The methods and the IACPaaS Platform tools for semantic representation of knowledge and development of declarative components for intelligent systems

1st Valeria Gribova
Intelligent systems lab.
IACP FEB RAS
Vladivostok, Russian Federation
gribova@dvo.ru

2nd Alexander Kleschev
Intelligent systems lab.
IACP FEB RAS
Vladivostok, Russian Federation
kleschev@dvo.ru

3rd Philip Moskalenko
Intelligent systems lab.
IACP FEB RAS
Vladivostok, Russian Federation
philipmm@dvo.ru

4th Vadim Timchenko
Intelligent systems lab.
IACP FEB RAS
Vladivostok, Russian Federation
vadim@dvo.ru

5th Leonid Fedorischev
Intelligent systems lab.
IACP FEB RAS
Vladivostok, Russian Federation
fleo1987@mail.ru

6th Elena Shalfeeva
Intelligent systems lab.
IACP FEB RAS
Vladivostok, Russian Federation
shalf@dvo.ru

***Abstract*—The paper discusses the problem of ensuring the viability of intelligent systems – systems with declarative knowledge bases. Software tools for the development of such systems that implement mechanisms for viability improvement are considered. These mechanisms are based on the construction of each component according to its declarative model, which is specified in a unified language for model description.**

***Index Terms*—intelligent systems, software system maintenance, software system viability, development tools**

I. Introduction

Ensuring the viability of software systems (SS) is one of the key problems in software engineering. The term viability refers to the SS sustainability (performance preservation) to changes in the environment and the ability to evolve during the lifecycle [1], [2], [3]. Viability is directly related to the SS transparency, which is characterized by three main properties: accessibility, clarity and relevance of the information and components of the SS to interest groups [4].

Among the many SSs, the class of intelligent systems is distinguished. They are systems with knowledge bases (KBS), which are actively used to solve various scientific and applied problems. Their architecture, among the traditional components – databases, business logic (solver) and user interface, contains an additional component – the knowledge base. At present, one can say that KBSs have reached the phase of maturity. But the problem of ensuring their viability is acute, since the development team of such systems includes knowledge engineers and domain experts in addition to programmers and interface designers. This class of SS is characterized by a continuous improvement of knowledge bases, and an occasional improvement of problem solving method and of an output explanation.

Despite the development of tools for creating of systems of this class, the problem of their viability remains urgent:

- domain experts still cannot independently (without intermediaries like knowledge engineers and programmers) build and maintain knowledge bases;
- part of the domain knowledge is "embedded" into the problem solver, which makes their modification more difficult, and its structure is hard to understand;
- the UI does not adapt to the requirements of users, of the platform, of the domain, it usually has a "firm" structure built into the problem solver.

These drawbacks make it necessary to use additional specialized mechanisms to ensure the viability of this class of systems. The aim of the work is to describe new models and methods aimed at providing the viability of the KBS.

II. Review

There are three main types of KBS development tools: programming languages, shells and specialized tool systems. General purpose programming languages (Python, C#, Java, etc.) or specialized ones (LISP, Smalltalk, FRL, etc.) are universal development tools. In [5] it is noted that the complexity of intelligent system development with the use of programming languages is so great that it is practically unaffordable.

Problem-independent and specialized shells greatly simplify the creation of the KBS, however, they limit the possibilities of

The work is carried out with the partial financial support of the RFBR (projects nos. 19-07-00244, 18-07-01079) and by PFI "Far East" (project no 18-5-078).

their evolution: they have a pre-defined solver and an embedded UI that cannot be modified if requirements change. Also, the disadvantages of specialized shells include limitations on the field of their use, and disadvantages of the problem-independent ones – their "non-transparency" primarily for domain experts who cannot independently (without knowledge engineers) form and maintain a knowledge base as part of the knowledge is built into the logical inference machine [5], [6].

Specialized tool systems are focused on a wide class of KBS. Typical representatives of tool systems are: Level5 Object, G2, Clips, Loops, VITAL, KEATS, OSTIS, AT-technology etc. [5], [6], [7]. They differ by the knowledge representation formalisms, by the used output mechanisms, and by the tools for UI forming.

Looking at these tools from the point of view of the viability of KBS created with their use, it can be noted that the evolutionary development of tool systems is focused on achieving this important goal in one way or another. It is primarily reflected in the tools which support the knowledge base (KB) creation, which is one of the most difficult stages of development of such systems, as well as in methods of coupling of KB with a problem solver.

According to [6], the most common model of knowledge representation remains the rule-based one. But by now, the trend of production model systems amount reducing is obvious. Given the need for alternative knowledge representation models, many development tools offer a mixed mechanism for their presentation. For example, LOOP and G2 use rules and object-oriented representation, ART – rules, frame-like and object-oriented models for declarative knowledge. However, the proposed types of representation are not oriented at independent (without knowledge engineers) formation and modification of knowledge by domain experts.

For the formation of knowledge bases one can consider specialized tools based on ontologies: Protégé, OntoEdit, GrOWL, Graphl, RDFGravity, WebVOWL, Ontolingua, OilEd, WebOnto, WebODE [9]. However, they usually implement an object-oriented paradigm of knowledge representation, incomprehensible to most domain experts. A question of their integration with a problem solver and UI also remains open. In accordance with the knowledge representation model, an appropriate mechanism for implementing the solver (reasoner) is proposed. If there are several models supported by the system, respectively, several solver implementation mechanisms and languages are supported. E.g., the SWORIER system uses a reasoning mechanism based on ontologies and rules. Such solutions, on the one hand, are aimed at giving the possibility of choosing the most adequate knowledge representation model and the corresponding solver, but on the other hand, the transparency of such systems remains quite low.

The support of UI development is carried out in several ways. The developer is offered a set of tools provided by the toolkit, for example, [7]. This may be a specialized programming language or tools similar to interface builders, offered by various CASE-tools: a set of WIMP interface elements that a user can define, specify their properties and associate them with commands (user and / or solver actions) and / or data (input or output). The interaction scenario in this case is embedded into the solver. Interface development can be carried out using the language in which the solver is designed. Interaction with different libraries provided by the toolkit is possible.

Thus, the most flexible tool for KBS implementation are specialized tool systems, as they allow one to implement different classes of KBS. However, the problem of the viability of this class of systems is still far from a final solution. Therefore, the search for new, improved mechanisms for viability improvement of such systems remains an urgent task.

III. Basic principles of KBS vitality

The viability of SS and KBS in particular is largely determined by their transparency. One of the main attributes of a transparent SS is clarity for interest groups. For KBS such groups are:

- domain experts who are responsible for the development and maintenance of KB,
- programmers who create and maintain a solver,
- interface designers who implement the UI of a solver and the UI for KB editors.

For KBS, it is fundamentally important to use relevant knowledge that must be formed and maintained by domain experts or inductively (but in the latter case its representation should be intelligible to experts). This is possible only if the knowledge representation language is focused on the class of problems to be solved, and its terminology is familiar to experts. To ensure the transparency of the solver, its structure and modules should be clear to the maintainer. This is possible if most of the solver is presented declaratively (which allows to control solvers with the help of editors), and domain knowledge is not included in the solver. UI transparency can be ensured, firstly, by providing users with different types of UI which suite the model for presenting information most appropriately, secondly, by separating the data from the logic of its processing and its presentation method. The latter also provides separate modification of each of the components.

To implement these requirements, the following basic solutions are proposed:

- common principles for creating KBS components;
- a two-level approach to the formation of components: first, a structural declarative model (component ontology) is formed, then the necessary component of the KBS is created by it;
- unified language and editor for creating models of all components;
- automatic generation of editors for creation of components basing on their models;
- implementation of instrumental and applied intelligent systems as cloud services.

All proposed solutions are based on a model description language that allows one to describe arbitrary models oriented and adapted to the terminology of developers, with the transition from general concepts to detailed ones. The models of

the components of intelligent services are formed in the model description language and are represented in the form of a connected marked rooted hierarchical binary digraph. The markup defines the semantics for the rules of formation (creation and modification) of components, imposing restrictions on their structure and content [10], [11].

A. Development and maintenance of the knowledge base

In accordance with the two-level approach to the formation of components of the KBS, at the first stage a specialized model of knowledge (data) representation is formed – the ontology of knowledge, which takes into account the specifics of the organization of knowledge and data in a given domain. Further, according to the model of knowledge (data), the component editor generator builds an editor of the knowledge base / database (see Fig. 1). Domain experts have the opportunity to form knowledge and data bases in terms of their concept systems but not in terms of some fixed knowledge and data representation language.

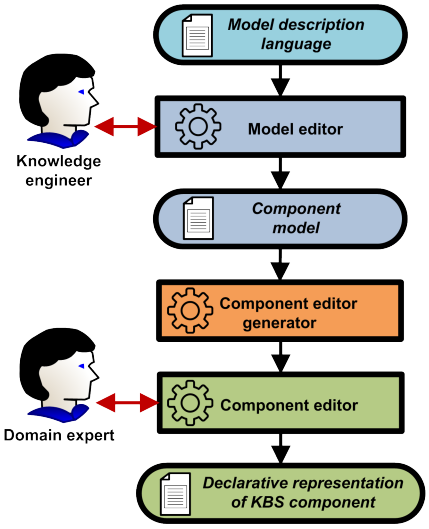

Figure 1. Knowledge base formation process.

B. Development and maintenance of problem solver

The problem solver is a set of agents that interact with each other by the exchange of messages. In accordance with the two-level approach, developers are offered unified agent and solver models for all services. To organize the launch of solvers with specific sets of input and output data, the cloud service model is also defined. To increase the transparency of the imperative parts of the agent and of the message template after the description of their declarative part is specified, their source code sketch in the Java language is generated. The developed imperative code is associated with the corresponding vertex of the agent (or message template) model.

C. UI development and maintenance

The development of an interface of intelligent services implies the development of a web interface. The interface design is based on the Model-View-Controller (MVC) pattern. Its fundamental principle consists in the separation of data, the logic of its processing and the way it is presented in order to provide independent modification of each component. The projection of this pattern on the interface model is as follows. The Model component includes: an abstract UI model containing a description of the structure of standard WIMP interface elements (simple and container ones) and a way for their recursive organization into a single nested structure, as well as a software interface (API) for generating fragments of abstract interfaces. The View component is implemented by the system *View* agent. Its main function is to create a description of a specific interface based on the description of an abstract interface and on rules of mapping from later to former. The Controller component is represented by agents which play the role of an *Interface Controller* being a part of various problem solvers. These agents interact with the *View* agent by exchanging messages using specific templates and implement necessary processing logic.

IV. Conceptual architecture of development tools

A comprehensive solution to the problem of the intelligent service viability also means providing the viability of the tools with which the service is created and maintained. As a rule, the toolkit is maintained by its developers, but it must also be maintained by the KBS developers [12]. For the successful implementation of this requirement, a three-tier toolkit architecture is proposed, consisting of the *Toolkit Core*, the *Basic Toolkit* and the *Extensible Toolkit*.

The *Toolkit Core* implements the basic principle of the construction of all components and includes the model description language, the model editor, the generator of component editors. The declarative language for model description is used to create component models, regardless of component's purpose. The component model editor allows developers to create models in simple and convenient way. The generator is designed to automatically build declarative component editors by component models (ontologies). It is responsible for generating the UI and the component formation scenario which includes checking the context conditions specified in the model and the completeness of the component. The *Toolkit Core* is sufficient for creating and controlling all intelligent service declarative components by their models.

The main task of the *Basic Toolkit* is to provide the developer with a set of tools for creating software components, assembling and binding them with information components, launching, and organizing infrastructure at all levels of the toolkit. Since all components are formed according to their structural declarative models, this level of toolkit includes component editors that are generated automatically. In addition to the elements mentioned above, it contains external software for creating the intelligent service UI and the imperative part of the aforementioned components.

The *Extensible Toolkit* is primarily intended for KBS developers, who can expand it with new convenient tools for

maintenance of KBSs developed by them and with specialized or universal shells of expert systems. The expansion may be carried out using the *Toolkit Core*, the *Basic Toolkit*, as well as with the tools and instrumental mechanisms of the *Extensible Toolkit* itself. This way, recursive use of its developed components is achieved.

The three-tier architecture forms the basis of the IACPaaS cloud platform (https://iacpaas.dvo.ru) [13], which is available for use by all developers of KBSs and their components. To date, portals of knowledge on medicine, mathematics, autonomous uninhabited underwater vehicles, diagnostics of crops, information security, educational psychology, and programming technology have been created on the platform.

V. Conclusion

The paper considers mechanisms aimed at ensuring the viability of one class of software systems – systems with knowledge bases. Their main difference from systems of other classes is the presence of a knowledge base, which is subject to continuous changes during the life cycle, and which must be created and maintained by domain experts. The proposed solutions are based on the model description language developed by the authors, which provides the tools for model specification in the form of connected labeled rooted hierarchical binary digraphs with possible loops and cycles. The KBS components which are built on the basis of the model have a unified representation and internal storage format, and are provided with a standardized and extensible set of software interfaces for uniform access to them. Domain experts get the opportunity to form knowledge and data bases in terms of their concept systems but not in terms of fixed the knowledge and data representation language. The problem solver architecture includes declaratively represented software units, which can constitute a dynamic configuration, interact by message exchange, and the structure of which is also described using a declarative model. A unified language and a uniform internal representation of both models and components, specified by them, allow the use of common principles for editor generation basing on the KBS component type. All proposed ideas are implemented on the IACPaaS cloud platform. Herewith, tool services providing support for the development technology are created on the same principles as the applied services.

At the same time, the experience of platform usage and the availability of user feedback has set a number of new scientific problems, including the creation of language-oriented queries to knowledge bases, methods for creating adaptive user interfaces of various types for KBS problem solvers and knowledge editors. Their solution will end up as additional increasing of the viability for this class of systems.

References

[1] Pressman R.S. Software engineering: a practitioner's approach. McGraw-Hill, 7th ed., 2010. 930 p. ISBN: 0073375977.

[2] Kryazhich O.A. Obespechenie zhiznesposobnosti informacii vo vremeni pri ee obrabotke v SPPR [Ensuring the viability of the information during its processing in decision support systems]. Matematicheskie mashiny i sistemy [Mathematical Machines and Systems], 2015, no. 2, pp. 170–176.

[3] Chernikov B.V. Upravlenie kachestvom programmnogo obespecheniya [Software quality management], Moscow: Forum-Infra-M, 2012. 240 p.

[4] Yu-Cheng Tu. Transparency in Software Engineering. A thesis submitted in fulfillment of the requirements of Doctor of Philosophy in Electrical and Electronic Engineering. The University of Auckland. New Zealand, 2014. 337 p.

[5] Rybina G.V. Intellektual'nye sistemy: ot A do YA. Seriya monografij v trekh knigah. Kn. 3. Problemno-specializirovannye intellektual'nye sistemy. Instrumental'nye sredstva postroeniya intellektual'nyh system [Intelligent systems: A to Z. A series of monographs in three books. Book 3. Problem-specialized intelligent systems. Tools for building intelligent systems], M.: Nauchtekhlitizdat, 2015. 180 p.

[6] Emmanuel C. Ogu, Adekunle, Y.A. Basic Concepts of Expert System Shells and an Efficient Model for Knowledge Acquisition. Intern. J. of Science and Research Intern. Journal of Science and Research (IJSR), India Online ISSN: 2319-7064, 2013, vol. 2, issue 4, pp. 554–559.

[7] Rybina G.V. Intellektual'naya tekhnologiya postroeniya obuchayushchih integrirovannyh ehkspertnyh sistem: novye vozmozhnosti [Intelligent technology for construction of tutoring integrated expert systems: new aspects]. Otkrytoe obrazovanie [Open Education], 2017, vol. 21, no 4, pp. 43–57.

[8] Golenkov V., Gulyakina N., Grakova N., Davydenko I., Nikulenko V., Eremeev A., Tarasov V. From training intelligent systems to training their development tools. Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems], 2018, pp.88–98.

[9] Ontology Tools. Available at: http://wiki.opensemanticframework.org/index.php/Ontology_Tools (accessed 2019, Jan).

[10] Gribova V.V., Kleshchev A.S., Moskalenko F.M., Timchenko V.A. A Two-level Model of Information Units with Complex Structure that Correspond to the Questioning Metaphor. Automatic Documentation and Mathematical Linguistics, 2015, vol. 49, no. 5, pp. 172–181.

[11] Gribova V.V., Kleshchev A.S., Moskalenko F.M., Timchenko V.A. A Model for Generation of Directed Graphs of Information by the Directed Graph of Metainformation for a Two-Level Model of Information Units with a Complex Structure. Automatic Documentation and Mathematical Linguistics, 2015, vol. 49, no. 6, pp. 221–231.

[12] Musen M. The protégé project: a look back and a look forward. Newsletter AI Matters, 2015, vol. 1, iss. 4, pp. 4–12.

[13] Gribova V., Kleschev A., Moskalenko P., Timchenko V., Fedorischev L., Shalfeeva E. The IACPaaS cloud platform: Features and perspectives. Computer Technology and Applications (RPC), 2017 Second Russia and Pacific Conference on. IEEE, 2017, pp. 80–84.

МЕТОДЫ И СРЕДСТВА ПЛАТФОРМЫ IACPAAS ДЛЯ СЕМАНТИЧЕСКОГО ПРЕДСТАВЛЕНИЯ ЗНАНИЙ И РАЗРАБОТКИ ДЕКЛАРАТИВНЫХ КОМПОНЕНТОВ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

В.В. Грибова, А.С. Клещев, Ф.М. Москаленко, В.А. Тимченко, Л.А. Федорищев, Е.А. Шалфеева

Лаборатория интеллектуальных систем,

Федеральное государственное бюджетное учреждение науки Институт автоматики и процессов управления Дальневосточного отделения Российской академии наук,

г. Владивосток, Российская Федерация

В работе обсуждается проблема обеспечения жизнеспособности интеллектуальных систем – систем с декларативными базами знаний. Рассмотрены инструментальные программные средства для разработки систем данного класса, реализующие механизмы повышения их жизнеспособности. Эти механизмы основаны на построении каждого компонента по его декларативной модели, специфицируемой на едином языке описания моделей.