Bull. Nov. Comp. Center, Comp. Science, 30 (2010), 81–88

An ontology-oriented approach to constructing user interfaces for an informational computational system to support innovations*

M.B. Ostapkevich, S.V. Piskunov, A.V. Veselov

Introduction

By now, supercomputers have become an efficient tool of mathematical simulation for both scientific and applied large-size problems. This makes it possible to thoroughly analyze such physical processes that otherwise could be too costly or even prohibitively expensive or time consuming to be analyzed in a usual way. This tool helps one considerable reduce the design time of Hi-Tech production. The demand for high performance computations constantly grows. And so does the list of applications and users, who are involved in an extremely wide range of problem domains. The efficient use of supercomputers is not a trivial goal. In order to achieve it, a network informational infrastructure has to be created. It should include not only a system and application software to provide the supercomputers-aided simulation but also software tools for the interactive communications between participants of an innovation project and tools for the access to vast repositories of general and professional information. The requirements imposed on the network Informational Computational System (ICS) for the support of innovation activity, its objectives, architecture and prototype implementation are discussed in [1–3]. The evolution of the system proceeds by the way of implementation of a feature that can be named mobility. Mobility implies an open architecture, support of multiple platforms, and what is more important, the ease of adaptation of a system to users' needs by users themselves. The steps of constructing a system that possess mobility are considered in [4–6].

The objective of this paper is to present a combined technology based at these steps and dedicated to the ICS construction and to describe the software tools to support this technology.

1. Step 1: selection of architecture of the mobile ICS

The selection of architecture is the principal step. It is based on the following considerations. The basic part of the system must be minimal. Its distribution package should be publicly available. At the same time, the

*Supported by Grant RAS 1.6.

basic part should be self-sufficient. There should be tools for the construction of auxiliary system modules. Also, it is desirable that the user could embed a wide range of imported procedures, packages and libraries into the system. All the auxiliary modules are stored into a dedicated repository on the ICS server. They are available to all the users of ICS. The system is assumed to operate in two modes. The user can gain access to the main ICS server maintained by its developers, or the user can download its distribution package and install the system on its own dedicated server. All this permits uniting the efforts of users and teams for the perfection of the ICS, extension of its functionality and elimination of errors.

Let us describe the architecture of ICS. The basis of the system is a kernel. It consists of DCMS library [7] and a minimal set of modules that form a skeleton of the system. These modules are created by the ICS developers and implement the functions for user administration and authentication, interfacing with Apache Web server, interfacing with QT GUI library, viewing and editing data objects in Web and GUI environments and managing application modular structure. The kernel also includes modules that implement data formats (textual and binary DCMS) and languages (such as HTML, SVG) and protocols (HTTP, SSH, SFTP). This basic set of modules can be extended by the user.

Further steps of the ICS development provide the users and developers with wide palette of possibilities for the system customization in order to meet their requirements. The assembly technology is widely employed at these steps. The ICS developers tried to impose a minimal possible set of requirements in order not to limit users in their selection of software tools for the construction of external ICS modules.

2. Step 2: using a conventional scheme of external module construction

2.1. The construction of external modules based on own ICS tools. This way of construction is fully based on DCMS and its assembly language. The DCMS library is a middleware built above OS (Win32, Linux). All the modules built on its basis consist of only event handlers. All intermodular interactions are performed by means of events. The event-driven interface allows one to embed new modules, substitute existing modules without need to recompile the kernel. The modules built on the DCMS basis are represented either by dynamically linked libraries (shared libraries in Linux) written in C/C++ or scripts written in the DCMS assembly language. New modules can be implemented and added to the system by any user who knows C/C++ or DCMS assembly language. The modules can be both system and application ones. For example, a user who uses KOrganizer to keep contact information wants to export this information to the ICS in-

formation space. In this case, the user builds a module for the KOrganizer data format support. Such a module, written in C, provides a handler for an event that arises at the moment of reading data in that particular format. The event-driven DCMS interface can integrate this module into the system without any modifications outside the module itself. Samples of the system modules added to the ICS are: a module for supporting a network protocol or accessing to a remote server; a module for interfacing with Java environment that allows one to execute a code written in Java from ICS; a module of VBScript code generation that helps one to generate pages with a better and more interactive design for the Web clients with Internet Explorer.

2.2. The use of loan modules. Another way of extending the ICS functionality is borrowing of loan modules (Web forum, Jabber server, Wiki). A modification of these modules is not required. In many cases, it would be even impossible. The developers of ICS have designed a few prototypes of docking modules and included them into the kernel. A docking module makes the interface functions of imported module visible in ICS. Its structure is typical, so in most cases the user can take one of the provided docking modules as a template and slightly modify it.

2.3. Limitations of the conventional module construction scheme. Above described two ways of module construction can be called classical. They represent a widespread approach of application construction [8]. Generally, it includes writing specifications, design of data structures and coding using a programming language. Such an approach implies that all the data types that an application should be capable to operate with have to be outlined by the developers at the stage of writing a specification. In this case, to operate with an object means to process it or to represent it in the user interface. After such an application is designed, it can operate only with objects from a predefined set of types. Thus, a modification of the source code and its recompilation are required when a new data type is added. Both the addition of new data types and modification of existing ones require the programmer's participance.

However, there is a certain difficulty: it is impossible to predefine all of the data types for all problem domains due to specific features of ICS. Thus, it seems to be evident that one more way of module construction is needed, the one that can be used by an ordinary user.

3. Step 3: using an ontology-oriented scheme of constructing the user interface

Let us make a few remarks before proposing a new way of the ICS module construction. The ICS must support the innovation process not only by

managing documents, but also by providing the access to computational resources. At the same time, the analysis of innovative activity shows that in either case the main kind of objects which is processed by the system is a *card*. It can be a card of information resource (book, article, reference), of a user (chief, participant), of a project or its stage, of simulating system invocation, of simulation results, etc. These cards are the most variable objects of the user interface. The relations between these objects actually define the innovation process that can be represented in the form of ontology. Ontology means a model of a problem domain that is written in a formal language of knowledge representation [9]. That is why the ontology-oriented approach to the construction of the user interface is proposed as a component of the ICS mobility support. This approach is still in its early stage and has not gained importance yet. However, it is assumed to be promising [10–12]. Its advantage is concluded in its ability to provide thorough modifications and constructing the new application modules (including those built at the second step) without coding.

3.1. Description of the ontology-oriented approach. An essential feature of this approach is that the description of application objects is given in an ontology rather than coded in a programming language. An application itself is extended by an additional module (a code generator) that allows one to obtain an object structure and code for their representation in the user interface at runtime. All the data that describe how an application operates are divided into the two parts: a constant part, which is coded in an application in the form of procedures and a variable part that is described by ontology. Using this approach to constructing an application, the user is capable of extending a set of supported types of objects without need to write a code. This approach helps in cutting the time taken for designing the user interface. It also makes it easier to modify it later, because the developer deals only with a declarative description. This means that the user can extend applicability of an application on his own, edit and create a new object types descriptions in an ontology editor.

3.2. Description of the ICS user interface. Let us adapt the above approach to constructing the user interface for ICS. The interface is described in the form of two related ontologies: the problem domain ontology (currently, a detailed ontology of an innovation process [4, 5]) and ontology of the user interface.

The innovation process ontology is built with orientation to serve a specification for functions to be implemented by the system. An ontology can be extended by subontologies that reflect peculiarities of a certain problem domain. For example, a node “simulation” might be substituted by the ontology of access to resources of the Siberean Supercomputing Center

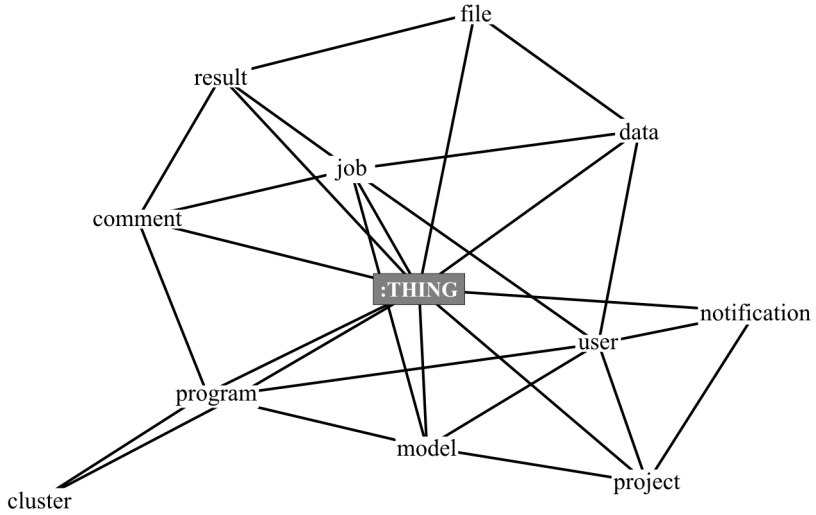

Figure 1. A sample fragment of ontology of access to the SSCC resources

(SSCC) SB RAS (developed by student A.V. Beloglazov, M. Sci.). This ontology describes objects, events and facts that are used in the simulation process. A sample fragment of the ontology is presented in Figure 1.

When the ontology of access to the SSCC is embedded, the class Simulation is inherited from the class Thing. Let us take the node Program, as an example. It has many models. It belongs to a cluster and to a user that has uploaded its code.

The second ontology is dedicated to the description of graphical representation of the objects of the system. The description is a graph. Its nodes are primitives (indivisible elements of interface). The arcs represent relations between them [13]. Samples of primitives are: a text string, a text edit line, a cell of a table, a string of a table, a checkbox, a menu item, a picture. The description also contains fragments of code for their visualization. Fragments can be written in a programming language or in a markup language. The graphical representation of objects is described by a hierarchical structure. Each level of hierarchy contains a description of intermediate representation, which is detailed at lower levels. The intermediate representation is defined by a primitive or a composition of primitives and their parameters. In addition, the problem domain ontology is used in order to determine object types. Thus, the name of a property can be specified at the intermediate level within a represented object, and a graphical representation of a relevant part of an object should be detected by the type of property.

The existence of two separate ontologies helps to make description of visual representation for an object dependable on its description in a problem domain. It also provides the ability to assemble a visual representation from

primitives. The division of ontologies makes it possible to combine the same ontology of a problem domain with multiple ontologies of the graphical representation for different platforms and languages. Ontologies are designed according to each other and can be united if necessary as they share one namespace. RDF/RDFS is used as a language of ontology description and Protege as an ontology editor. Query processing is performed by an open source Rasqal RDF Query Library with RDQL query language.

3.3. Automated code generation for the user interface. Both ontologies are input into a code generator. Currently two generators are implemented. One is built as an external module on the DCMS basis [4, 5]. Another one is built as an external module on the basis of Perl and HTML. The GUI generation scheme is presented in Figure 2 with a user's card as a sample.

Let us note that both implementations are made using the same scheme. This demonstrates the independence of ontology-oriented approach of a language of implementation and an output language of interface description.

The generator builds a fragment for the user interface description based on the problem domain and the user interface ontologies. An object can belong to a type that is described in the problem domain ontology. The object processing would mean, for example, its modification or deletion. All

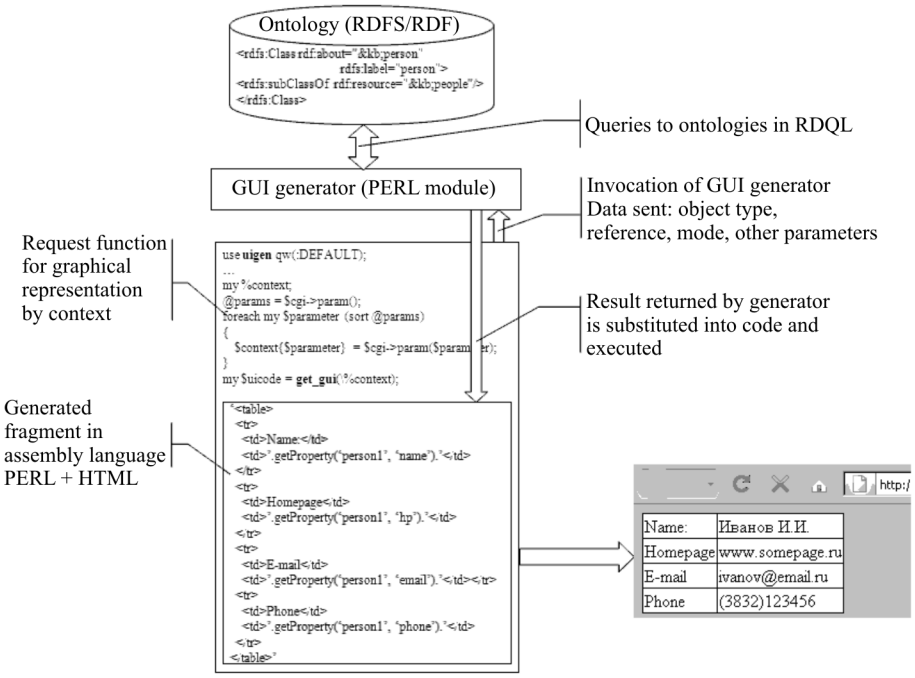

Figure 2. The user interface generation

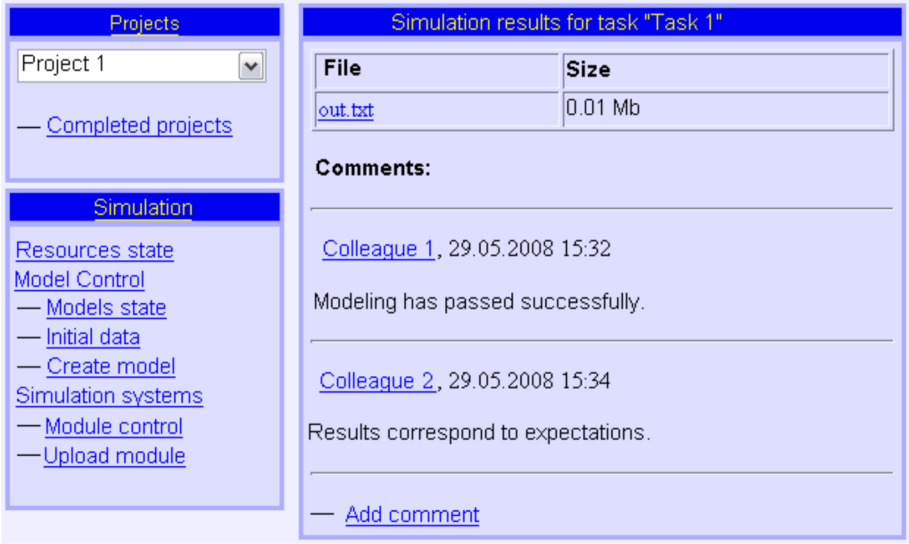

Figure 3. The user interface generation

kinds of the object processing must be described in one of the ontologies for all supported types. Each of these kinds is implemented by the so-called visual modes for both GUI and Web interfaces. From the practical point of view this means that the user has means to construct his workspace including desktop, toolbars of operations, subwindows to view objects or their directories in a versatile set of modes (list, tree, chronology), subwindows for the access to the SSCC resources(project windows, simulating system upload, its compilation and execution). For example, a fragment of interface depicted in Figure 3 can be built.

Conclusion

The proposed assembly technology and especially its part that is based on the ontology-oriented technology permits the user to customize the system with a particular problem domain or software environment. The use of the ontology-oriented approach helps the user to assemble his own applications and customize the existing ones and to involve computational supercomputer resources for some of the stages of an innovation project. The user is also given a choice of adding his modules or information resources to the publically available repository of the web site of the developers of the system.

References

[1] Alekseev A.S., Ostapkevich M.B., Piskunov S.V. About a project of informational computational system of support of innovation activity // Optimization. Control. Intellect. — 2005. — Vol. 2, No. 10. — P. 203–210 (In Russian).

[2] Zagorulko Yu.A., Piskunov S.V., Borovikova O.I., Ostapkevich M.B. Distributed Internet system of formation and support of innovation projects // Proc. 8th Intl. Conf. "Problems of Control and Simulation in Complex Systems". — Samara, 2006. — P. 427–432 (In Russian).

[3] Alekseev A.S., Kratov S.V., Ostapkevich M.B., Piskunov S.V. Siberian network system for the support of innovation activity // Proc. Conf. "Scientific Service in Internet: Multicore World, 15 years of RFBR". Novorossijsk, 24–29, September 2007. — Moscow: Moscow State University, 2007. — P. 352–354 (In Russian).

[4] Veselov A.V., Ostapkevich M.B., Piskunov S.V. Automated generation of users' interfaces for the network informational computational system // Proc. 7th Intl. Conf. "Perspectives of System Informatics". Seminair "Knowledge Intensive Software". — Novosibirsk: Siberian Scientific Publisher, 2009. — P. 84–90 (In Russian).

[5] Veselov A.V. Using ontologies for the construction of graphical user interface // Proc. Conf. of Young Scientists. — Novosibirsk: ICM&MG SB RAS, 2008. — P. 18–28 (In Russian).

[6] Veselov A.V., Kratov S.V., Ostapkevich M.B., Piskunov S.V. The design of users' interfaces of network informational computational system on the basis of model-oriented approach. // Proc. 5th Intl. Conf. Seminair "Problems of Optimization of Complex Systems". Ser. Informatics. — Iss. 9. — Novosibirsk, 2009. — P. 34–39 (In Russian).

[7] Ostapkevich M.B. The DCMS library for open architecture application // Bull. Novosibirsk Comp. Center. Ser. Comp. Science. — Novosibirsk, 2004. — Iss 21. — P. 99–111.

[8] Sommerville I. Software Engineering. — Moscow: Williams, 2002 (In Russian).

[9] Dobrov B.V., Ivanov V.V., Lukashevich N.V., Solovjev V.D. Ontology and Thesurus. — Kazan: Kazan State University, 2006 (In Russian).

[10] Molina P.J. A review to model-based user interface development technology // Proc. 1st International Workshop on Making Model-Based User Interface Design Practical: Usable and Open Methods and Tools. — 2004. — (http://ftp1.de.freebsd.org/Publications/CEUR-WS/Vol-103/molina-moreno.pdf).

[11] Molina J.P., Lopez M.G., Lopez P.G. Model-based design and new user interfaces: current practices and opportunities // CEUR Workshop Proceedings. — 2004.

[12] Gribova V., Kleschev A. A. Conception of development of user interface on the basis of ontologies. Part 1. Tools for user interface development (an overview). The main ideas of the approach / Technical report. — Vladivostok, 2003 (In Russian). — (http://www.iacp.dvo.ru/es/publ/179_1.pdf).

[13] Veselov A.V. The evolution of ontology-oriented approach to the construction of graphical interfaces // Proc. Conf. of Young Scientists. — Novosibirsk: ICM&MG SB RAS, 2009. — P. 34–42 (In Russian).