I T H E A

# International Journal

# INFORMATION THEORIES & APPLICATIONS

2009 Volume 16 Number 2

# International Journal
# INFORMATION THEORIES & APPLICATIONS
**Volume 16 / 2009, Number 2**




**IJ ITA is official publisher of the scientific papers of the members of the ITHEA® International Scientific Society**

IJ ITA welcomes scientific papers connected with any information theory or its application.

IJ ITA rules for preparing the manuscripts are compulsory.
The **rules for the papers** for IJ ITA as well as the **subscription fees** are given on *www.ithea.org* .
**The camera-ready copy of the paper should be received by http://ij.ithea.org.**
Responsibility for papers published in IJ ITA belongs to authors.

General Sponsor of IJ ITA is the **Consortium FOI Bulgaria** (www.foibg.com).


**International Journal "INFORMATION THEORIES & APPLICATIONS" Vol.16, Number 2, 2009**

**Printed in Bulgaria**

Edited by the **Institute of Information Theories and Applications FOI ITHEA®**, Bulgaria,
in collaboration with the V.M.Glushkov Institute of Cybernetics of NAS, Ukraine,
and the Institute of Mathematics and Informatics, BAS, Bulgaria.

Publisher: **ITHEA®**
Sofia, 1000, P.O.B. 775, Bulgaria. www.ithea.org, e-mail: info@foibg.com





**ISSN 1310-0513 (printed)** **ISSN 1313-0463 (online)** **ISSN 1313-0498 (CD/DVD)**

# METHODS FOR AUTOMATED DESIGN AND MAINTENANCE OF USER INTERFACES

**Valeriya Gribova**

*Abstract: Methods for automated development and maintenance of intellectual software user interfaces are proposed. They rest on an ontology-based approach and intended for decreasing effort and time for user interface development and maintenance. A survey, principal conception of the approach, project components and methods of automated development of the project as well as comparison the methods with their analogues are described.*

*Keywords: ontology, interface project, automated generation*

*ACM classification: I.2.2 Automatic Programming*

## Introduction

An important task of software development is user interface development according to users' requirements. A major tendency is user interface complication. It grows out of increasing functionality of software and different conditions of its application which are changing constantly. As a result, development, implementation and modification of user interfaces are costly and time-consuming tasks because the developer has to take into account a lot of interrelating but often conflicting factors. According some reviews (for example, [1]) the user interface development takes 50% time required to software and it's the source code aggregates 48% of the total code.

To decrease labour-intensiveness and time for user interface development, an ontolology-base approach to its development, automatic generation and maintenance is proposed. The basic ideas of the approach are:

- to form a user interface project by ontologies that describe features of every component of the project;
- based on this project to automatically generate the user interface code to a programming language;
- to link the interface code with the application code (a programming language, an interaction type with application, local or distributed, is determined by the developer).

Therefore, development, implementation, maintenance of a user interface is added to development and maintenance of its project.

Application of the tool based on the ontological approach has shown that design and maintenance of the presentation component in the user interface project remains difficult for complicated and complex interfaces due to high requirements to developers. They are to know usability principals, various standards of development, take into account users' requirements, conditions and environment of using software, and so on. As a result, development of methods and tools for automation of the presentation component design in the user interface project is an urgent task.

## Present state in the field of user interface design automation

User interface development is a very complicated task. Experts in different fields take part in its design. To create a user interface in accordance with tasks to be solved the following information is required for the developer [2]:

- A domain model for choosing appropriate user interface elements for input/output data;

- A Style Guide for the target platform (for example, Apple Human Interface Guidelines [3], Windows Vista User Experience Guidelines [4], Osf/Motif Style Guide; SAP - R/3 Style Guide [5], Style Guide and Certification Checklist [6] etc.) for providing uniform for a target platform interface style, choosing interface elements and including in a user interface additional features specific for the platform;
- A task model for determining a set of windows according to a scenario dialog;
- A user model for design the user interface suitable for a user of the interface;
- An application model for linking the application with the user interface.

At present user interfaces are developed, as a rule, with specialized tools – Interface Builders and Model-Based Interface Development Environments.

Fig. 1 shows the basic scheme of the user interface development using Interface Builders.

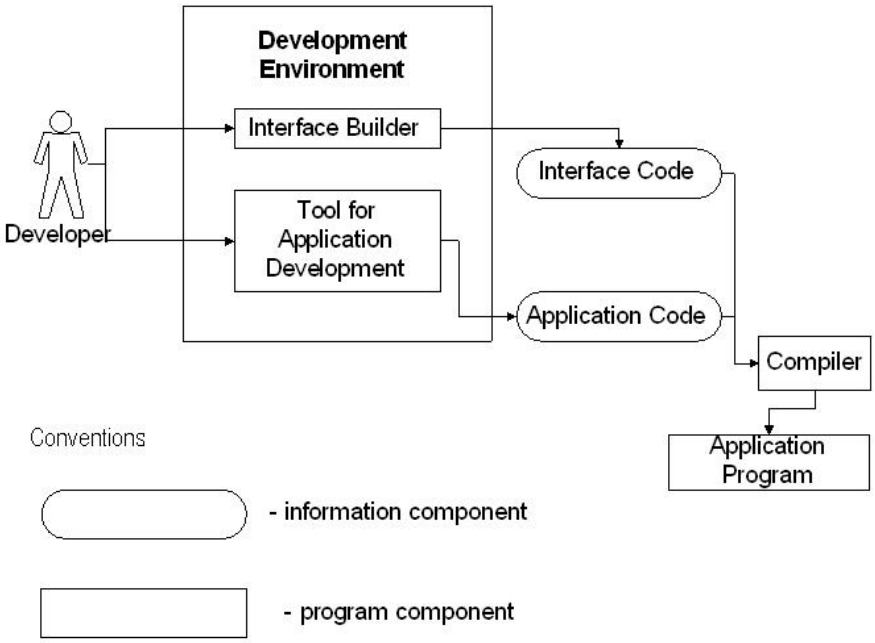


Fig 1. The basic scheme of the user interface development using Interface Builders.

Interface Builders do not have special tools for development of domain and task models [2] so developers use third-party software or an application specification. Interface Builders support design of only standard interface elements and designers should know rules of choosing and locating them. Solutions of these tasks depend on designer experience and qualification. For linking the user interface and the application a program code in a programming language (C++, Pascal, Java, C# etc.) should be written.

As an alternative to eliminating problems with Interface Builders, Model-Based Interface Development Environments (MB-IDE) are used. The key difference between Interface Builders and MB-IDEs is a declarative model interface, including all required information about the user interface [7-10]. Fig. 2 shows the basic scheme of the user interface development with MB-IDEs. Using modeling tools the developer forms components of the user interface model. Then, using this model, the source code is generated. First, the developer forms a domain model and a task model; after that based on these models, the user interface designer forms a presentation and scenario dialog models. The next step is to form a model of linking the interface with the application. These models are necessary for automated generation of the user interface code.

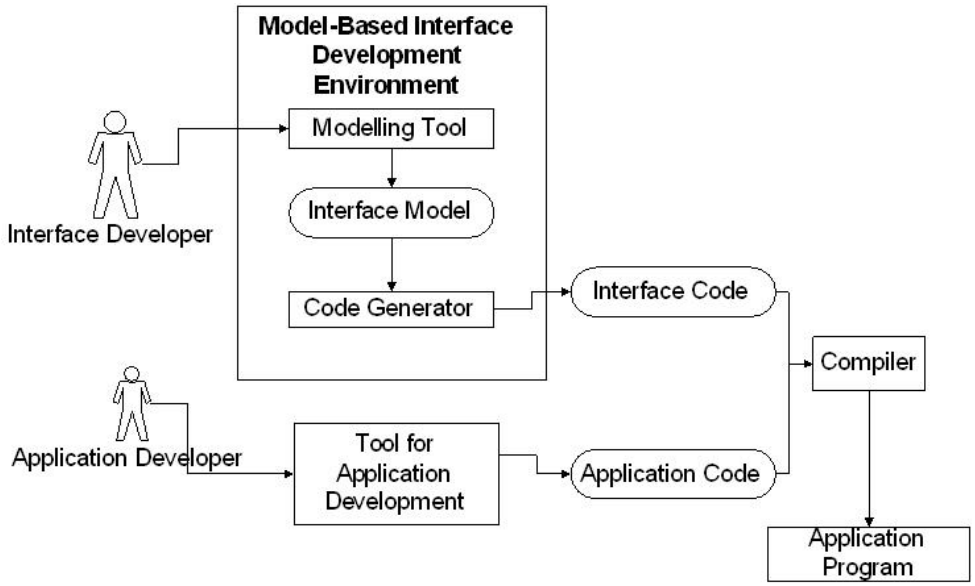


Fig. 2. The basic scheme of the user interface development using MB-IDEs

Some MB-IDEs have modeling tools. They assist the developer to build the components of the user interface model [10-13]. The main goals of the modeling tools are to hide the syntax of the modeling languages from developers, and to provide a convenient interface for developers for specifying large quantities of information stored in the model. A wide range of modeling tools has been developed, often specified different levels of the model. These tools may be text editors to build textual specifications of models (ITS [13, 14], Mastermind [15]), form-based tools to create and edit model elements (Mecano [16]), and specialized graphical editors (Humanoid [12, 17, 18], FUSE [11], and many others).

As shown in **Error! Reference source not found.**, automated design tools usually use a repository of design knowledge or design guidelines to control the behavior of the design tool.

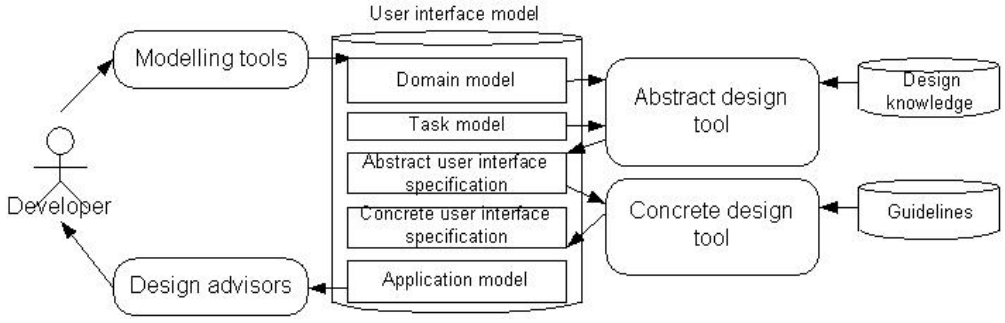


Fig. 3. The basic scheme of user interface model development using modeling tools

The primary goal of many MB-IDEs is to automate the design and implementation of a user interface. Experts develop a domain model (it describes the structure and attributes of the information that the application provides)

and a task model (it describes the tasks that users need to perform) and then the abstract and the concrete user interface specifications are automatically generated from these models.

Most MB-IDEs make the following steps to automatically design an interface:

- To determine the presentation units; this step determines the windows that will be used in the interface, and information that will be shown in each window;
- To determine the navigation between the presentation units; this step computes a graph of the presentation units;
- To determine the abstract interaction objects for each presentation unit; the abstract interaction objects specify the behavior of each element in a presentation unit in an abstract way (for example, "to select one from a set").
- To map abstract interaction objects into concrete interaction objects; the concrete interaction objects represent the widgets available in the target tool;
- To determine the window layout; this step determines the size and position of each concrete interaction object;

The first three steps build the abstract user interface specification, and the last two build the concrete specification.

Even though automated design MB-IDEs help to considerably decrease development efforts, the quality of the generated interfaces is low. The main shortcomings of MB-IDEs are:

- MB-IDEs and their modeling tools realize the only style guide and do not have any facilities to add other style guides;
- Usability requirements are often changed, style guides are often updated (for example, rules of appropriate interface element choosing), but MB-IDEs do not allow developers to keep up with the changes;
- All interface elements and their attributes are rigidly inserted in the modeling tool code so there it is impossible to extend attributes interface elements or insert new elements;
- All MB-IDEs use their own declarative models and specification languages incompatible with each other so developers can't use advantages of different tools;
- There are no modeling tools which take into account all aspects necessary for user interface development: user requirements, the context of using application, and domain and user task structures.

One possible way to increase the quality of automatically generated interfaces, and it has been applied in MB-IDEs, is to allow developers to make post-editing. However, many MB-IDE's developers are faced with the challenge:

How should the changes performed during the post-editing in all components of the interface model be recorded and reapplied (for example, if the developer deleted an interface element, how should he save this information in components of the model and change all links between them).

Another solution made in MB-IDEs is to specify steps 1 and 2 in a structure called a dialogue graph by the developer. Steps 3 and 4 are generated by the system in a table with default entries, but developers can edit these tables and delete any entry. Step 5 is done automatically. However, this solution, as mentioned in literature, increases the labor of development [10].

## User interface project

A user interface project defines a set of dialogs with the user for input/output parameters, control of the application program, and providing context-sensitive help for users [19, 20]. The interface project consists of a

domain project, a task project, a presentation project, an application program project, a scenario dialog project, and a mapping project.

**The domain project (DP)** describes a set of domain terms consisting of input/output parameters of an application program, control information, and intellectual support of users activities. This component is specified by the domain ontology and is expressed by the pair: DP=( G, $\sigma$ ), where G is an oriented graph of the DP, $\sigma$ is a graph labeling. The oriented graph G=<Vertices, Edges, RootVertex>, where Vertices is a set of graph vertices, Edges is a set of graph edges, RootVertex is a root vertex.

The set of graph vertices Vertices={$Vertex_i$}$_{i=1}^{vertexescount}$ consists of two subsets – a set of terminal vertices and a set of nonterminal vertices, $Vertex_i \in$ Nonterminal_Vertices $\cup$ Terminal_Vertices, where Nonterminal_Vertices is a set of nonterminal vertices, Terminal_Vertices – a set of terminal vertices.

The set of graph edges Edges = {$Edge_i$}$_{i=0}^{arcscount}$, $Edge_i$=<$VertexFrom_i$, $VertexTo_i$ >, where $VertexFrom_i \in$ Vertices, $VertexTo_i \in$ Vertices. The root vertex RootVertex is nonterminal vertex, RootVertex$\in$Nonterminal_Vertices. The graph labeling $\sigma$ is transformation of the vertex set and the edge set to a name set $\Omega$, where $\Omega$ = DomainName $\cup${$GroupName_i$}$_{i=1}^{termgroupcount}$ $\cup$ {$TermName_j$}$_{j=0}^{termcount}$ $\cup$ {$TermAttribute_i$}$_{i=1}^{attributecount}$ $\cup$ {$QualityValue_i$}$_{i=2}^{valuecount}$ $\cup${(($RealMin$, $RealMax$), $RealMeasure$)$_i$}$_{i=0}^{floatcount}$ $\cup${(($IntegerMin$, $IntegerMax$), $IntegerMeasure$)$_i$}$_{i=0}^{integercount}$ $\cup${$N_i$}$_{i=0}^{stringscount}$ $\cup${TermGroup, Term, Attribute, Joint, Disjoint, RealValue, IntegerValue, SrtingValue}. The name set $\Omega$ is defined by names of the domain ontology, and graph labeling is defined by the vertex type $VertexFrom_i$ and $VertexTo_i$.

**The task project (TP)**. The propose of an application program is to solving user tasks. Tasks could be divided into subtasks. Subtasks are steps for solving the main task. There are some relations among tasks. They determine conditions and the order of task performance [21]. TP=(T, $\sigma$), where T is a task tree, $\sigma$ is a labeling tree. The task tree T=<Vertices, Edges, RootVertex>. The tree labeling $\sigma$ is a transformation of the vertex set to a name set $\Omega$, where $\Omega$ ={$TaskName_i$}$_{i=1}^{taskcount}$ $\cup$ SetType. The name set $\Omega$ consists of task names and set names. SetType ={«choice», «interleaving», «enabling», «deactivation»}. Labeling $\sigma$ is defined as RootVertex$\rightarrow$ CommonTaskName, the root vertex transforms to the common task name. Every vertex $Vertex_i \rightarrow$ ($Label_{1i}$,$Label_{2i}$), where $Label_{1i} \in$ {$TaskName_i$}$_{i=1}^{taskcount}$, $Label_{2i} \in$SetType (semantics of sets is defined in [21]).

**The presentation project** describes the structure and parameters of WIMP-interfaces. This component is specified by the WIMP-presentation ontology and is defined as a set of windows: Windows={$Window_i$}$_{i=1}^{windowcount}$, where $Window_i \in$ Controls | Controltype = ContainerWindow. According to the WIMP-presentation ontology every interface element $Control_i$ is defined by a type, sets of parameters, functions, and events. The type, functions and events of an interface element have been defined in the ontology so the presentation project describes values of interface parameters for the interface project. Other interface elements could also be parameters of a window $Window_i$. In this case $Param\_Type_k$ = $Control_m$, where $Control_m \in$Controls.

**The application program project** describes a method of interaction between the user interface and the application program, and program interfaces to provide this interaction. This component is specified by the corresponding ontology and is defined as a set of Interfaces={$Interface_i$}$_{i=1}^{interfacecount}$. Every element of the set consists of the program interface definition assigned by the application program, $Interface_i$ = < $Interfacename_i$, $InteractionModel_i$, $Functions_i$>, where $Interfacename_i$– a program interface name. $InteractionModel_i$ is an element

from the set IModels={Local, Distributed}; a function set Functions describes program interfaces assigned by the application program.

**The scenario dialog project** describes a first window StartWindow, specifying the beginning of dialog with users, StartWindow=$Window_i$, where $Window_i \in$ Windows, and so a set of possible conditions of dialog States = $\{State_i\}_{i=1}^{statecount}$. A state $State_i$ consists of a definition of the event $Event_i$, the set of variables $Variables_i$ for definition of a instruction sequence called $Instructions_i$ and the instruction sequence $Instructions_i=<Instruction_{ij}>_{j=1}^{instructioncount}$. The instructions can be:

- program interface calls of the application program;
- system function calls;
- compound elements ("if" and "cycle" constructions) consisting of a sequence of the mentioned above elements.

The scenario dialog project is specified by the scenario dialog ontology.

**The mapping project** describes a mapping set among different elements of the interface project: elements of the domain and the task projects are mapped on parameters of interface elements in the presentation project, parameters of interface elements on output parameters of the application program in the scenario dialog project, and so on.

## Methods for specifying the presentation project

Interface element parameters of the presentation project are specified by the domain and the task projects, that is for every domain term (groups of terms) and every task (tasks) the interface developer chooses its (their) presentation in the interface, a set of interface elements with specified properties (Fig. 4).

The following problems arise in this case:

- Time-consuming development of presentation projectы for large size of the domain or the task projects;

- Labour-consuming maintenance of the interface project (modification of the domain project requires modification of the presentation project);

- High-level professional requirements to developers (they must know usability principals, requirements of standards, users, a domain for information presentation, and so on).

To meet the main requirement to the interface project, namely, easy modification and maintenance, some methods for specifying the presentation project are suggested. The choice of one of them is determined by the developer and depends on features of components of the interface project (for example the size of the presentation or task project, the domain features specifying type of information, and so on).

**The method for direct specifying the presentation project.** In this case values of interface element parameters are direct references (Fig. 4) either to domain project elements or task project elements, that is, for an interface element of the presentation project $Control_i \in$Windows, where Param_Type=String, Param_Value=Value, Value$\in \{TaskName_i\}_{i=1}^{taskcount} \cup$ DomainName $\cup \{GroupName_i\}_{i=1}^{termgroupcount} \cup \{TermName_j\}_{j=0}^{termcount} \cup \{AttributeName_i\}_{i=1}^{attributecount} \cup \{QualityValue_i\}_{i=2}^{valuecount}$. For example, in Fig. 4 parameter values "Text" of the list box elements are references on attribute values "localization" from the domain project. The number of list box elements is determined by the number of "localization" attribute values; modification of the number of this attribute values results in modification of the number of list box elements; modification of themselves values of this attribute in the domain project results in modification of the "Text" parameter.

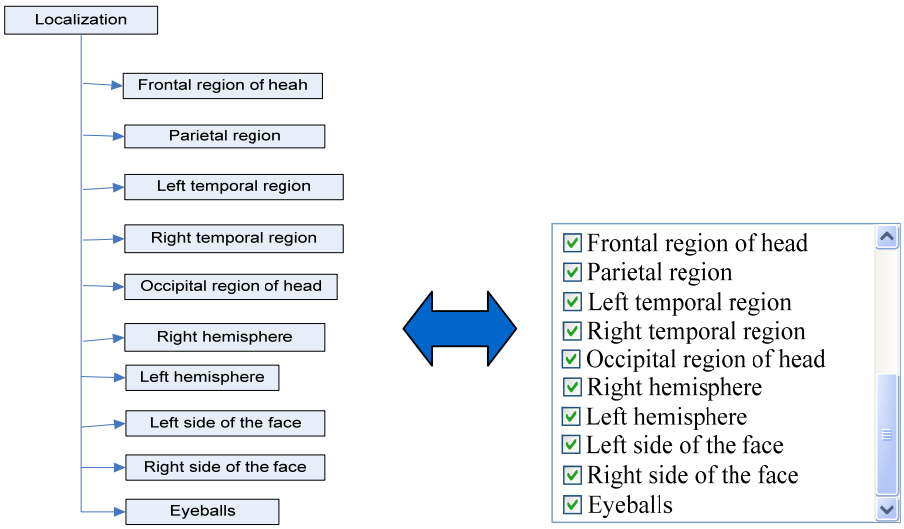


Fig. 4 The example of direct specifying of the presentation project

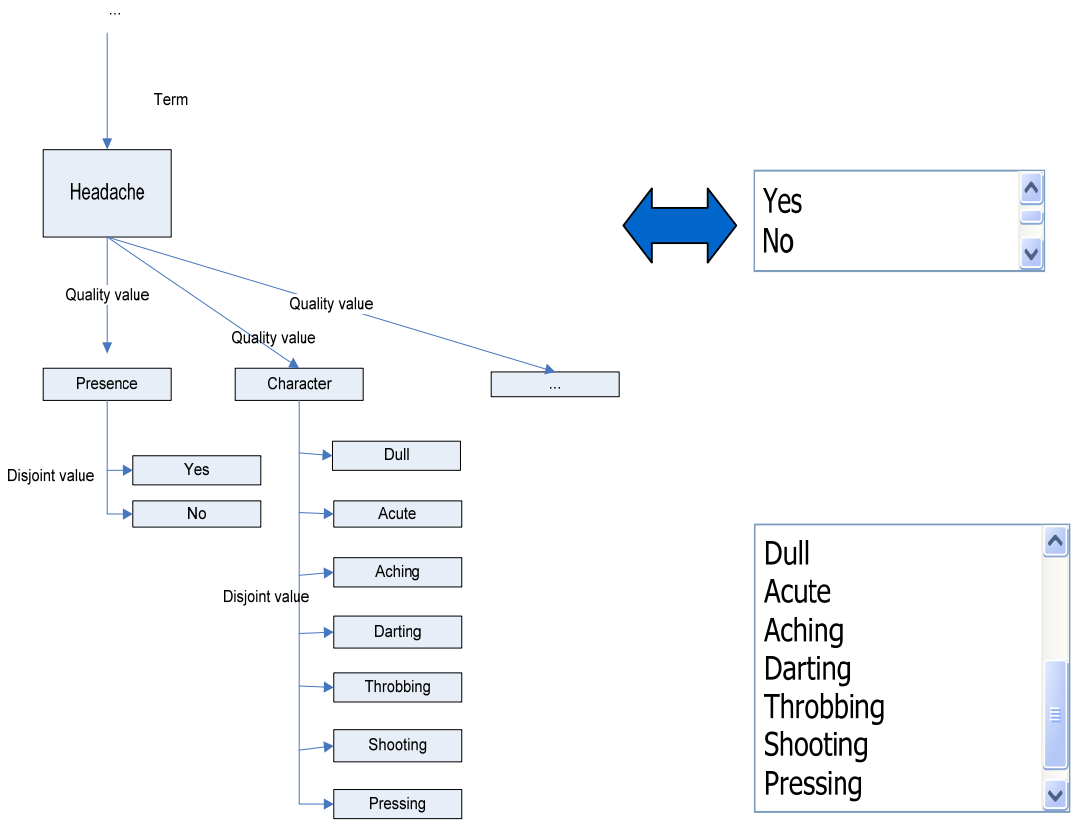


Fig. 5 The example of regular specifying of the presentation project

This method of specifying of the presentation project is convenient for small size of the domain or the task projects and so if accurate presentation domain terms in an interface is required (for example if the user has special requirements).

**The method of regular specifying of the presentation project.** In this case values of interface elements parameters are references on components the domain ontology (but no on components of the domain project): groups of terms, terms, values, attributes. That is for an interface element of the interface project $Control_i \rightarrow Element \mid Controli \in Windows, Element \in \{TermGroups, Terms, Attributtes, QualiyValues\}\}_{i=1}^{controlcount}$.

As a result every term group (terms, values, attributes) included in the domain project is represented by an interface element which the developer choses. In this case the value of a string parameter(s) is (are) elements of the domain project.

For example, the parameter value "Elements" = $\{ListBoxElement_i\}_{i=1}^{elemcount}$ of list box elements are corresponded with all quality joint values from the domain ontology. It means that all quality joint values from the domain project are presented by the same interface elements (Fig.5). At the same time their quantity could be arbitrary large and is determined by the domain.

This method of specifying of the presentation project is convenient for large size of the domain or the task projects (hundreds and thousands of terms).

**The method of fragmentary specifying of the presentation project.** In this case the developer divides the presentation project on a set of fragments based on user requirements, functionality and semantic of the application program; every fragment is corresponded with their representation in the interface: $\{Fragment\_Presentastion \rightarrow Fragment\_Terms \mid Fragment\_Presentastion \in \{Control_i\}_{i=1}^{controlcount}, Control_i \in Windows, Fragment\_Terms \subset G\}_{i=1}^{fraggmentcount}$.

The basic statements of this method are:

- Every fragment of the domain project is automatically corresponded with a set of possible representations.
- Corresponding are based on usability principals.
- If a fragment has some possible representations in according to usability principals then the developer has the case:
  - He or she choose the only representation from the set of possible representations;
  - The representation is automatically chosen in according to default rules.
- The explanation containing the protocol of executing is given to the developer. The protocol consists of the set of used criteria with description of accepted and unaccepted criteria (criteria in chosen in according to usability principals).
- A set of criteria must be expanded (because usability principals are grown and modified).

To provide the expanding set of possible representations of a presentation mapping ontology is suggested. In the terms of the ontology rules of corresponding domain terms with their possible representations (in according to usability principles) is determined. In this way the set of mappings (the knowledge base) is formed. A mapping is described as a pair – a mapping condition and a set of possible representations. The condition could be consisted of a set of conditions, every of them determines parameter values. Conditions are verified by the domain ontology. Parameters of conditions are determined in the terms of the domain ontology. For example, a condition could be quantity of joint values of a term from the domain project and a method of their selection. (joint or disjoint). Based on various values of these conditions a set of representations are chosen: a set of radio-buttons,

check boxes, a list box, a combo box, etc.). Two various fragmentations for the same domain project fragment and corresponding to it's the presentation project fragment are presented at Fig. 6.

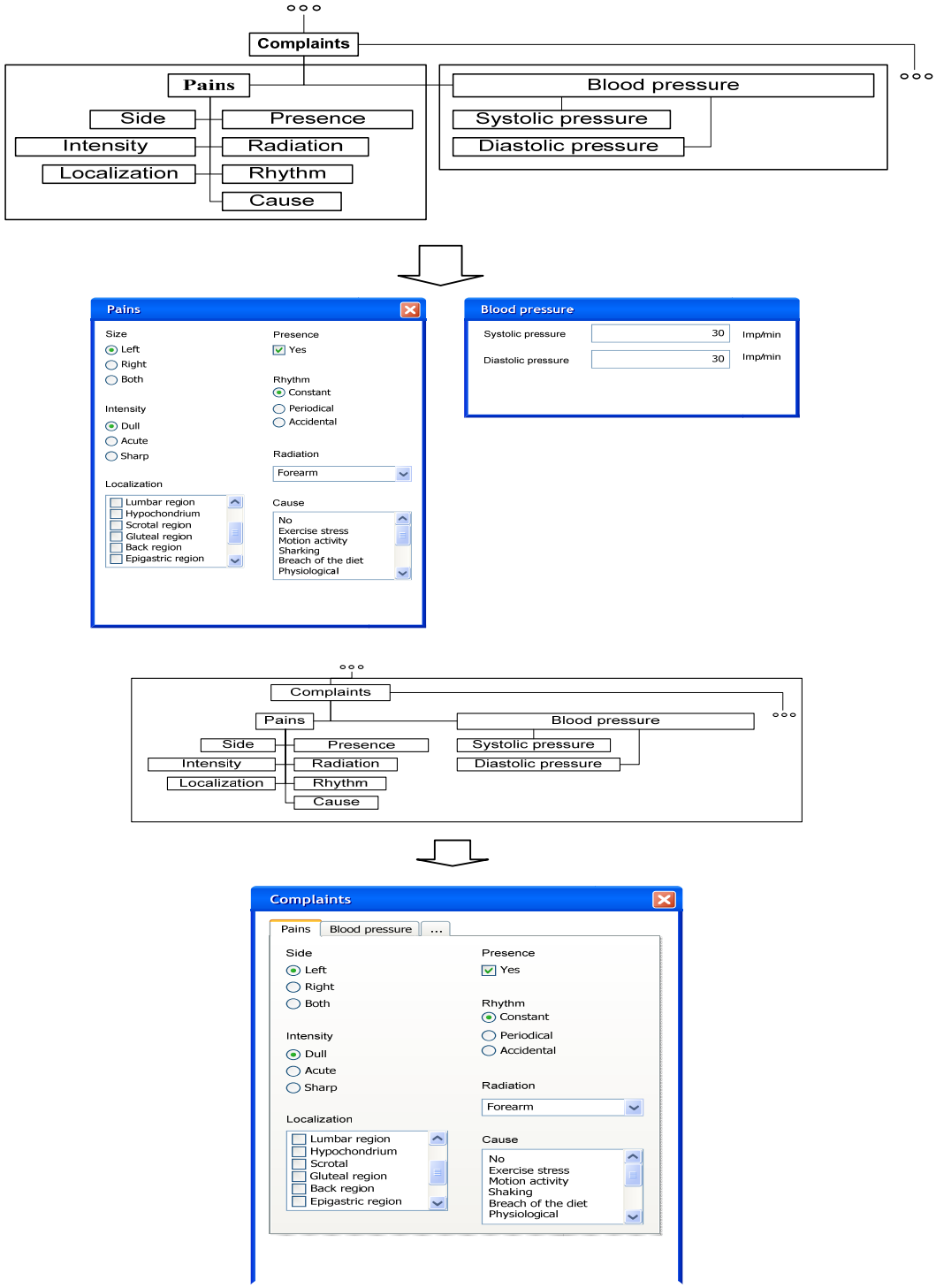


Fig. 6 The example of fragmentary specifying of the presentation project

As you see from the Fig. 6 different fragmentations of the domain project lead to different presentation projects. In the first case the fragment has two fragmentations. Every fragmentation is corresponded with the window ("Pain"

and "Raising blood pressure"). In the second case the fragment has the only fragmentation corresponded with the window "Complaints".

This method of specifying of the presentation project is convenient (as the previous method) for large size of the domain or the task projects (hundreds and thousands of terms).

## Discussion of results

At present the methods described above have been implemented and integrated in a tool for development, automatic generation and maintenance of user interfaces based on the ontology-based approach. These methods have been used for implementation and maintenance interfaces for a number of application programs.

In discussing results obtained it is reasonable to compare them with Model-based interface development environments (MB-IDE) for development and automatic generation of user interfaces [22-24]. The other class of specialized tools – Interface Builders does not realize such methods.

The method for direct specifying the presentation project is implemented in some MB-IDE in the following way: string parameter values of interface elements are domain terms or users' tasks. However, any modification of domain terms or users' tasks constrains to modify string parameter values of interface elements also, and then to recompile the interface. This results in laborious interface maintenance. In the suggested method string parameter values of interface elements are references to domain terms or users' tasks, so modification of domain terms or users' tasks result in automatic modification of the string parameter values of interface elements and no recompilation is necessary.

The method for regular specifying the presentation project is suggested for the first time. It allows the developer to considerably decrease laboriousness of development and maintenance. For example, the domain project of the System for intellectual support of medical activity for urologists [25] consists of 700 terms and groups of terms and about 5000 possible values. In this case the method of regular specifying the presentation project provides fast specification and automatic maintenance of the presentation project.

The method for fragmentary specifying the presentation project has been used in some MB-IDE. With this method developers determine a window (a set of windows) and information for every window (from the domain and the task projects). Then a presentation for this information is chosen automatically and the developer only enhances the presentation. On the one hand, such an approach decreases the time of presentation project development; on the other hand, the main problem for developers is difficult maintenance of the project, because modification of terms and tasks result in automatic representation of these information and all enhancements are lose. Attempts to save results of enhancements were realized in some MB-IDEs. Nevertheless, they failed in case of addition or deletion of new terms or tasks. Using references solves this problem. Another drawback of this method in some MB-IDE is strict embedded of mappings (a set of presentations) to the tool. As a result, the developer is not aware of these mappings and they could not be expanded. According the method suggested all mappings are in the knowledge base and could be viewed and modified.

## Bibliography

1. Myers Brad, Rosson Mary. Survey on user interface programming // Tech. Rpt. CMU-CS-92-113, Carnegie-Mellon, School of Comp. Sci., February 1992. http://citeseer.ist.psu.edu/myers92survey.html
2. Baar de D., Foley J.D., Mullet K.E. Coupling Application design and User interface design // CHI'92 Conference Proceedings, Monterey, ACM Press, 1992, pp.259-266
3. Apple Human Interface Guidelines – Electronic resource – http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html

4. Windows Vista User Experience Guidelines – Electronic resource – http://msdn2.microsoft.com/en-us/library/aa511258.aspx
5. SAP - R/3 Style Guide – Electronic resource – http://www.sapdesignguild.org/resources/MiniSG/index.htm
6. Common Desktop Environment: Style Guide and Certification Checklist - Electronic resource – http://docs.sun.com/app/docs/doc/802-6490?q=Motif+Style+Guide
7. Eisenstein J., Puerta A. Adaptation in Automated User-Interface Design // Intelligent User Interfaces, 2000, pp. 74-81
8. Myers B.A. User Interface Software Tools // ACM Transactions on CHI'95. Vol. 2, No. 1, 1995, pp 64-103
9. Puerta A.R., Eriksson H., Gennari J.H., Musen M.A. Beyond Data Models for Automated user Interface Generation // British HCI'94 Confirence Proceding, Glasgow, Cambridge University Press, 1994, pp.353-366
10. Szekely P. Retrospective and Challenges for Model-Based Interface. 1996. http://citeseer.nj.nec.com/szekely96retrospective.html
11. .Lonczewski, F., Schreiber, S.: The FUSE-System: an Integrated User Interface Design Environment. In: Vanderdonckt J. (ed.): Proceedings of CADUI'96. Namur: Presses Universitaires de Namur 1996 (pp. 37-56).
12. Luo, P., Szekely, P., Neches, R.: Management of Interface Design in Humanoid. In Ashlund S., Mullet K., Henderson A., Hollnagel E., White T. (eds.): Proceedings of INTERCHI'93. New York: ACM Press 1993 (pp. 107-114)
13. 48 Wiecha, C., Bennett, W., Boies, S., Gould, J., Green, S.: ITS: A Tool for Rapidly Developing Interactive Applications. ACM Transactions on Information Systems, Vol. 8, No. 3, 1990, p.204-236
14. Wiecha, C., Bennett, W.: Generating Highly Interactive User Interfaces. In Bice K., Lewis C. (eds.):.Proceedings of CHI'89. New York: ACM Press 1989 pp. 277-282.
15. Szekely P., Sukaviriya P., Castells P., Muthukumarasamy J., Salcher E. Declarative interface models for user interface construction tools: the Mastermind approach. In Bass L., Unger C. (eds.) // Engineering for Human-Computer Interaction, Proc. of EHCI'95. London, Chapman & Hall, 1995. P. 120-150.
16. Puerta, A.: The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. In: Vanderdonckt J. (ed.): Proceedings of CADUI'96. Namur: Presses Universitaires de Namur 1996 (pp. 19-36)
17. Szekely, P., Luo, P., Neches, R: Facilitating the Exploration of Interface Design Alternatives: The Humanoid Model of Interface Design. In Bauersfeld P., Bennett J., Lynch G. (eds.): Proceedings of CHI'92. New York: ACM Press 1992 (pp. 507-514).
18. Szekely, P., Luo, P., Neches, R.: Beyond Interface Builders: Model-Based Interface Tools. In Ashlund S., Mullet K., Henderson A., Hollnagel E., White T. (eds.): Proceedings of INTERCHI'93. New York: ACM Press 1993 (pp. 383-390).
19. Gribova V, Kleshchev A. Control of user interface development and implementation based on ontologies // Control Sciences, 2006. №2. P.58-62.
20. Gribova V., Kleshchev A. From an ontology-oriented approach conception to user interface development International //Journal Information theories & applications. 2003. vol. 10, num.1, p. 87-94
21. Gribova V. Automatic generation of context-sensitive help using a user interface project // Proc. of XШth Intern. Conf. "Knowledge-dialog-solution" – Varna, 2007. V.2. P. 417-422.
22. Puerta A. A model–based interface development environment IEEE Software, 14(1), July/August 1997. P. 41–47.
23. Puerta A.R. Issues in Automatic Generation of User Interfaces in Model-Based Systems. Computer-Aided Design of User Interfaces, ed. by Jean Vanderdonckt. Presses Universitaires de Namur, Namur, Belgium, 1996. P. 323-325.
24. Szekely P. Retrospective and Challenges for Model-Based Interface. 1996. http://citeseer.nj.nec.com/szekely96retrospective.html
25. Gribova V, Tarasov A, Chernyakhovskaya M. A system for intellectual support of patient examination controlled by the // Software & systems, 2007. №2. P. 49-51

## Author information

*Gribova Valeriya, Prof., Head of the Intelligent System Laboratory, Institute of Automation & Control Processes, Far Eastern Branch of the Russian Academy of the Sciences: Vladivostok, +7 (4323) 314001 gribova@iacp.dvo.ru, http://www.iacp.dvo.ru/is.*