The technology for the development of viable intelligent services

Valeria Gribova, Philip Moskalenko,
Vadim Timchenko, Elena Shalfeyeva
Lab. of Intelligent Systems named after A.S. Kleshchev
Institute of Automation and Control Processes FEB RAS
Vladivostok, Russian Federation
gribova@iacp.dvo.ru, philipmm@iacp.dvo.ru,
vadim@iacp.dvo.ru, shalf@iacp.dvo.ru

***Abstract*—The paper presents a technology for the development of intelligent multi-agent services. This technology is put to reduce the labor-intensiveness of intelligent cloud application development and maintenance. The key aspect of the technology is an independent development of knowledge bases, a user interface and a problem solver as an assembly of agents, and their integration into an intelligent service. The principal specific of the technology are: two-level ontological approach to the formation of knowledge and data bases with a clear separation between the ontology and knowledge base (database) formed on its basis; the ontology-based specification of not only the domain terminology and structure, but also of the rules for knowledge and data formation, control of their integrity and completeness; division of all software components into declarative and procedural parts; unified semantic representation of all declarative components (ontologies, knowledge and data bases, declarative parts of software components), etc. The proposed approaches are supported by the IACPaaS cloud platform where expert-oriented formation of each service component with an appropriate tool is provided.**

***Keywords*—intelligent system, multi-agent system, intelligent software development technology, hierarchical semantic network, cloud platform, cloud service**

I. Introduction

Development and maintenance of an intelligent system (IS) with knowledge are hard and labor-intensive processes. This is primarily due to the fact that a knowledge base (KB) is a part of such a system's architecture which brings in some specificies. That's why special tools are used. The typical representatives are: Level5 Object, G2, Clips, Loops, VITAL, KEATS, OSTIS, AT-technology, RT Works, COMDALE/C, COGSYS, ILOG Rules, Protégé and others [1], [2], [3], [4], [5]. The considerable differences between them are determined by the formalisms used to represent knowledge, methods and tools for their acquisition, forming and debugging, the used inference mechanisms, and also technologies for IS with a KB (hereinafter KBS – knowledge based system) development and maintenance.

However, the problem of creating tools for development and maintenance of KBS is far from being solved. Scientific publications mention the following main problems that need to be dealt with. First of all, there is still an open issue of including domain experts in the process of developing and maintaining knowledge bases, ensuring their real impact on the quality of computer systems being developed. Experience in developing complex computer systems shows that the intermediation of programmers between designed computer systems and experts significantly distorts the contribution of the latter ones. When developing a new generation of tools, it is not programmers who should dominate, but experts who are able to accurately represent their knowledge [6], [7]. The developed models and methods of knowledge acquisition solve the problem of their initial formation, however, as noted in [8], it is the maintenance phase that is the most complex and significant. The KB is a component of KBS that changes much more often than other components, so expert-oriented knowledge creation and maintenance tools providing is an important and relevant task. KB maintenance implies its refinement or improvement, which at the same time does not break the performance of the IS (i.e., a change in the knowledge base should not lead to the need of changing the solver and its user interface). Among the problems of existing tools for creating KBS, the following is also worth mentioning: the use of various approaches and mechanisms for creating knowledge bases, solvers and interfaces, complex linking of these components, or vice versa, the lack of a clear separation between KBS components, which makes it difficult to reuse solutions when creating other KBS. Our long-term experience in creating practically useful KBS for solving problems in various domains has shown an urgent need to construct problem-oriented shells with accompanying instrumental support.

The key requirement for any complex software system, including KBS, is its viability, which is implemented in software engineering through architectural solutions (separation into loosely coupled components with logically clear functions), declarative representation of software components of a system, automation of code fragments generation, reuse of components, separation of competencies between developers of different types. One of the well-known solutions for implementing the requirement to involve domain experts in the KBS development process is the ontology-based (metainformation-based) KB formation, using a semantic knowledge repre-

This work was partially supported by the Russian Foundation for Basic Research (RFBR) (grant numbers 19-07-00244, 20-07-00670)

sentation model (the knowledge base is separated from the ontology) [9].

In software engineering, an agent-based approach is actively used to create viable software systems [10], which in comparison with object-oriented programming has potentially greater flexibility, gives the possibility of agent reuse, and simplifies parallelization (which is important for IS as many tasks have great computational complexity [11]). However, the issue of creating a comprehensive technology that supports the development of all KBS components with ontological knowledge bases remains open.

To address these challenges in the development and maintenance of viable KBS we have proposed a concept of the development toolkit [12], which supports the following technological principles:

- knowledge base, problem solver and user interface (UI) are developed separately;
- a single language is used to describe the ontology of knowledge (metainformation for the knowledge base) and models of all declarative components (their metainformation);
- a unified declarative semantic (conceptual) representation of all information (ontologies, knowledge bases, databases) resources and software components of KBS is used – a labeled rooted hierarchical binary digraph with possible loops and cycles (thus universality of their software processing and user editing can be achieved);
- formation and maintenance of knowledge bases is carried out by domain experts and is based on knowledge ontologies;
- UI of the knowledge base editor is generated for experts on the basis of ontology (metainformation);
- method of problem solving is divided into subproblems, each one is solved by a correspondent software agent, either a reflective or a reactive one [13] – a component of a problem solver;
- software agents are provided with an API for access to information resources;
- KBS is available as a cloud service.

This paper describes the technology for the development of a KBS as a cloud multi-agent service, using such toolkit.

II. Intelligent service development technology

Technological foundations supported by the proposed toolkit concept are consistent with the general trend when KBS development is based on the use of methods and tools for ontological modeling of design and specification processes of developed systems, i.e. ontological engineering tools (Ontology-Based (-Driven) Software Engineering [14]); reusability of ready-to-use solutions (components); involving stakeholders in the development process through knowledge portals.

Methods of formalizing knowledge based on the domain ontology and ontologies of known intelligent problems allow us to create structured and at the same time understandable to domain experts knowledge, as well as systematization of terms.

Further, knowledge, formalized on the basis of an ontology, are either integrated with ready-to-use problem-oriented software solvers, or require the creation of new software components. Thereby the proposed technology for the development of intelligent services consists of the following processes: *assembly of an intelligent service from components* and possibly *development of those components* (Fig. 1). Components are represented by rectangles, and activities – by rounded rectangles. The components are information ones (metainformation and information) and software ones (a problem solver, its agents, message templates for their interaction, UI). Symbols on arcs have the following meaning. **sel** – searching for and selecting the appropriate component (to which an arc is pointing). **[ ]** – optionality, i.e. the pointed activity is performed if the required component is absent or if the condition (italic text on the arc) is true. **+** ("plus") – multiplicity, meaning that the selection or creation (development) of the corresponding component can be performed more than once.

III. Intelligent service development

An intelligent *service* consists of a *problem solver* (which has been integrated with *formal parameters* and with a *UI*) and actual parameters (*input actual parameters* – information resources accessible only for reading and *output actual parameters* – information resources accessible for CRUD modification). Distinction of service components into information and software ones pursues the following objectives:

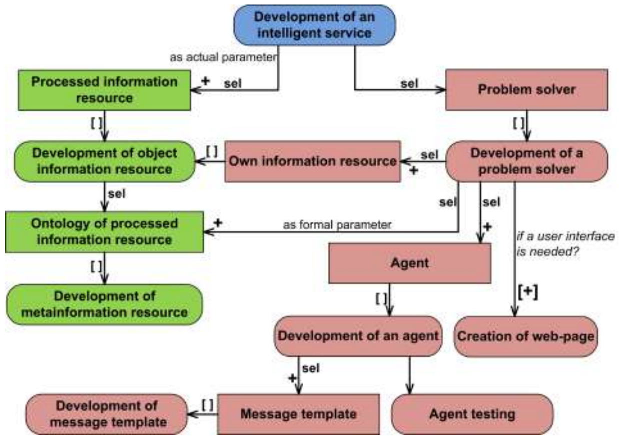

Development of an intelligent service
as actual parameter
+ sel
sel
Processed information resource
Problem solver
[ ]
[ ]
Development of object information resource
[ ]
Own information resource
sel
+
Development of a problem solver
sel
sel
sel
Ontology of processed information resource
+
as formal parameter
+
if a user interface is needed?
Agent
[ ]
Development of metainformation resource
[ ]
[+]
Development of an agent
Creation of web-page
sel
+
Development of message template
[ ]
Message template
Agent testing

Figure 1. Intelligent service development technology.

- independent development by different groups of specialists;
- reuse – the same problem solver can be bound with various information resources and vice-versa.

In both cases the compatibility (between those two types of components) is provided at the level of *formal parameters* of the *solver*.

A *formal parameter* of the *problem solver* is an information resource which represents metainformation (ontology), an actual parameter is one of the information resources which represent information and which are formed in terms of this metainformation.

The declarative specification of a *service* is formed in two stages:

1) creation of a new information resource of “service” type with setting of its name and *Service structure* information resource (representing a language for declarative specification of platform services) as metainformation;
2) creation of content of this new information resource (service assembly) with the use of *Editor for digraphs of information* [15], where the process of editing is controlled by metainformation *Service structure* and consists of the following:
   - a link to the *problem solver* (to root vertex of digraph which represents it) is created,
   - for each *formal parameter* of the *solver* (in order of its appearance in the description of solver) a link to the corresponding actual parameter (to root vertex of digraph which represents it) is created.

IV. Development of knowledge bases and databases

A. General description

Network (graph) structures are now widely used as a visual and universal mean for representing various types of data and knowledge in different domains. In principle, with the help of such structures that are best suited for explicitly representing associations between different concepts, it is possible to describe any complex situation, fact or problem domain. At the same time, as noted, for example, in [16] various kinds of information (data and knowledge), regardless of the concrete syntax of the language for their representation, in the abstract syntax, in the general case, can be represented as (multi-) graphs, possibly typed.

In accordance with a 2-level approach for formation of information resources [17], [18], two types of them are distinguished by the abstraction level of represented information. They are information resources which represent ontology (i.e. metainformation – abstract level) and information resources which represent knowledge and data bases (i.e. information – object level).

B. Object information resources development

Development of an *information resource* which is processed by an intelligent service and represents information requires another information resource which represents its metainformation to be present in the storage. Otherwise it must be developed as described in the next subsection. The development of an *information resource* consists of two stages:

1) creation of a new information resource with setting of its name and metainformation;
2) formation of its content by means of the *Editor for digraphs of information* where the process editing is controlled by set metainformation.

During the work of the *Editor for digraphs of information* it forms and maintains a correspondence between the arcs of digraphs of information and metainformation. Formation is carried out in “top-down” way: from vertices which are composite concepts to vertices which are atomic concepts. This process starts from the root vertex. In this case the user doesn’t have to sharply envision and keep in mind the whole structure (connections between vertices) of the formed digraph as in the case of “bottom-up” way.

The UI of the *Editor for digraphs of information* is generated by the metainformation. This implies that as the latter one describes an ontology for knowledge or data in some domain so its experts can create and maintain knowledge bases or databases in terms of their customary systems of concepts (without mediators, i.e. knowledge engineers).

C. Metainformation resources development

The development of a *metainformation* resource consists of two stages:

1) creation of a new information resource with setting of its name and metainformation (in such case it is an information resource which contains the description of the *language for metainformation digraph representation*);
2) formation of its content in a “top-down” way (starting from the root vertex of the digraph) with the use of the *Editor for digraphs of metainformation* (this step also makes up the maintenance process which may automatically modify correspondent object information in order to keep it in consistency with modified metainformation).

A digraph of metainformation describes the abstract syntax of a structural language in terms of which digraphs of information are further formed. The language for metainformation digraph representation is declarative, simple, and at the same time powerful enough to describe arbitrary models, which are adapted to the domain terminology and to the form adopted by the developers of KBS components as well as tools for their creation. A detailed description of the language is given in [19].

Users of the *Editor for digraphs of metainformation* are metainformation carriers who are usually knowledge engineers and systems analysts from various fields of professional activities. The editing process model which is set into the basis of the *Editor for digraphs of metainformation* has much in common with the one which is the basis of the *Editor for digraphs of information*. The differences are caused only by formalism of representation of the correspondent digraphs and by the fact that metainformation digraph can have an arbitrary form (limitations are set only by expressive means of the *language for metainformation digraph representation*), whereas information digraph has a structure limited by its metainformation digraph.

V. Problem solver development

A *problem solver* is a component of some intelligent service which processes information resources and which encapsulates business logic for problem solving. It is a set of agents, each of them solves some subproblem(s) and interacts with others by message exchange. A lifecycle of a message starts from its creation by some agent, followed by it being sent to another agent which receives and processes it. Then a message ceases to exist.

The proposed approach for development of agent communication means is a multilanguage one. Messages must be represented in some language(s) which syntax and semantics must be understood by interacting agents. Each language can have an arbitrary complex syntax structure (represented by metainformation digraph), and contents of messages (represented by digraphs of information) are formed in accordance with it.

There are two agents within the solver (in general case) which have particular roles:

- *root agent* – an agent to which an *initializing message* is sent by utility agent *System agent* when a service starts to run after its launch (meaning that this agent runs first among all agents of problem solver);
- *interface controller agent* – an agent whose interaction with the utility agent *View agent* (see section VIII) provides coupling of UI with problem solver (in case when a service with UI is developed).

The mentioned *initializing message* is represented in the communication language which belongs to the class of utility ones. Another such distinguished languages are used for interacting with utility agents and for stopping the work of the problem solver.

During the process of service execution agents which are parts of the problem solver connect with each other dynamically. In case of a service without UI this interaction starts from the *root agent*. In the case of a service with a UI it usually starts from the *interface controller agent* – for processing the events which are generated in the UI (the *root agent* may do no work at all in that case).

In order for a problem solver to become usable for various services at the stage of their assembly the information resource which represents the declarative specification of this problem solver must be created. It is formed in two stages:

1) creation of a new information resource of “problem solver” type with setting of its name and *Problem solver structure* information resource as metainformation;
2) formation of content of this new information resource (problem solver assembly) with the use of *Editor for digraphs of information* where the process of editing is controlled by the set metainformation and consists of setting of the following:
   - information about the purpose of the problem solver,
   - a link to its *root agent*,
   - links to *formal parameters* (in case when services should process information resources, i.e. must have actual parameters at runtime),
   - a *UI* which includes a link to *interface controller agent* (in case when a service with the UI is being developed), links to own information resources (shared among all running instances of the problem solver).

The described organization of a problem solver (which implies dedication of a special *interface controller agent* among others) gives an opportunity to separate development and maintenance of problem solver business-logic from same work on UI. This leads to the possibility of involving independent appropriate specialists to these types of work.

VI. Agent development

An *agent* is a (possibly reusable) software component which interacts with other agents with the use of message reception/sending. It is capable of processing (reading and modification) information resources. Reusability means that an agent can become a part of various problem solvers with no modification. Data processing is organized in form of productions which are grouped into one or several production blocks – by the amount of message templates that an agent can process.

An agent consists of two parts: a declarative one and a procedural one. Declarative part of an agent consists of two sections: agent’s documentation (which is presented by a set of descriptions: for an agent itself and for each of its production block, written in natural language) and formal specification for its set of production blocks (which comprise an agent). The declarative part is used as a basis for the support of automation of agent’s documented source code development and maintenance. A digraph 2-level model of information resources representation allows storing of declarative description and code of an agent (procedural part) in a single information resource which represents this agent.

Development of an agent consists of the following stages:

1) creation of new information resource of “agent” type with setting of its name and *Agent structure* information resource as metainformation;
2) formation of content of this new information resource with the use of *Editor for digraphs of information* (with extensions) where the process of editing is controlled by the set metainformation and consists of setting of the following initial data: agent name, agent class name, description, local data structure and production blocks specification (description, templates of incoming messages and corresponding templates of outgoing messages)[1];
3) acquiring agent source code (by generating its sketch using its declarative description or using pre-saved source code) and executable code of used message templates;
4) writing (modifying) the source code of an agent (code for its production blocks in particular) and forming its executable code (as a result of compilation of its source code);
5) uploading source code and executable code into the correspondent information resource (thus extending agent’s declarative specification);
6) agent testing (optional).

[1]Different production blocks of the same agent must have different incoming message templates.

Acquiring agent source code and executable code of reused message templates must be done with extended functionality of the *Editor for digraphs of information*. For agent source code it provides downloading of either sketch of source code which is generated on the basis of new agent declarative description or downloading of the stored source code of modified agent.

Writing the source code of an agent must be done using some modern programming language powerful enough for solving intelligent problems. A suitable IDE can be used or such functionality can be implemented within the toolkit (within some extension of the *Editor for digraphs of information*). The source code of agent's production block implements the whole of part of the ontology-based algorithm for knowledge-based processing using toolkit API methods for: incoming message data reading, knowledge base traversing, outgoing message creation. Note that an agent can make an arbitrary number of messages to be sent to a set of other agents (including system ones and itself).

After the source code of an agent is ready it is necessary to form its ready-to-run version (e.g. bytecode) and load it into the information resource which represents an agent. In order to achieve this a compilation of the source code must be committed either locally or online (if the toolkit has such support). While code uploading and/or compilation it is checked for correctness and safety.

Agent testing is carried out by a separate tool which performs multiple start and execution of the set of its production blocks on a provided set of tests (formed as information resources with metainformation *Agent tests structure* by use of the *Editor for digraphs of information*). Reports are saved with results of test executions. A single test generally includes the information resource representing incoming message, a set of information resources representing expected outgoing messages, and tuples of information resources which act as input, output (initial and final states) actual parameters and own ones (initial and ending states). A test is considered to be passed if the amount and the contents of the outgoing messages and of processed information resources are as expected. Formed sets of tests can be used for regressive testing during the stage of agent maintenance.

VII. Message template development

A *message* in multi-agent systems is a mechanism for coordinated interaction of agents which provides exchange of information between them and transfer of requests [20]. Agents communicate in different languages whose amount is extensible (by means of creating new message templates and adding new production blocks into agents or extending existing ones) and is limited only by the total amount of message templates. Specific languages of agent interaction (message templates) are set at the level of separate production blocks.

As messages are object information resources so languages for sets of messages are represented by information resources that are metainformation – *Message templates*. They not only contain the structure for a set of messages but may also hold a set of methods for processing these messages. Thus, like an agent, a message template consists of two parts – a declarative one and a procedural one. Its structure is simpler though: name, class name, description, message structure, source code and code.

Development of the message template consists generally of the same stages as of an agent. The differences appear at the stages of creation (when the *Message template structure* information resource is set as metainformation) and of writing the source code (due to differences in the structure of declarative specifications of agents and message templates).

Message template testing can be done only through testing of an agent (see section VI).

VIII. User interface development

Development of an interface for an intelligent cloud service is a development of web-interface (as services are available online and are accessed via web-browsers). The interface consists of interface controller agent (development described in section VI) and a set of web-pages (with one selected as the starting one). Each *web-page* has a name and data which can be of *content* or *design* type:

- *content* – a mixture of text, *ui tags* and a set of names of *design web-pages* (of the same problem solver);
- *design* – a description of CSS classes (a set of CSS rules).

Such separation allows:

- to apply various CSS to the same contend of the interface and vice versa – to use the same CSS for interface of various services, which leads to increase of flexibility and adaptability of developed interfaces and to simplification of their maintenance;
- to divide the processes of development and maintenance (thus, to make them independent) of these parts and to involve independent appropriate specialists to these types of work (their only interaction would lie in setting/using same names for classes in CSS).

The *ui tags* of a *web-page* are processed as *ui requests* (one request per tag) sent through *View agent* to agents which act as *interface controllers* within solvers at the initial *web-page* display. Further interactions of user with web-page elements produce other *ui requests* (determined by those elements). Each *ui request* is a set of pairs: *parameter* = *value*. After the request is passed to solver agent, the *View agent* waits for result and in case it is a fragment of UI – puts it into the shown web-page content in place of the processed *ui tag*.

A detailed scheme of interaction of the *View agent* with Web-server and *interface controller agent* (which is a part of some problem solver) is shown in Fig. 2. A process of interaction consists of request transferring from browser to agents and returning of result to browser. A processing of a request is performed in between. Numeration of arrows in Fig. 2 sets the order of interaction.

The basis of the UI presentation model is an “MVC” (Model-View-Controller) conception [21]. Its projection on this conception on the proposed interface model is as follows.

1. Model. This component includes:

- *Model of abstract user interface* – an information resource which represents metainformation and holds a description of structure for standard interface elements of WIMP-interfaces and a way of their organization into a single nested structure.
- *Abstract interface generation API* – a set of high-level functions for creation of fragments of abstract interfaces. Performing calls of these functions (with necessary arguments) significantly increases the level of abstraction at which the information resource that represents some abstract interface is formed. To form a description for some fragment of an abstract interface one has to construct a superposition of function calls of this API.

2. View. This component is presented by the utility agent *View agent*, which is a "hybrid" one. It is divided into two parts so that it can mediate between the two:

- Web-server – it interacts with external part of the *View agent*,
- *interface controller agents* of problem solvers – they interact with agent part by receiving, processing and replying to messages created by *Request from View agent* utility message template.

The other tasks of the *View agent* are the following:

- the production of specific interface (HTML-code) on the basis of the its abstract model (with the use of built-in mapping rules for all supported interface elements);
- uploading/downloading binary data to/from Web-server.

3. Controller. This component is represented by agents which play a role of *interface controller agent* (within problem solvers). These agents interact with the *View agent* by message exchange. Such messages are created by particular message templates. Agents implement (possibly by interaction with other agents) logic for processing *ui requests* of the following origins: *ui tags* (from web-pages content data) and ui events (generated by interface elements in response to user actions).

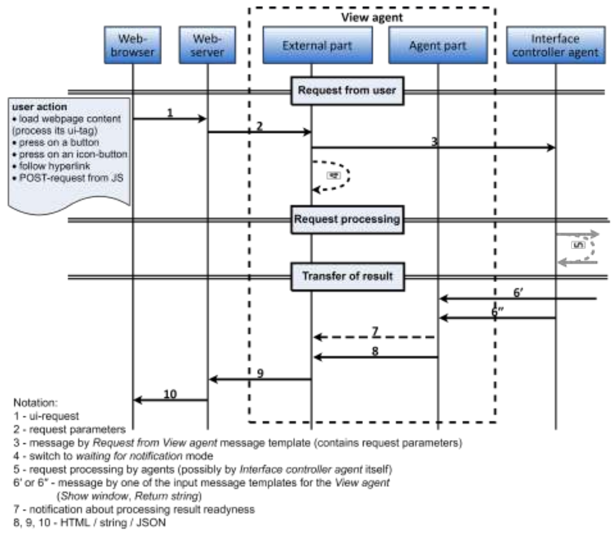

View agent
Web-browser
Web-server
External part
Agent part
Interface controller agent
Request from user
user action
• load webpage content (process its ui-tag)
• press on a button
• press on an icon-button
• follow hyperlink
• POST-request from JS
1
2
3
4
Request processing
5
Transfer of result
6'
6"
7
8
9
10
Notation:
1 - ui-request
2 - request parameters
3 - message by Request from View agent message template (contains request parameters)
4 - switch to waiting for notification mode
5 - request processing by agents (possibly by Interface controller agent itself)
6' or 6" - message by one of the input message templates for the View agent (Show window, Return string)
7 - notification about processing result readyness
8, 9, 10 - HTML / string / JSON

Figure 2. A scheme of interaction of the *View agent* with Web-server and *interface controller agent*.

A result of processing a *ui request* is either an information resource (which represents a description of an abstract interface which is passed to the agent part of the *View agent*) or an arbitrary string of characters.

The development of UI consists of the following steps:

1) creation of a set of *web-pages* within the declarative specification of the used problem solver – with formation of content for each page;
2) development of *interface controller agent*, which must implement the logic for processing *ui requests*.

IX. Discussion and future work

Let's highlight the main features that distinguish the described technology for the development of cloud services and their components from other available solutions.

The possibility to include domain experts in the process of developing knowledge bases and databases. The proposed technology supports a two-level ontological approach to the formation of information resources. Its feature is a clear separation between the ontology and the knowledge base (database) formed on its basis. The ontology (metainformation) sets not only the structure and terminology for a knowledge base, but also the rules for its formation, control of the integrity and completeness. The ontology is formed by a knowledge engineer (possibly in cooperation with domain expert) with the use of the ontology editor. On its basis, a UI is automatically generated that allows domain experts to create (form) knowledge bases and databases without involving professional intermediaries. All information resources have a semantic representation that is understandable to domain experts.

Providing reuse of ontologies and problem solvers. A clear separation between ontology and knowledge bases (databases) also provides another significant advantage. When an ontology is formed – the solver is designed in its terms. Then using this ontology, an arbitrary number of knowledge bases can be developed and their binding with the solver (which is integrated with UI and plays the role of KBS shell) makes the new KBS. So this moves us from developing of specific KBSs each time from a scratch to developing of shells first which can then be used with different knowledge bases to comprise different KBSs. Thus, the ontology and the solver are reused for a whole class of problems.

Modification of knowledge bases without changing the problem solver's code. This is also provided by the separation between the knowledge base and its ontology, the lack of domain knowledge in the solver. The ontological solver is implemented not as an inference in calculus, but as an algorithm that traverses the knowledge base in accordance with its ontology to match statements in the knowledge base with the input data and, thus, consistently confirming or refuting elements of knowledge.

Transparency and maintainability of problem solvers. The development of KBS solvers is based on the processing of hierarchical graphs of concepts, which makes it possible to create a set of common APIs (application programming interfaces) for working with such graphs. Based on the specifics of the problem, the expected repeatability of their smaller subproblem, the developer can choose software components that will provide a more transparent architecture of solvers. The use of an agent-based approach provides the possibilities to parallelize the execution of subtasks, which is important for problems with high solvation speed requirements.

It is important that the model (metainformation) of each software component is hard-set and allows one to create them using the common editor or its appropriate specialized adaptations (Fig. 3). This increases components' transparency and maintainability, as it is known from software engineering [22] that declarative components are easier to maintain than procedural ones. A unified declarative representation of agents, as well as message templates made it possible to automate the creation (process of generation) of code templates for them.

The proposed technology is implemented on the IACPaaS platform [12] – a cloud computing system for support of development, control and remote use of multi-agent services. Thus, it provides not only users with remote access to applied intelligent systems but also developers with remote access to appropriate tools for creation of such systems and their components which make this process more automatic (Figs. 4, 5, 6).

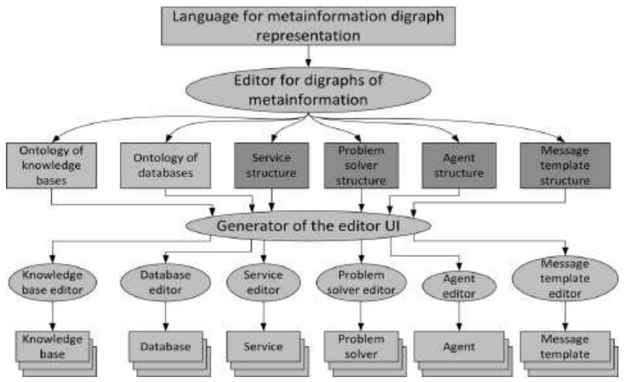
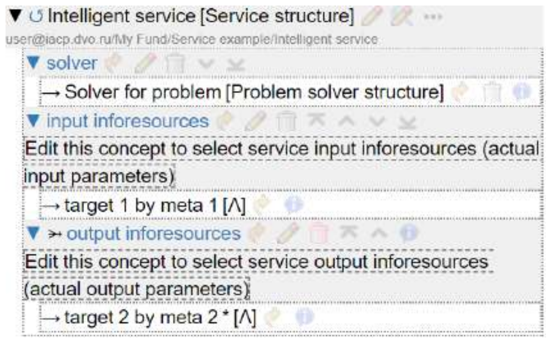


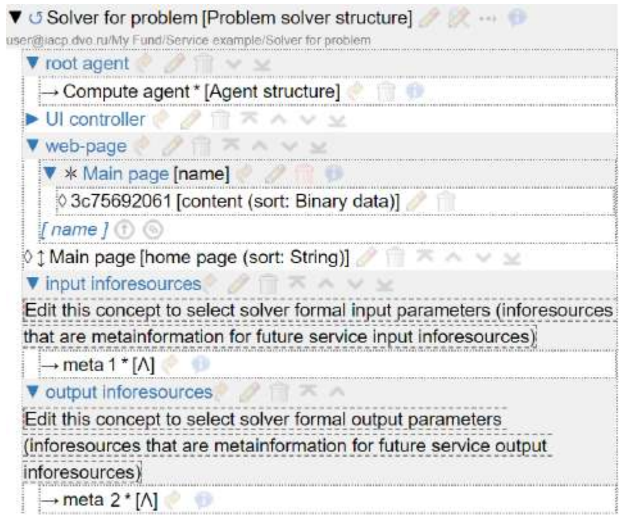
Figure 3. Usage of the development toolkit for creating information and software components of intelligent services.



▼ ↺ Intelligent service [Service structure]
user@iacp.dvo.ru/My Fund/Service example/Intelligent service
▼ solver
→ Solver for problem [Problem solver structure]
▼ input inforesources
Edit this concept to select service input inforesources (actual input parameters)
→ target 1 by meta 1 [Λ]
▼ ➣ output inforesources
Edit this concept to select service output inforesources (actual output parameters)
→ target 2 by meta 2 * [Λ]

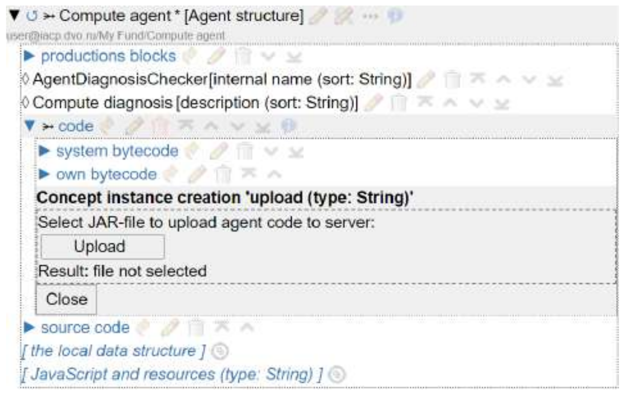
Figure 4. UI fragment of the *Service editor*.

Currently, the IACPaaS platform has several hundreds of users who, using the proposed technology, have created ontologies, knowledge bases, tools and problem solvers for various domains: a set of tools for developing professional virtual



▼ ↺ Solver for problem [Problem solver structure]
user@iacp.dvo.ru/My Fund/Service example/Solver for problem
▼ root agent
→ Compute agent * [Agent structure]
▶ UI controller
▼ web-page
▼ ⁎ Main page [name]
◊ 3c75692061 [content (sort: Binary data)]
[name]
◊ ↕ Main page [home page (sort: String)]
▼ input inforesources
Edit this concept to select solver formal input parameters (inforesources that are metainformation for future service input inforesources)
→ meta 1 * [Λ]
▼ output inforesources
Edit this concept to select solver formal output parameters (inforesources that are metainformation for future service output inforesources)
→ meta 2 * [Λ]

Figure 5. UI fragment of the *Problem solver editor*.



▼ ↺ ➣ Compute agent * [Agent structure]
user@iacp.dvo.ru/My Fund/Compute agent
▶ productions blocks
◊ AgentDiagnosisChecker[internal name (sort: String)]
◊ Compute diagnosis [description (sort: String)]
▼ ➣ code
▶ system bytecode
▶ own bytecode
Concept instance creation 'upload (type: String)'
Select JAR-file to upload agent code to server:
Upload
Result: file not selected
Close
▶ source code
[the local data structure]
[JavaScript and resources (type: String)]

Figure 6. UI fragment of the *Agent editor*.

cloud environments [23], services for interactive verification of intuitive mathematical proofs, underwater robotics, practical psychology, agriculture and others. In particular, the formed portal of medical knowledge includes a wide range of medical ontologies, complex knowledge bases formed on their basis (for example, the knowledge base for medical diagnosis of diseases of various nosologies contains more than 130 000 concepts) and problem solvers (computer-based training simulator using classical research methods in ophthalmology, diagnosis of diseases [24], differential diagnosis of acute and chronic diseases: infectious diseases, gastro-intestinal diseases, hereditary diseases, diseases of the cardiovascular and respiratory systems, etc.), the prescription of personalized medical and rehabilitation treatment, etc.).

By now, users of the platform have created more than 2 thousands of resources, rough numbers are: almost 1750 information resources (ontologies – 650, knowledge and data bases – 1100) and 500 software components (problem solvers – 180, agents – 250, message templates – 70). So we can say that the active use of the platform demonstrates that the technology proposed by the authors meets modern requirements for the development of viable KBS and fits the needs of users.

However, we continue to work on improving the platform and technology for KBS development. Our main efforts are aimed at creating tools for generating adaptive interfaces, tools for intelligent user support and design automation. Special attention is paid to the creation of specialized technologies for developing classes of KBS and increasing the level of instrumental support for various technologies. Important tasks are the improvement of tools for safety and security of the platform, of the common APIs for processing the storage units and creation of high level abstraction operations for information resources.

References

[1] Gensym G2. The World's Leading Software Platform for Real-Time Expert System Application. Available at: http://www.gensym.com/wp-content/uploads/Gensym-l-G2.pdf (accessed 2021, May).

[2] V. V. Golenkov, N. A. Gulyakina, I. T. Davydenko, D. V. Shunkevich, Semantic technologies of intelligent systems design and semantic associative computers, Doklady BGUIR, 2019, 3, pp. 42–50.

[3] M. A. Musen, The Protégé Project: A Look Back and a Look Forward. AI Matters, 2015, 4, pp. 4–12.

[4] G. V. Rybina, 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. (in Russian).

[5] G. Rodríguez, Á. Soria, M. Campo, Artificial intelligence in service-oriented software design. Engineering Applications of Artificial Intelligence, 2016, 53, pp. 86–104.

[6] I. Gupta, G. Nagpal, Artificial Intelligence and Expert Systems. Mercury Learning & Information, 2020, 412 p.

[7] G. Tecuci, D. Marcu, M. Boicu, D. Schum, Knowledge Engineering: building Cognitive Assistants for Evidence-based Reasoning, Cambridge, U.K. Cambridge University Press, 2016.

[8] C. E. Grant, D. Z. Wang, A Challenge for Long-Term Knowledge Base Maintenance, Journal of Data and Information Quality, 2015, 6(2-3):7.

[9] R. S. Pressman, B. G. Maxim, Software Engineering: Practitioner's Approach, New York, 9th ed. McGraw-Hill, 2019, 704 p.

[10] A. Torreño, Ó. Sapena, E. Onaindia, FMAP: A platform for the development of distributed multi-agent planning systems. Knowledge-Based Systems, 2018, 145, pp. 166–168.

[11] M. Gholamian, G. Fatemi, M. Ghazanfari, A Hybrid System for Multiobjective Problems – A case study in NP-hard problems, Knowledge-Based Systems, 2007, 20(4), pp. 426–436.

[12] V. V. Gribova, A. S. Kleschev, Ph. M. Moskalenko, V. A. Timchenko, L. A. Fedorischev, E. A. Shalfeeva, A cloud computing platform for lifecycle support of intelligent multi-agent internet-services. In proc. of International Conference on Power Electronics and Energy Engineering (PEEE-2015) (19-20 Apr. 2015), Hong Kong, pp. 231–235.

[13] AI portal. Agent classification. Available at: http://www.aiportal.ru/articles/multiagent-systems/agent-classification.html (accessed 2021, May).

[14] V. F. Khoroshevsky, Proyektirovaniye sistem programmnogo obespecheniya pod upravleniyem ontologiy: modeli, metody, realizatsii [Ontology Driven Software Engineering: Models, Methods, Implementations], Ontologiya proyektirovaniya [Ontology of designing], 2019, 9(4), pp. 429–448. (in Russian).

[15] V. V. Gribova, A. S. Kleshchev, F. M. Moskalenko, V. A. Timchenko, Implementation of a Model of a Metainformation Controlled Editor of Information Units with a Complex Structure. Automatic Documentation and Mathematical Linguistics, 2016, 1, pp. 14–25.

[16] G. Taentzer, K. Ehrig, E. Guerra, J. Lara, L. Lengyel, T. Levendovszky, U. Prange, D. Varro, S. Varr´o-Gyapay, Model transformation by graph transformation: A comparative study, in: Proceedings Workshop Model Transformation in Practice, Montego Bay, Jamaica, 2005, pp. 1-–48.

[17] C. Atkinson, T. Kuhne, Model-driven development: A metamodeling foundation, IEEE Software, 2003, 5, pp. 36–41.

[18] A. Kleppe, S. Warmer, W. Bast, MDA Explained. The Model Driven Architecture: Practice and Promise. Addison-Wesley Professional, Boston, 2003.

[19] V. V. Gribova, A. S. Kleshchev, F. M. Moskalenko, V. A. Timchenko, A two-level model of information units with complex structure that correspond to the questioning metaphor. Automatic Documentation and Mathematical Linguistics, 2015, 5, pp. 172–181.

[20] V. S. Dyundyukov, V. B. Tarasov, Goal-resource networks and their application to agents communication and co-ordination in virtual enterprises, IFAC Proceedings Volumes, 2013, 46(9), pp. 347–352.

[21] R. Trygve, MVC. Xerox PARC 1978-79. Available at: https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html (accessed 2021, May)

[22] I. Sommerville, Software Engineering, 10th edition, Pearson, 2015.

[23] V. V. Gribova, L. A. Fedorischev, Software toolset for development of cloud virtual environments. Software products and systems, 2015, 2, pp. 60–64.

[24] V. Gribova, Ph. Moskalenko, M. Petryaeva, D. Okun, Cloud environment for development and use of software systems for clinical medicine and education. Advances in Intelligent Systems Research, 2019, 166, pp. 225–229.

Технология разработки жизнеспособных интеллектуальных сервисов

В.В. Грибова, Ф.М. Москаленко,
В.А. Тимченко, Е.А. Шалфеева

Предложена технология разработки интеллектуальных мультиагентных облачных сервисов. Ее ключевым аспектом является независимая разработка баз знаний, пользовательского интерфейса и решателя задач в виде ансамбля агентов, а также их интеграция в интеллектуальный сервис. Технология реализована на облачной платформе IACPaaS, обеспечивающей экспертно-ориентированное создание каждого компонента сервиса с помощью соответствующего инструментария.

Received 28.05.2021