWER PSEUDO-PROGRAMMING

A. *Forms for Pseudocode text*

The language L^P as any language for writing the programs includes means for data declarations and means for coding the programs operators. In WIQA any line of any H-program is being written on the "surface" of the corresponding QA-element. In this case the used QA-element can be interpreted as a "material for writing" which has useful properties.

This "material" consists of visualized forms for writing the symbols string originally intended for registering the texts which include questions and answers used in processes of the problem-solving. The initial orientation and features of such type of strings are being inherited by data and operators of H-programs and for this reason they are declared as H-programs of the QA-type. In order to separate this type of H-programs from H-programs of the others types, they will be named below as QA-programs.

The feature inheritance gives the possibility to use the necessary subset of basic attributes and useful additional attributes for processing any line of the source code of the QA-program.

It is necessary to remind, that separate writing of each line of any program was used on punched cards in recent times. Any punch card fulfilled the role of the individual record. Any QA-program also consists of "individual records" but records of the QA-type the efficiency of which is essentially above than at punch cards. Let's notice that means of AA can be used for QA-program strings with means of markup language collaboratively.

B. *Emulation of pseudo-code data*

There are two types of lines of the source code one of which intends for the data emulation and another for the operator emulation. Let's begin to describe the emulation with 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 forms of packed classes (Fig. 7).

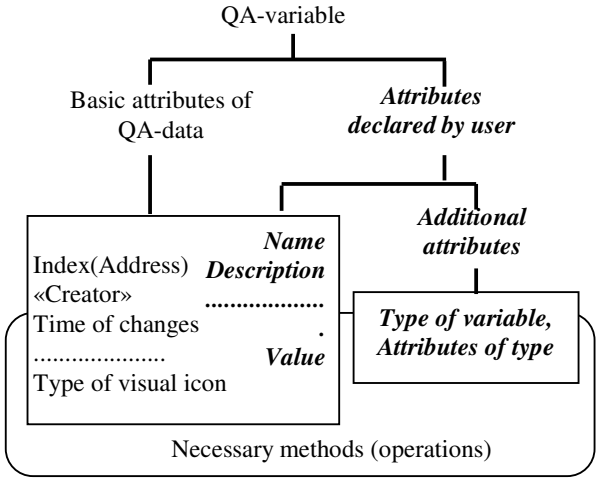

Figure 7. Imitation of variable

For the declaration of variables the constructor of QA-data was developed. This constructor gives the possibilities to name the QA-variable, to choose its type and to appoint the initial value for 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 the H-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 the H-processor. But any declared QA-variable is accessible automatically for the appropriate programs executed by the computer processor also.

An example of keeping the array with elements of integer type is presented in Fig. 8 where a set of additional attributes are used for translating the array declaration to computer codes.

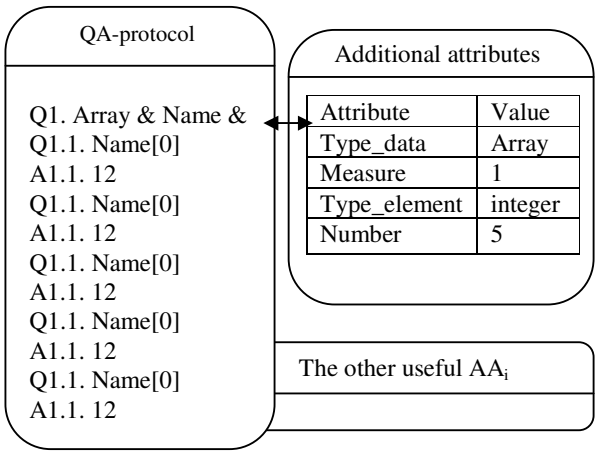

Figure 8. Declaration of array

Attributes which are assigned for the array are visually accessible for the designer at any time and can be used not only for translating. The designer 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 the QA-program.

C. Emulation of pseudo-code operators

The second type of pseudocode lines is 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 symbol string of the "question" can be used for writing (in this place) the operator in the pseudocode form. The fact or the result of the operator execution will be marked or registered in the symbol string of the "answer".

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 the 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 of the algorithmic language;
- any procedure is a typical sequence of actions which are accessible in QA-processor for the execution by the user.

The set of basic operators includes traditional pseudocode 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 operator emulation they will be indicated as QA-operators.

In pseudo-programming languages a set of basic operator is being expanded usually. In 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 pseudocode programs for different tasks [20].

VI. Specimens of QA-Programs

A. Types of QA-Programs

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

But HCI on the base of QA-programs has the additional feature which is implemented in interactions of designers with Z-, Q- and A-objects. This feature is the usage of pseudocode strings of QA-programs as means of HCI. As told above such interactive objects open very useful positive effects for designers who can use or change any string as QA-data in the real time.

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

The structure of any precedent includes a condition part and a part of a reaction each of which has to 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" and therefore a set 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 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 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 the H-processor with precedents the following QA-program scheme is useful:

QA-PROGRAM_1 (condition for the access to the precedent):

Q1. Variable V_1 / Comment_1?

A1.Value of V_1.

Q2. Variable V_2 / Comment_2?

A2. Value of V_2.

……

QN. Variable V_M / Comment_M?

AN. Value of V_M.

Q0. F = Logical expression (V_1, V_2, …, V_M)?

A0. Value of Expression.

End.

It is necessary to notice that the designer 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, designer 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):

Q1.K_i, K_j, …, PL_k ?

A1. *

Q2. K_m, QA-P_n, …, K_q?

A2.*

……

QN. K_s, Pl_t, …, QA-P_v?

AN. #

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 the 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 designer. In such "points" of human-computer interactions the suggested means of HCI are being combined with traditional means of HCI. In second 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 A-units) one type of QA-procedures:

Q1. Quit all programs.

Q2. Start On the menu Run, click.

Q3. Open In the box regedit, type, and then OK the click.

Q4. Move to and select the following key:

HKEY_CURRENT_USER/Software/Microsoft/Office/9.0 /Outlook/

Q5. In the Name list, FirstRunDialog select.

Q6. 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.

……

Q9. In the Confirm Value Delete dialog box click Yes, for each entry.

Q.10. On the Registry menu, click Exit.

Q11. End.

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 is impossible. Therefore all typical QA-programs are kept in the special library.

If the definite typical QA-program should be used the designer must extract this QA-program from the library, create the new task, include the task to the tasks tree and after such actions the designer can start to solve the task (to execute the corresponding QA-program).

The reality of the designer activity is a parallel work with many tasks at the same time. Therefore the special interpreter for executing the QA-procedures and the system of interruption (of the H-processor) 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.

B. Example of QA-Functions

WIQA is the instrumental system which supports the collaborative development of SISs. Moreover WIQA can be used as a kernel of the developed SIS. If the developed SIS is implemented with such kernel then such SIS inherits all potential of WIQA and the possibilities of the QA-programming also.

The expert system of monitoring the sea vessel surrounding is an example of such SIS (named EmWIQA). The following example of the QA-function supports the access to the precedent which presents the 15^{th} rule of the International Rules for Preventing Collisions at Sea [5]:

QA-PROGRAM_3 (conditional access to the precedent).

Q1. Velocity V_1 of the power driven vessel V_1?

A1.Value of V_1.

Q2. Bear_B_1 of the vessel V_1?

A2.Value of B_1.

Q3. Place of the vessel V_1?

A3. Coordinates of the place_1.

Q4. Velocity V_2 of the power driven vessel V_2?

A4.Value of V_2.

Q5. Bear_B_2 of the vessel V_2?

A5.Value of B_2.

Q6. Place of the vessel V_2?

A6. Coordinates of the place_2.

Q7.CPA = expression for computing the Closest Point of Approach (CPA)?

A7. Value of CPA.

Q8. Cond = (V_1, "keep out of the way")&

& ($\left|$ Bear_1 - Bear_2 $\right|$ > 11, 5^o) &

& (CPA-D^{DA}- $\Delta D_1 \leq 0$)?

A8. Manoeuvre_M_i.

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. Such functions are 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.

VII. 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 the last version of WIQA which has been created on C# at the platform of Microsoft.NET 3.5.

The first compiler provides the processing of QA-programs which describe the conditional parts of precedents. Copies of such compiler can be embedded to 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 H-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 the 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. 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. 9 with commentary labels as for Fig 2. (all interfaces in Russian).

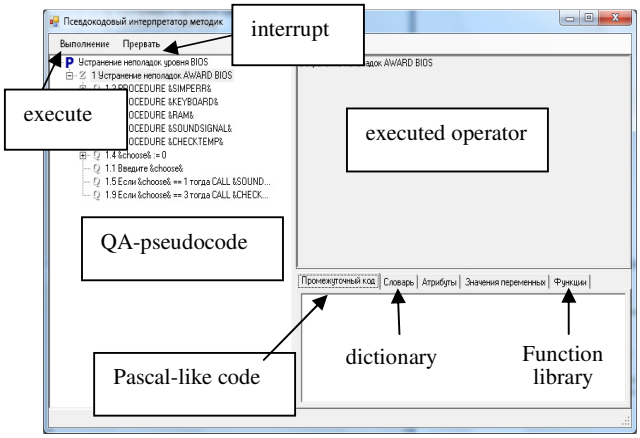

Figure 9. Screenshot of interpreter

Interfaces of the main form help to control as executing the QA-program so its debugging. The user who is fulfilling the role of the H-processor can interrupt the H-process on any operator of the QA-program with the possibility of returning to the point of 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).

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 the QA-programs which can be translated for their executing by computer processors also.

Pseudocode texts of QA-programs can be written and executed (in the real time) by designers working in the corporate network. Designers interact with QA-programs as with intermediators between the human and computers and it gives the arguments to qualify **QA-programs** as new type of means for HCI. Moreover, such intermediators can be translated (in WIQA) firstly to the C# source code and then to the executed code.

VIII. Aspect-Oriented Designing of Interface Prototypes

The presented means of HCI open the effective possibilities for aspect-oriented designing the traditional versions of HCI. For all usability metrics, which are defined in the standard ISO/ MEK–9126, corresponding precedents were created. All of them are united in the library of the typical tasks. Any task of this library is programmed with the usage of QA-means so that the corresponding usability metrics is accessible to designers as the definite interface precedent. The created library consists of 73 typical tasks any of which can be used for generating the necessary quantity of copies adjusted to the places of their materializations in the implemented system.

When in current solving of the project task the designer discovers the next "point of human-computer interaction" then the appropriate metrics task is being included to the tasks tree of the designed SIS. Such task has two subtasks one of which is a pseudo-program of the precedent condition. The second subtask is a pseudocode technique providing the inclusion of the chosen metrics into the solution of the project task.

In order to simplify the use of the aspect-oriented technique the special plug-ins for the interface prototyping of the project solutions is developed and embedded into WIQA. The necessary interface prototype 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 the QA-program is filling by the chosen interfaces precedents.

IX. Conclusion

Told above contains sufficient arguments to assert that the real time programming of HCI by the user leads to many positive effects in the usage of SISs and their development. QA-programming is the rational way for such work which can be implemented with the help of WIQA means. QA-programming of HCI can be implemented at the project level (as the creation of the tasks tree) and at the pseudocode level (as writing the QA-programs for H-processors and computer processors).

QA-programs are useful means of HCI which are additional for traditional means of HCI. Such means of HCI are adjusted for the access to the human experience in the precedents forms which were used in creating the library of the usability metrics implemented as the set of tasks with embedded interfaces

precedents.

QA-programs are the kind of pseudo-programs. Any line of the source code of such pseudo-program inherits the property of the appropriate QA-unit which is used as the "material" for writing this line. At any time the programmer can expand the set of attributes for any line of the definite QA-program if it helps to solve the corresponding task. The programmer has the possibility to use the line attributes of the source code in the operators of the created pseudo-program.

QA-programs also manage accustomed (habitual) semi-automatic actions when QA-programs (as techniques of the guide type) show to the designer the sequence of actions which designer must execute by "touching" with the help of the marker (or another way) the special signs or definite area on the monitor screen. Moreover, QA-programs can be translated to the form which can be executed by the computer processors.

References

[1] L. Bass, J. Ivers, M. Klein, P. Merson. "Reasoning Frameworks," *Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2005-TR-007*, 2005.

[2] G. Booch, A. W. Brown. Collaborative development environments. In M. Zelkowitz (Ed.), *Advances in computers, 59, San Diego, CA: Academic Press*, 2003.

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

[4] S.K.Card, T.P. Thomas, A. Newell. *The Psychology of Human-Computer Interaction*, London: Lawrence Erbaum Associates, 1983.

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

[6] A. Crystal, B. Ellington. "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, 2004.

[7] E. Dijkstra. "Programming Considered as a Human Activity." *Classics in software engineering. ACM Classic Books Series*, pp. 1-9, 1979.

[8] S. Henninger. "Tool Support for Experience-Based Software Development Methodologies," *Advances in Computers,* Vol. 59, pp. 29-82, 2003.

[9] T. Hewett, R. Baecker, St. Card, T. Carey, J. Gasen, M. Mantei, G. Perlman, G. Strong, W. Verplank. "ACM SIGCHI Curricula for Human-Computer Interaction."*ACM Technical Report*, p. 162, 2002.

[10] L. Hirschman, R. Gaizauskas. "Natural Language Question Answering: The View from Here".*Natural Language Engineering,* Vol. 7, pp. 67-87, 2001.

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

[12] D. Kieras, D.E. Meyer. "An overview of the EPIC architecture for cognition and performance with application to human-computer interaction". *Human-Computer Interaction*, Vol. 12, 391-438,199.

[13] D. Knuth. "Computer Programming as an Art." *Communications of the ACM,* vol. 17,(12). pp 667-673.

[14] P. Kroll, Ph. Kruchten. *The Rational Unified Process Made Easy: A Practitioners Guide to the RUP*. Addison-Wesley, 2003.

[15] M.H. Lee. "Model-Based Reasoning: A Principled Approach for Software Engineering", *Software - Concepts and Tools*, Vol.19, #4, pp. 179-189, 2000.

[16] C. Potts, K. Takahashi, A. Anton. "Inquiry-based Requirements Analysis, " *IEEE Software*, Vol. 11, #2, pp. 21-32, 1994.

[17] S. K. D'Mello, A. Graesser, B. King. "Toward Spoken Human-ComputerTutorialDialogues*" Human Computer Interaction*, Vol. 25, # 4, pp. 289-323, 2010.

[18] R.Reiff, W.Harwood, T. Phillipson. "A 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, 2002.

[19] C. Rich, Y. Feldman. "Seven Layers of Knowledge Representation and Reasoning in Support of Software Development," *IEEE Transactions on Software Engineering*, Vol. 8, # 6, pp.451-469. 1992.

[20] P. Sosnin. Means of question-answer interaction for collaborative development activity", *Hindawi Publishing Corporation, Advances in Human-Computer Interaction*, vol. 2009, Article ID 619405, 2009.

[21] P. Sosnin. "Question-Answer Approach To Human-Computer Interaction In Collaborative Designing." // *In proc. IASDIS: Human Computer Interactions, Freiburg, Germany*, pp. 219-226, 2010.

[22] J.J.B. Vicente, F. Klett. "A Generic Evaluation Framework for Knowledge-Based Infrastructures: Design and Applications", *International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM)*, Vol. 3, pp. 290 -297, 2011.

[23] F. Yang, R. Shen, P. Han. "Adaptive Question and Answering Engine Base on Case Based and Reasoning Technology," *Journal of Computer Engineering*, Vol.29, #11, pp. 27-28, 2003.

Author Biography

PETR SOSNIN was born in Ulyanovsk in the USSR, on July 12, 1945. He graduated from the Ulyanovsk Polytechnic Institute (1968).

His employment experience included the Ulyanovsk Polytechnic Institute and Ulyanovsk State Technical University. His special field of interests includes AI applications for computer aided design. P. Sosnin defended doctor degree in Moscow Aviation Institute (1994). He is an author of eight books and more three hundred articles.