



A Scalable Framework for Developing Viable Ontology-Centric Systems

*Valeria V. Gribova, Philip M. Moskalenko, Elena A. Shalfeeva,
Vadim A. Timchenko*

Institute of Automation and Control Processes of the Far Eastern Branch of the Russian Academy of Sciences (IACP FEB RAS), Vladivostok, Russia

E-mails: gribova@iacp.dvo.ru philipmm@iacp.dvo.ru shalf@iacp.dvo.ru vadim@iacp.dvo.ru

Abstract: *The paper addresses the question of creating long-living software systems. One of the possible challenging comprehensive solutions for this might be to provide a viable framework which supports their complete life-cycle. The concept and architecture of a scalable framework for creating and maintaining software ontology-centric systems are proposed. The basis for the framework is the 2-level approach to the formation of system components of all types. It allows one to separate component models (ontologies) from components themselves. The 3-layer architecture of the framework is proposed. An implementation of the framework concept as a cloud platform was created. Several knowledge portals have been developed with its use. This evaluation showed that the proposed solution is a promising way for the organization of an ecosystem for collaborative development of viable information and software resources.*

Keywords: *Software engineering, Software development framework, Ontology, Ontology-centric system, Component model, IACPaaS cloud platform.*

1. Introduction

The issues of knowledge and data management, their intellectual processing for solving a wide range of professional problems, are very topical. Among the various ways to their solution, the creation of specialized Software Systems (SS) within the ontological approach occupies one of the leading places [1-3]. Ontology is the basis for knowledge and data formalization for a wide class of information and intelligent systems, in particular, for expert systems, Decision Support Systems (DSS), various information support systems for scientific and industrial activities, including

knowledge portals [4, 5]. At the same time, it is necessary to provide not only access to domain data and knowledge, but also their processing (automatic or automated): knowledge-based inference, data mining, transformation and generation of new data and knowledge.

Solving a set of problems is impossible without creating “ecosystems” – hardware and software platforms that support the full life cycle of intelligent systems development, control and maintenance. Such ecosystems should have tools for developing ontologies, DataBases (DB) and Knowledge Bases (KB), tools for creating software solvers that provide their intellectual or computational processing. The feature of such platforms is the integration in a single shared information (possibly distributed) space of an ensemble of related information and software resources for the developers’ cooperation. They organize workplaces for individual users and teams that create, test, modify, and update information (databases, knowledge bases, ontologies) and software components [6-8].

Such ecosystems are characterized by the availability of methodologies and technologies for creating various classes of SS – from sets of templates to tools and software complexes for development automation. They allow for reusing components, automatically generating components, and scaling solutions. At the same time, it is important to ensure the viability of SS, which is one of the key problems of software engineering. The viability of SS is considered as the maintenance of their operability and usefulness in conditions of variability of user requirements, domain, environment (new technologies, methodologies, platforms, methods and protocols of interaction, etc.) [9, 10]. The achievement of this SS criterion is ensured through self-adaptation mechanisms laid down in the SS and/or through maintenance (with human participation) [11-14].

As for maintenance cost, in some publications, for example, in [15], it is noted that it reaches 80%. Among the factors that affect the cost of software maintenance and its viability are the tools that are used for SS development and maintenance. They are noted in [16-19]. Thus, considering the viability of SS, it is impossible to ignore the tools for their development. Such tools, which are also SS, should be developed and improved, providing new and advanced features to users.

We aim to describe the conception and architecture of a scalable framework for creating and maintaining SS based on ontologies, as well as their components. It supports not only some *basic methodology for SS development* but also provides means for framework enhancement.

2. Related works

Ontologies, ontological tools and frameworks, ontological portals, platforms and metasystems are created for intellectual processing of knowledge and data, control of their quality, and improvement of the decision-making process [20-23]. Domain ontology promotes the exchange of knowledge and data, supports their reuse, and ensures rapid growth of the domain digitalization [24]. Technologies and tools for ontology-driven editing are widely represented: Protégé, Apollo, OntoStudio, Swoop, Fluent Editor, NeOn Toolkit, OsaWL, Vitro, Semantic Turkey, IMS.ostis, etc. [20, 25]. The tools have a web-based or client-server architecture; some are deployed as platforms for developers, where semantic editors are integrated with tools for semantic search, identifying and analyzing relationships, tools for reasoning and development of semantic applications and services.

Semantic editors make it possible to describe ontologies of a domain or some problem for their use during the formalization of knowledge and other conceptual information. Most of them use frame-based languages and propose implementing an ontology in terms of classes and subclasses of entities, indicating binary relationships, restrictions, and several axioms. Other tools support the generation of specific instances of concepts and relationships based on the ontology. This does not yet allow one to reflect a real professional knowledge structure, but only a close-to-reality ontology can ensure the explainability of intellectual processing results for professional tasks [26]. One of the platforms for building decision support systems and knowledge graph management systems is ArchiGraph KMS [27]. It supports the construction of rules (such as “if ... then ...”) applied to the contents of the knowledge base, which allows one to create search queries and solve search problems. To describe sequences of logical reasoning, advanced platforms, such as OsaWL [28], provide a graphical editor of rules (or chains of triplets) and an editor of semantic queries in a visual form.

Inference facilities on ontology platforms are often limited to direct inference via “subClassOf” axioms as well as constraint and consistency checking mechanisms. From the relationships between classes, subclasses of a superclass are automatically calculated through reasoning. Some tools provide generation of hypotheses for decision making or complex queries using an “If-Then-Else” reasoning engine. OsaWL-editor allows creating rules to calculate relationships between objects by combining triplets into chains and imposing restrictions on the equality or non-equality of subjects and objects of triplets. Protégé’s SWRL plugins are often used, or a mechanism for inserting SWRL rules into an ontology for reasoning within forward and reverse inference chain strategies (SWARMS

platform, ROBOT tool & library). Many tools are limited to modeling RDF(S) and OWL and developing semantic applications and services based on RDF (TopBraid Composer, TopBraid Suite). Others rely on a plug-in architecture to build knowledge base applications (OntoStudio, Protégé). The cross-platform version of Protégé for Java allowed the creation of a flexible architecture for various plug-ins.

Knowledge graphs or semantic networks are used to formalize knowledge that is close to the “real world”, in particular, reflecting the dynamic nature of knowledge: some have a basic set-theoretic interpretation, others propose to record any named relations, and not only binary ones; less versatile ones offer graphic-dynamic models (OSTIS) [22]. Tools are offered to obtain results by dynamic Knowledge Graphs (KG reasoning) [29]. Often, agent-based knowledge processing is used [30].

As for the programming languages on such platforms, Java is usually used. Tools integration is becoming popular, e.g., the SpringBoot framework in conjunction with MyBatis and using MySQL Workbench OSS. For integration with other successful solutions, it was proposed to export ontologies to relational databases [31].

Without continuous toolkit evolution, it is impossible to create long-living software systems. Constantly changing requirements, the influence of other successful projects and ecosystems, and the emergence of new platforms, technologies, and interfaces are the factors that require the development of tools. Moreover, toolkit developers are forced to “be one step ahead” so that when requirements for a software product change, the toolkit already has the necessary capabilities. So, for example, Protégé, on the one hand, has evolved from the Xerox Lisp platform to the NeXTSTEP, Windows, and Java version; on the other hand, from the desktop to the client-server and web-based ontology editor. The modern trend is to support the collaborative development of ontologies (OntoStudio) and create complex solutions for knowledge exchange and scientific research (FluentEditor).

Other factors that require the extensibility of the toolkit and affect the viability of the SS are new scientific achievements, user feedback and requests, and User Interface (UI) evolution. Such extensions were mentioned for Protégé in [25]. The evolution of the AT-TECHNOLOGY [11] consisted of the creation of additional interface and configuration components, subsystems for integration with application programs that simplify the development of SS created on its basis. The Gensym G2 system [32] is constantly supplemented with new knowledge debugging methods, knowledge representation formalisms, and new problem-oriented extensions [33].

Despite the expansion of the range of platforms, they still do not provide an opportunity to build specialized shells for domains with EXplainable Artificial Intelligence (XAI) based problem solvers. Platforms do not offer tools for creating shells for assembly of DSS for the class of intelligent problems. Platforms do not offer SS developers the opportunity to adapt the proposed tools and expand the capabilities of technologies by themselves – to adjust to the formats and methods of information processing.

There are two approaches to the development of tools. The traditional approach is based on the improvement of the toolkit by its developers. In this case, the toolkit is considered as a “black box” into which the developers themselves add new service functions.

The second approach is based on the extension of the toolkit by its users. In this case, development is possible through the creation of modules (plugins) that implement new toolkit features and libraries of reusable components.

Thus, an urgent challenge is to provide a comprehensive solution to the problem of toolkit extensibility. This can be achieved through the addition of reusable component libraries, new functions, technologies, methodologies, etc., done by toolkit creators and users. And the development tools for shells of XAI-based SS (expert, teaching, etc.) are the most in demand.

3. Proposed approach

Let’s highlight the basic principles for creating a viable, scalable framework for building and maintaining ontology-centric software systems.

- **The framework architecture should support openness and scalability, enabling third-party users to develop and integrate additional functional components.**

- **A unified environment and a set of tools should be provided both for scaling the framework and for developing an SS based on it.** This requirement holds significant importance for several reasons. Firstly, the development of the SS can yield elements that are directly applicable as extensions to the framework. For example, some of those elements can be placed into libraries of reusable components. Secondly, users have the capacity to develop new “add-ons” tailored to address specific challenges, which may possess broader utility beyond their immediate needs, benefiting other potential users of the framework. Such add-ons could encompass novel technologies with corresponding tool support, new interfaces, or enhanced sets of development tools, among other possibilities. Obviously, if the development of the framework and of the SS is based on different

principles and methodologies, then progress in the development of tools will be difficult. It will require the study of new programming languages, technologies, methodologies, and relevant experience.

- **Reduction of the development complexity, and provision of the possibility for making changes throughout the entire life cycle.** This requirement is ensured by the reuse of components, transparency of their architecture (clarity for making changes), and automation of the component creation process. To implement this key requirement, five solutions are proposed.

- 1) **Explicit separation of component ontologies from the components themselves and a two-level approach to their formation.** The ontology-driven approach is widely recognized as the most prevalent, or, as noted in [34], is the de facto standard for developing Knowledge Bases (KB) and other information resources. Still, existing methodologies for creating such resources, despite incorporating some form of ontological superstructure, fail to provide a clear distinction between the resource itself and its ontology. Our approach advocates for such strict separation for three key reasons. First, the processes of ontology development and the creation of information resources based on them constitute distinct types of activities. Ontologies are typically formed by cognitive scientists and analysts (in collaboration with domain experts, using a specialized ontology specification language, mentioned later here). Whereas information resources (such as KBs and DBs) are constructed by domain experts in alignment with the ontology (without requiring knowledge of the ontology specification language). Second, an ontology should generalize a class of problems rather than be overly specific to a particular problem. This allows for the creation of a wide range of diverse KBs or DBs based on this single ontology, contingent on the level of abstraction of the ontology. Third, by explicitly separating ontologies from KBs and DBs, problem-solving mechanisms can be developed based on the former independently of specific representatives of the second group. This enables modifications to the KBs or DBs without necessitating changes to the problem solver's code, thereby enhancing maintainability.

- 2) **Unified declarative semantic representation of all information structures (ontologies, knowledge bases, databases) and software components.** This approach ensures universality in both their software processing and their interactive formation of these elements through component-oriented editing tools. By adopting such a consistent approach, the framework provides seamless integration and interoperability across diverse information and software entities, facilitating efficient development, modification, and utilization.

3) A common declarative approach to the assembly of SS, to the design of processing tools (problem solvers), and declarative parts of their software components based on unified ontologies. The principal solution is to identify declarative parts of software components of SS, develop ontologies for them, and provide editors driven by these ontologies. This significantly reduces the “entry threshold” for developers. It also has a significant impact on the transparency of the SS architecture and its documentability.

4) A meta-level language for defining ontologies of all declarative components within the SS. Ontologies are constructed using a specialized language tailored for representing their structural models. This language is universal, independent of specific domains or problem contexts, and is inherently declarative. It features a straightforward structure while encompassing a rich set of expressive tools capable of modeling diverse, problem-centric meta-terminological structure for KBs and DBs with varying degrees of complexity.

5) Automatic generation of specialized knowledge base/database/software component editors based on ontologies (ontology-driven editors). Each editor is built upon a process model that reflects the ontology of the corresponding component, ensuring alignment and consistency (correspondence) between the component and its ontological structure. Furthermore, the editor can verify adherence to the constraints and agreements specified for the ontology model. By incorporating ontology-based UI and interactive dialogue scenarios, these specialized editors empower SS developers to design and manage SS components using domain-specific meta-terminology and concepts that are both clear and comprehensible.

4. Results

Taking into account the requirements and principles for the creation of a framework that were outlined above, a three-layer architecture of a scalable framework for developing ontology-centric systems is proposed. It consists of a *core layer*, a *basic layer*, and an *extensible layer* (which thus makes the whole framework scalable).

4.1. Framework architecture

4.1.1. Core layer

The framework core layer defines the basic principle for developing all components. This principle states that information components, as well as declarative parts of software components, are formed in accordance with a two-level approach. This means that it is possible to build an ontological model that

most accurately reflects the specifics of the domain at first. Then, this ontology (information structure and a set of ontological agreements) guides the development of typical components. This principle applies not only to the development of components for the SS, but also to the framework itself.

The core layer consists of the three components (Fig. 1). To specify the ontologies of components, regardless of their purpose, a single declarative language (metalanguage) is used – the *Ontology specification language*. To facilitate the creation of ontologies using this language, the core includes an *Ontology editor*. The core also contains a *Component editor generator* designed to automatically produce declarative component editors based on corresponding ontologies. The tasks of the Component editor generator are: generation of a UI and a component formation scenario with verification of the agreements specified in the ontology, as well as its formal completeness.

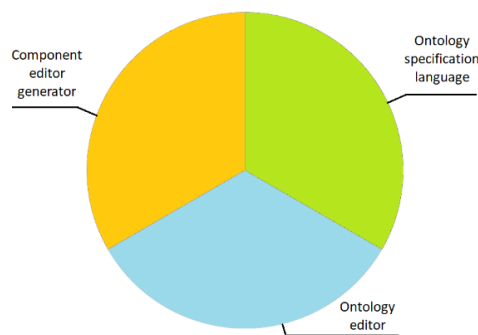


Fig. 1. Structure of the core layer

The ontology specification language is the key component of the entire framework. The following main ideas are laid in its creation:

- the language should be declarative, fairly simple, and at the same time, powerful, allowing to specify arbitrary models of ontologies, oriented and adapted to the terminology and form adopted by developers of ontology-centric system components and tools for their creation;
- the formation of the necessary components by developers should be carried out “from top to bottom”, with transition from general concepts to their details;
- the maintenance of the correspondence between the ontology and the component formed on its basis.

Ontology editor. The main requirement for the ontology editor is to simplify the development of component ontology models by ensuring their interactive formation without forcing developers to study the syntax of the ontology specification language. During such editing, the developer must choose the actions acceptable in each specific development context to form the necessary ontology. At

the same time, the editor should be aimed at a wide range of users, which may include knowledge engineers (cognitologists) and domain experts (when developing systems, they form a knowledge/data representation model or its ontology), programmers, UI designers, analysts, and other developers.

Component editor generator. The development of specialized editors is a separate time-consuming task. A universal language and a unified internal representation of both ontology models and ontology-based components allow one to apply uniform principles for automatic generation of editors for all typical SS components. The editor is based on the model of the process for generating an SS component in accordance with its ontology.

The *core layer* is sufficient for creating and maintaining information components of any complexity (as well as for specifying declarative parts of software components of any type).

4.1.2. Basic layer

For further support of the framework's *basic development methodology*, the basic layer (in addition to a set of tools of the core layer) includes a set of *Component ontologies* and software tools (Fig. 2), sufficient to create an SS and to control it. Since all components are formed according to their structural declarative ontology models, respectively, this layer of the framework includes *Component editors* for creating components. Such editors are generated by the *Component editor generator* for each of the ontologies. In addition to the above elements, this layer contains a *Control system* and *External software components*. The *Control system* provides access to framework functionality, including the creation and management of SS components. The *External software components* are used to create the UI and the imperative part of those components where the declarative representation is not enough to implement the functionality they need. The maintenance of the framework's basic layer elements is carried out by its developers.

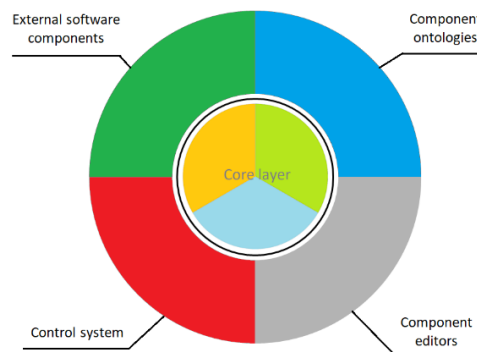


Fig. 2. Structure of the basic layer

The framework basic layer is sufficient to manage the process of creating software components, linking them with information components, assembling an SS, and launching it for execution.

4.1.3. Extensible layer

The structure of the framework extensible layer is shown in Fig. 3. This layer can be enhanced with new development methodologies (technologies) and their tool support by both framework developers and their users (domain experts, knowledge engineers, programmers, analysts). The extension can be carried out in three ways: using the elements of the *core layer*, of the *basic layer*, as well as using the tools of the *extensible layer* itself. Thus, the recursive use of the developed components of the scalable framework is achieved.

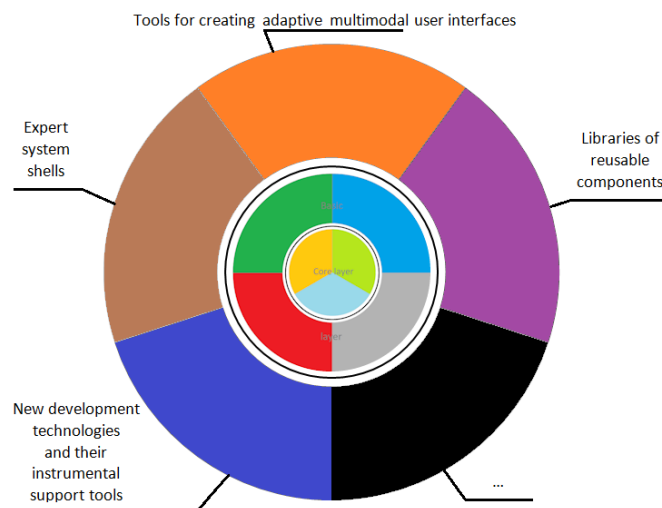


Fig. 3. Structure of the extensible layer

This third layer opens up the possibility of designing new SS development technologies with generic architectures (or solving certain classes of problems) that show themselves to be in demand in practice.

4.2. Framework implementation

4.2.1. IACPaaS platform architecture

The described framework conceptual architecture is set as a basis for the IACPaaS cloud platform [35, 36]. It is intended for the development, management, and remote usage of intelligent cloud services. Its components are presented in Fig. 4. There, the circled numbers on a component indicate the layer of the framework to which this component relates (① – core, ② – basic, ③ – extensible).

A graph representation [37-45] is a successful universal means for representing various types of complex structured information in different domains. Thus, a certain type of graph is used as a formalism for declarative representation of all kinds of information on the platform: A connected labeled root hierarchical binary directed graph (hereinafter-digraph) with possible loops and cycles [46].

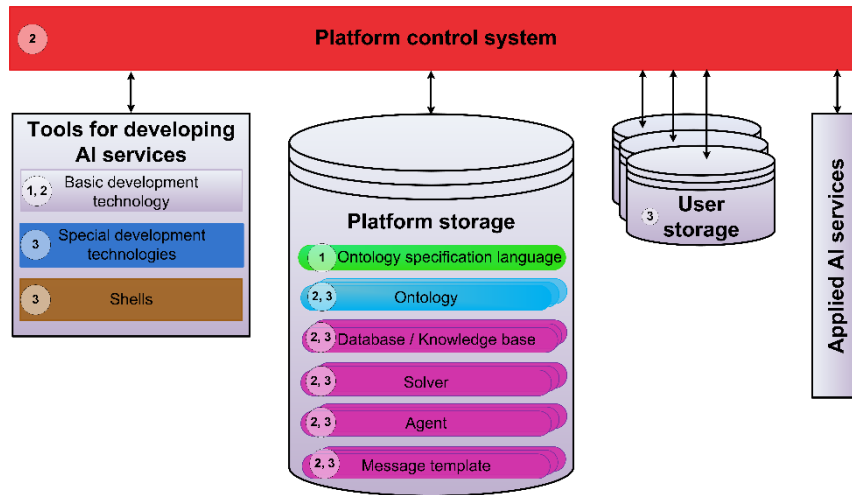


Fig. 4. The conceptual architecture of the IACPaaS platform

4.2.2. Core layer

The *Ontology specification language* provides means of representing ontology models [46]. The *Ontology editor* is an ontology specification language interpreter. It provides consistent “from top to bottom” formation and modification of ontology digraphs [47]. This forms an advantage compared to the approach used for the formation of graph structures underlying Protégé [43]. The formation of information units (components of ontology-centric systems) is carried out using the *Digraph editor*. The ontology model, in fact, controls the processes of editing and of UI generation for the *Digraph editor* (Fig. 5).

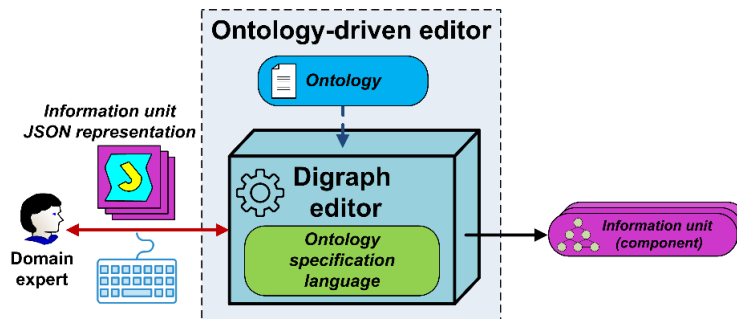


Fig. 5. Scheme of information unit digraphs formation with the use of Digraph editor

When changes are made to the ontology, the UI of the component editor and the editing process are immediately adapted to these changes. The corresponding information units are also automatically brought into a form consistent with the changed ontology. In addition, the check of the component digraph for the fulfillment of the ontological agreements is performed during digraph formation.

The tools of the core layer are sufficient to create ontologies, knowledge bases, and databases (with a graph data model).

4.2.3. Basic layer

The IACPaaS platform implements an agent-based approach to creating solvers (shells) [35]. Thus, the basic layer includes unified ontologies for all software components (Fig. 6).

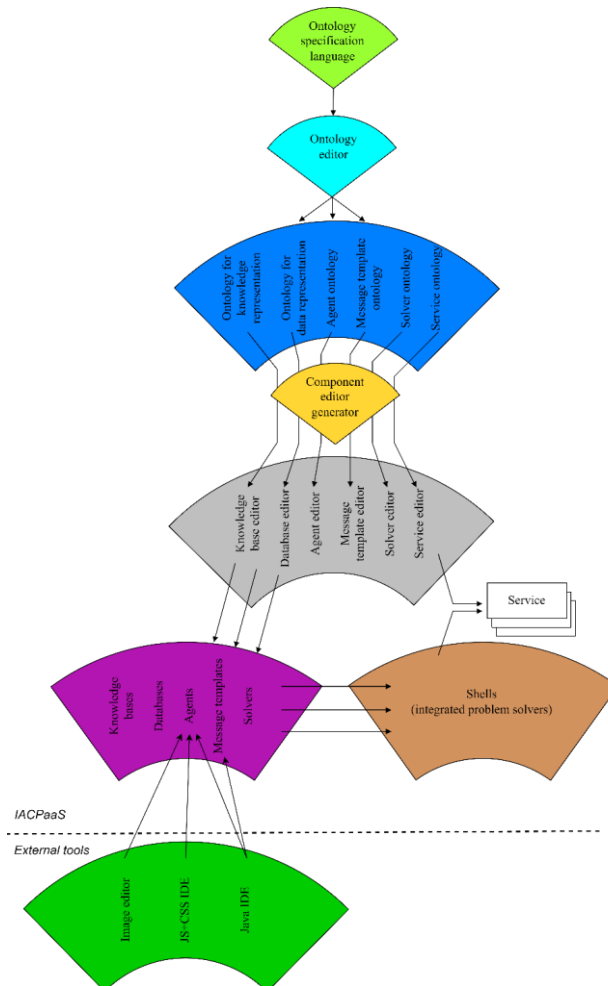


Fig. 6. The scheme of using the framework for creating components of ontology-centric systems

Component ontologies and editors for component formation are created by tools of the core layer. Agents and message templates, in addition to the declarative part, have an imperative component – source code (Java) and executable code (bytecode). The source code sketch can be generated with the use of corresponding editors. They are also used for building the bytecode. However, for the convenience of applied developers, their familiar development tool (*Java IDE*) is considered to be a part of the framework. Platform API (Java methods which form HTML) and JavaScript can be used for UI development. Thus, a *JavaScript and CSS IDE* and an *Image editor* are, in a way, a part of the framework too.

The basic development methodology [35] allows one to create an SS as a cloud application (service) without any special requirements for its architecture. The workflow of this methodology is as follows.

Stage 1. Specification. A knowledge engineer (cognitologist, analyst) uses the *Ontology editor* to create domain *Ontologies for knowledge and data representation*.

Stage 2. Tool generation. The framework automatically generates specialized ontology-driven *Knowledge base editors* and *Database editors*.

Stage 3. Forming the data and knowledge. The domain expert, using the generated *Knowledge base editors* and *Database editors*, creates and maintains *Databases* and *Knowledge bases*, following the structures set by the corresponding *Ontologies*.

Stage 4. Solver's components development (Agents, etc.).

Stage 5. Solver and Shell assembly. The developer builds the *Solver* using the Basic layer tool (*Solver editor*) by binding the components from the previous stage. Then he binds this *Solver* with *Ontologies* from the initial step to get a *Shell*.

Stage 6. System integration and launch. The system (*Service*) is assembled from components (*Shell*, *Knowledge bases* and *Databases*) and is executed under the control of the platform.

4.2.4. Extensible layer

Currently, the extensible layer includes several tools that make software development easier. Three specialized technologies (methodologies) have been introduced to speed up the creation of services with specific architectures. They appeared as a result of generalizing the experience of initial development on the platform.

The first specialized technology is the *Technology for developing services with agents attached to the ontology that controls the process of problem solving* [48]. Using this technology, several platform tools have been redesigned and improved.

The specialized *Technology for creating solvers based on control graphs* [49] was created in order to attract more specialists to the implementation of problem solvers – by providing the opportunity to describe that method declaratively (without writing the code in the programming language).

In addition, the *Technology for cloud service interaction with external software* was developed [50].

With the use of tools of an extensible layer, users have created thematic knowledge portals and workspaces for the accumulation and processing of knowledge bases in order to solve intellectual problems in various domains. They include: medicine, mathematics, autonomous uninhabited underwater vehicles, experimental diagnostics of crops, information protection, educational psychology, programming technology, etc. Let's briefly describe the structure of the portal of medical knowledge and services.

4.3. Framework evaluation

4.3.1. Medical knowledge and services portal

The main purpose of this portal is to help medical researchers and specialists perform the accumulation, integration, and evaluation of modern clinical knowledge and to help physicians with electronic consultations in difficult cases with patients.

For a uniform representation and unambiguous interpretation of knowledge bases by domain experts from different institutions, a common set of all concepts, words, and terms used in practice is required. The experts formed a universal resource *Base of medical terminology and observations*. It was used for the formation of knowledge bases on diseases and syndromes from various nosological groups, on treatment, and on critical conditions assessment. These ontological knowledge bases support the description of fuzzy relations, various approaches to symptom complex forming, including the description of simple and compound signs of diseases, which change over time, various types of conditions for therapy choosing, and other types of relationships that are important for diagnostics, treatment prescriptions, prognosis, etc. One such basis is the *Knowledge base for the diagnosis of diseases and syndromes* [51].

With the use of the first specialized technology for service development and based on the expandable set of medicine ontologies, the shells (solvers with UI) have been created [52] – *Shell for differential diagnostics*, *Shell for the choice of personalized treatment tactics*, *Shell for identifying and assessing the risks of critical conditions*, etc. On the basis of these shells, many clinical XAI-based DSS and services have been created. Most often, they are intended for preliminary and

accurate diagnosis, medical and rehabilitation treatment, and monitoring the recovery process after prescribed treatment.

The used development approach and tools allow significantly reducing the labor costs for the development of applied DSS in medicine (based on KBs). Let's consider an example of developing several DSS for prescribing treatment for 10 diseases. To do this, in accordance with the traditional methodology for knowledge-based intelligent systems development, one needs to develop 10 knowledge bases, 10 solvers and user interfaces, including subsystems for inputting and generating explanations. Developing a KB on treatment for one disease using the traditional approach requires, according to our experience [52], an average of 12 man-months (with a total of 120 man-months). The use of the proposed approach and a set of tools reduces the development time of one KB by more than two times – to 5 man-months (giving 50 man-months in total). To create one solver, an average of 14 man-months is required if the solver is not universal (giving a total of 140 man-months for 10 solvers). Creating a UI (for electronic medical record input subsystem, generating explanations) requires about 12 man-months (with a total of 120 man-months). In the case of a medical knowledge and services portal, the provided shell already includes a universal ontology-driven solver and user interface – they are not required to be developed. Usually, the input subsystem remains the same, but it may be necessary to customize the explanation and adapt the interface to the user's needs. This process is not labor-intensive and takes no more than 1 man-month (10 man-months for 10 diseases in total). Thus, when developing 10 DSS for prescribing treatment, development labor costs are reduced by approximately six times (380 versus 60 man-months).

4.3.2. Laser-based additive manufacturing knowledge portal

The main purpose of the knowledge portal on Laser-Based Additive Manufacturing (LBAM) of metal parts (Fig. 7) is to assist operators of laser and robotic technological equipment in identifying an effective (according to the quality criterion of the LBAM process) operating mode for that equipment while part manufacturing.

The structure of the knowledge portal has been created using the *Control system*. Information resources of the portal include a set of related ontological models, as well as those formed on their basis: a set of databases and reference books used in LBAM, a *Knowledge base on the LBAM mode settings*, a *Case database*, and a *Database of mathematical models* that describe thermodynamic processes occurring during laser melting of metal material.

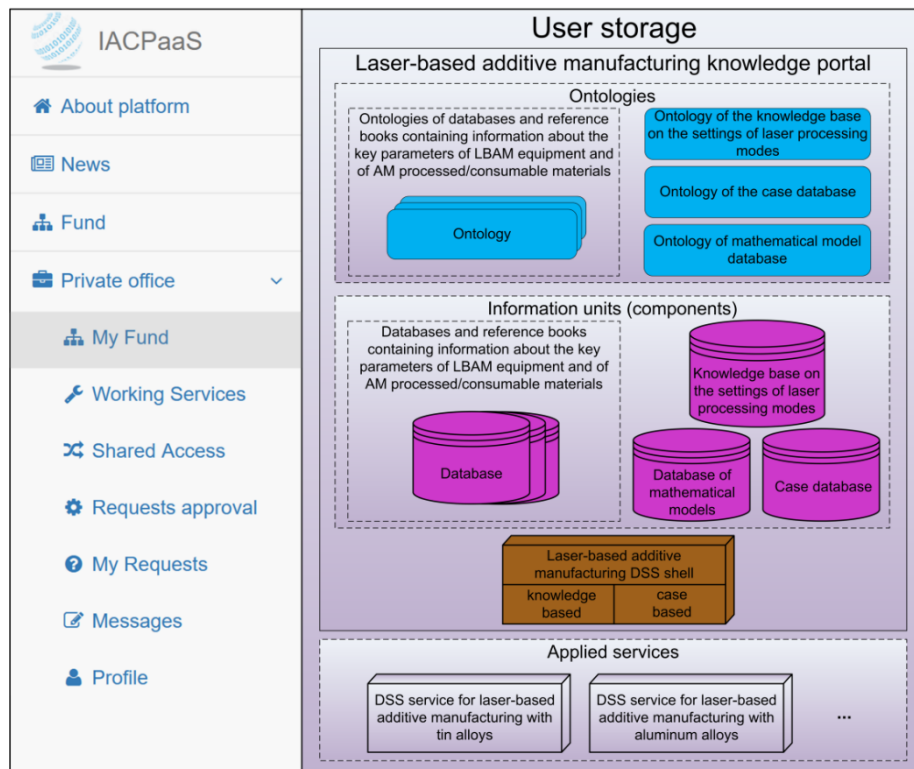


Fig. 7. Portal of knowledge on laser-based additive manufacturing of metal parts

The set of related ontological models provides the foundation for a unified and terminologically consistent representation of all databases and knowledge bases. This representation conforms to established LBAM standards and reflects the conceptual models of domain specialists. The ontological models were developed using the *Ontology editor*.

The *Knowledge base on the LBAM mode settings* contains dependencies that establish the relationships between chemical composition and features of the parts to be processed, chemical composition and technological properties of the clad metal powders, gas environment to be created in the cladding process, a set of key controlled process parameters, and the resulting part features. For each key process parameter, a set of rules for setting its possible values is specified.

The accumulated *Case database* contains an archive of formalized protocols of performed technological operations, structured by types of processed alloys and types of technological process.

The *Database of mathematical models* is a structured repository for software implementations for numerical modeling of thermodynamic processes that take place in the area of interaction of a focused laser beam with a material to be processed.

The software components of the portal include specialized ontology-driven editors for the formation and maintenance of all databases and knowledge bases by domain experts, a DSS shell, and a set of applied services. The editors are created “on-the-fly” with the use of core layer components.

The DSS shell is based on the methods of knowledge engineering, search by analogy (case-based reasoning), as well as numerical mathematical modeling.

To provide the possibility for interaction of the DSS shell with mathematical software for numerical modeling of thermodynamic processes (Wolfram Mathematica), the *Technology for cloud service interaction with external software* is used [53].

5. Discussion and future work

The advantages and limitations of the approach can be inferred from the experience of use of the described IACPaaS platform. It is being used for building AI systems based on ontologies and is accessible to users (programmers, cognitive scientists, analysts, domain experts, etc.) via the Internet (<https://iacpaas.dvo.ru>). Currently, the platform has almost 700 registered users. They have created roughly 370 problem solvers, around 1200 ontologies, and about 2400 knowledge bases and databases. This imposes a responsibility on the team – to ensure its operability, continuous development, and improvement of tools based on the accumulated user experience (including the received feedback).

Unlike existing analogues, the platform provides support for the development of all components of intelligent services. The framework takes into account the specifics of systems with knowledge bases; in particular, it is designed for different types of specialists – domain experts, knowledge engineers, software developers, and it is scalable. Tools for this scaling and tools for creating SS using the framework are based on the same technologies (methodologies). SS are developed based on ontologies, and the tools themselves are also ontology-based. Unified principles and mechanisms for creating applied and instrumental services, common means of expanding and developing tools allow increasing their functionality not only by platform authors, but also by users. Thus, both developers and users are responsible for maintainability and quality of new extensions.

Nevertheless, we periodically meet problems and come across certain limitations. They are put as a basis for the directions of our future research.

Much attention is paid to the automation of tools for the development of software components that process ontological information resources. In particular, work is underway to create a methodology (and its tool support) for the formation

of domain-independent problem solvers. This methodology is meant to simplify the development of services based on solvers of typical intelligent problems for different domains. It offers an ontology of knowledge for solving a class of problems described in abstract terms, a problem solver based on an abstract ontology, and a method of adapting the ontology to a specific domain that does not require reprogramming of the solver.

Work continues improving the API of remote interaction with the platform for making external services, including mobile ones, act as convenient wrappers over the intelligent services of the platform.

As part of the support for the needs of knowledge engineers and experts (who form ontologies and information resources based on them), the *Digraph editor* is regularly updated. Another block of work is related to improving the team development of services and their components.

A promising direction of research being conducted is the development of new methods for creating adaptive multimodal UI, the evaluation of knowledge bases, the expansion of methods for the team creation of intelligent systems, using various methodologies, including the creation of hybrid intelligent systems (based on ontological knowledge bases, machine learning, reasoning by analogy, mathematical modeling), and implemented on heterogeneous hardware.

Another task is the further reduction of database and knowledge base creation labor intensity. Currently, they are formed mainly using appropriate ontology-oriented editors. In this regard, we are initiating two directions of work. The first one is to generate formalized knowledge bases based on datasets using an inductive approach. The second one is the application of natural language processing methods for automatic extraction of relevant information from semi-structured and unstructured texts with generation of corresponding information resources. Here we plan to use an approach based on the so-called large language models (LLMs). The result of such text analysis should be a graph-like structure (in JSON format) which coincides with a given ontology. This structure can then be imported using an editor into an information resource created based on this ontology (see Fig. 5).

One more utilization of the LLM within the platform emerges from the availability of its combination with the Retrieval Augmented Generation mechanism. This modern technique can be used for working with information (and knowledge in particular) in order to solve a challenging problem of providing support for human-like requests (i.e., “chat with knowledge base”). Such a feature can surpass our recently implemented query mechanism. A query must be written in a specific language (provided as an ontology), and this requires particular skills for mastering the creation of the query for the desired result. A human-dialog-like

chatbot could be much more friendly for users, though the trust in its answers can be lower. The desired chat feature is planned for implementation as an additional (and partly remote) tool – via the technology, which is described in [50].

Users make a great contribution to the development of the platform. They extend the functionality of the platform both by creating shared problem-independent solutions and by creating a large number of domain-specific information resources and software components. The described knowledge portals are being implemented and supported by a broad professional community.

6. Conclusion

A necessary condition for the viability of any SS is the availability of extensible tools with the use of which it is created. The longer the life cycle of the SS is, the more expected becomes the need for its developers to improve its adaptation mechanisms to changes in user requirements, domain, and operating conditions. Users of the framework can improve it if the means of its scaling are as close as possible to the means of developing SS themselves, and the complexities of these processes are comparable. Considering that the developers of the SS are not only programmers, but also knowledge engineers and domain experts, the proposed framework contains development mechanisms based on a unified declarative representation of information and software components with subsequent automatic generation of ontology-driven editors for their formation.

All the principles described in this article are implemented on the IACPaaS cloud platform. The platform has tool support for the development of all SS components. Currently, the platform is considered as an environment for software development based on ontologies with semantic (graph) representation. It can be used for the development of graph databases and applications which use them, or as an integrating basis for the development of hybrid systems. Other ways of exploitation are also possible.

Platform usage has shown that the development of tools is actively carried out by its users. At the same time, tools developed “for personal needs”, as well as various components (ontologies, knowledge and databases, message templates, agents, solvers and services), are reused by both their creators in other projects and third-party users. So, a component approach to the development of SS is adopted in practice.

The enhancement of the platform by its developers and users is ongoing dynamically, which makes it possible to create viable ontology-centric systems. It is also important to note that the viability of those systems is ensured not only by the

scalability of the implemented framework. The architecture and ways of representing all SS components play a key role too. The agent-based approach, the formation of components based on their declarative structural models of ontologies using editors, and automation tools implemented on the IACPaaS platform and described earlier in [35, 36] are also aimed at achieving this important characteristic of the SS.

Acknowledgements: The research was supported by the Ministry of Education and Science of Russia, Grant No 075-15-2024-642 (Study of the processes and patterns of occurrence, development and transformation of catastrophic phenomena in the oceans and on continents using seismoacoustic monitoring methods).

References

1. Beniwal, R., K. Abhiheet, K. Kumar, M. Sagar. Ontology Driven Software Development for Better Understanding and Maintenance of Software System. – In: A. K. Luhach, R. C. Poonia, X. Z. Gao, D. Singh Jat, Eds. Proc. of 2nd International Conference on Sustainable Technologies for Computational Intelligence, Dehradun, India, 23-24 December 2022, Singapore, Springer, 2022, pp. 73-81.
2. Chuprina, S. I., K. V. Ryabinin, D. V. Koznov, K. A. Matkin. Ontology-Driven Visual Analytics Software Development. – Program. Comput. Software, Vol. **48**, 2022, pp. 208-214. DOI: 10.1134/S0361768822030033.
3. Abu-Salih, B., M. Al-Tawil, A. Khoury, D. A. Al-Qudah, I. Abu Zaid, M. Alabdale, D. Azar. MAD-Onto: An Ontology Design for Mobile App Development. – Front. Artif. Intell., Vol. **8**, 2025, 1508225. DOI: 10.3389/frai.2025.1508225.
4. Kudryavtsev, D., T. Gavrilova, M. Smirnova, K. Golovacheva. Modelling Consumer Knowledge: The Role of Ontology. – Procedia Computer Science, Vol. **176**, 2020, pp. 500-507. DOI: 10.1016/j.procs.2020.08.052.
5. Moreno Torres, B., C. Völker, S. M. Nagel, T. Hanke, S. Kruschwitz. An Ontology-Based Approach to Enable Data-Driven Research in the Field of NDT in Civil Engineering. – Remote Sens., Vol. **13**, 2021, 2426. DOI: 10.3390/rs13122426.
6. Paper, D. State-of-the-Art Deep Learning Models in TensorFlow: Modern Machine Learning in the Google Colab Ecosystem. – Apress Berkeley, CA, 2021. DOI: 10.1007/978-1-4842-7341-8.
7. Onagh, E., M. Nayebi. Extension Decisions in Open Source Software Ecosystem. 2024. DOI: 10.1016/j.jss.2025.112552.
8. Bönnisch, K., L. Losaria. Kaggle Chronicles: 15 Years of Competitions, Community and Data Science Innovation, 2025. DOI: 10.48550/arXiv.2511.06304.
9. Chernikov, B. Software Quality Management. Moscow, Russia, INFRA-M Publishing House, 2020.
10. Islam, M., N. Farooqui, M. Haleem, S. Zaidi. An Efficient Framework for Software Maintenance Cost Estimation Using Genetic Hybrid Algorithm: OOPs Prospective. – International Journal of Computing and Digital Systems, Vol. **14**, 2023, pp. 933-943. DOI: 10.12785/ijcds/140172.

11. Rybina, G. V., A. A. Slinkov. The Implementation of the Ontological Approach to Control of the Processes of Designing Integrated Expert Systems Based on the Problem-Oriented Methodology. – In: S. M. Kovalev, S. O. Kuznetsov, A. I. Panov, Eds. Artificial Intelligence, RCAI 2021. Lecture Notes in Computer Science. 12948, Cham., Springer. DOI: 10.1007/978-3-030-86855-0_25.
12. Gonedally, S., K. Amuda, P. Kumbum, V. Adari, V. Chunduru. The Evolution of Software Maintenance. – Journal of Computer Science Applications and Information Technology, Vol. 6, 2021, pp. 1-8. DOI: 10.15226/2474-9257/6/1/00150.
13. Kudryavtsev, D., T. Gavrilova. An Overview of Practical Ontology Implementation in Decision Support Systems. – In: D. Arseniev, L. Overmeyer, H. Kälviäinen, B. Katalinić, Eds. Cyber-Physical Systems and Control, CPS&C 2019. Lecture Notes in Networks and Systems. Vol. 95. 2020, Cham. Springer. DOI: 10.1007/978-3-030-34983-7_3.
14. Mitrofanova, O. A., N. S. Konstantinova. Ontologies as Knowledge Storage Systems. All-Russian Competitive Selection of Review and Analytical Articles in the Priority Direction “Information and Telecommunication Systems” (Accessed on 27 March 2025). <https://lib.nsu.ru/xmlui/handle/nsu/8979>
15. Bastías, O. A., J. Díaz, J. López Fennér. Exploring the Intersection between Software Maintenance and Machine Learning – A Systematic Mapping Study. – Appl. Sci., Vol. 13, 2023, 1710. DOI: 10.3390/app13031710.
16. Glinskiy, B., A. Sapetina, A. Snytnikov, Y. Zagorulko, G. Zagorulko. The Automated Construction of a Scheme for Solving Compute-Intensive Problems Based on the Ontological Approach and SemanticWeb Technologies. – Journal of Physics: Conference Series, Vol. 2099, 2021, 012022. DOI: 10.1088/1742-6596/2099/1/012022.
17. Ahmed, E. O. The Study of Software Engineering for Development and Maintenance of Software Systems. – International Journal of Intelligent Systems and Applications in Engineering, Vol. 12, 2023, No 8s, pp. 128-137.
18. Gwendolyn, J. O. Trends and Insights on Tools Used for the Development of Expert Systems: A Systematic Review of Research Articles (2018-2022). – International Journal of Engineering Applied Sciences and Technology, Vol. 8, 2023, No 1, pp. 331-347. DOI: 10.33564/IJEAST.2023.v08i01.050.
19. Al-Rousan, T. Success Factors for Conducting Software-Process Improvement in Web-Based Software Projects. – Cybernetics and Information Technologies, Vol. 24, 2024, No 1, pp. 64-81.
20. Dolzhenkov, V. N., I. D. Maltzegov, A. I. Makarova, N. S. Kamarova, P. V. Kukhtin. Software Tools for Ontology Development. – International Journal of Advanced Trends in Computer Science and Engineering, Vol. 9, 2020, No 2, pp. 935-941. DOI: 10.30534/ijatcse/2020/05922020.
21. Lyadova, L., N. Suvorov, V. Zayakin, E. Zamyatina. An Ontological Approach to the Development of Analytical Platform Language Toolkits. – In: Proc. of 16th IEEE International Conference on Application of Information and Communication Technologies (AICT'22), Washington, DC, USA, 12-14 October 2022, pp. 1-6. DOI: 10.1109/AICT55583.2022.10013576.
22. Pileggi, S. F. Ontology in Hybrid Intelligence: A Concise Literature Review. – Future Internet, Vol. 16, 2024, 268. DOI: 10.3390/fi16080268.

23. Konaté, J., A. Gueye, P. Zaraté, G. Camilleri. Collaborative Decision-Making: A Proposal of a Semi-Automatic Facilitation Based on an Ontology. – International Journal of Information Technology & Decision Making, Vol. **22**, 2023, No 1, pp. 447-470. DOI: 10.1142/S0219622022500420.
24. Ko, A., S. Gillani. A Research Review and Taxonomy Development for Decision Support and Business Analytics Using Semantic Text Mining. – International Journal of Information Technology & Decision Making, Vol. **19**, 2020, No 1, pp. 97-126. DOI: 10.1142/S0219622019300076.
25. Mahoro, L. J., J. V. Fonou-Dombeu. A Comparative Analysis of dot NET-Based and Open Source Platforms for Ontology Development. – In: S. Pudaruth, U. Singh, Eds. Proc. of International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD'20), Durban, South Africa, 2020, pp. 1-7. DOI: 10.1109/icABCD49160.2020.9183887.
26. Deckers, R., P. Lago. Systematic Literature Review of Domain-Oriented Specification Techniques. – Journal of Systems and Software, Vol. **192**, 2022, 111415. DOI: 10.1016/j.jss.2022.111415.
27. ArchiGraph.KMS (Accessed on 27 March 2025).
https://trinidata.com/asuz_info.htm
28. OsaWL. OsaWL framework functionality (Accessed on 27 March 2025).
<https://t-systema.com/osawl-framework-functionality-2>
29. Di Pierro, D., S. Ferilli, D. Redavid. LPG-Based Knowledge Graphs: A Survey, a Proposal and Current Trends. – Information, Vol. **14**, 2023, No 3, 154. DOI: 10.3390/info14030154.
30. Abdalla, R., A. Mishra. Using Agent-Based Methodologies in Healthcare Information Systems. – Cybernetics and Information Technologies, Vol. **18**, 2018, No 2, pp. 123-132.
31. Gan, L., Y. Huang, Y. Cheng. Research on Intelligent Learning Platform System Based on Spring Boot. – In: H. Wu, T. Mishra, V. Erokhin, Eds. Proc. of International Conference on Computer Science, Information Engineering and Digital Economy (CSIEDE'22), Guangzhou, China, 28-30 October 2022, pp. 168-179. DOI: 10.2991/978-94-6463-108-1_19.
32. Gensym G2. The World's Leading Software Platform for Real-Time Expert System Application. (Accessed on 27 March 2025).
<http://www.gensym.com/wp-content/uploads/Gensym-l-G2.pdf>
33. Gensym – G2 Applications. Articles. (Accessed on 27 March 2025).
<https://gensym-ignitetechnology.com/section/3003-articles>
34. Gavrilova, T. A., E. V. Strakhovich. Visual Analytical Thinking and Mind Maps for Ontology Engineering. – Ontology of Designing, Vol. **10**, 2020, No 1, pp. 87-99. DOI: 10.18287/2223-9537-2020-10-1-87-99.
35. Gribova, V. V., Ph. M. Moskalenko, V. A. Timchenko, E. A. Shalfeeva. Intelligent Services Development Technology Using the IACPaaS Cloud Platform. – Communications in Computer and Information Science, Vol. **1625**, 2022, pp. 19-38. DOI: 10.1007/978-3-031-15882-7_2.
36. Gribova, V. V., Ph. M. Moskalenko, V. A. Timchenko, E. A. Shalfeeva. The IACPaaS Platform for Developing Systems Based on Ontologies: A Decade of Use. – Scientific and Technical Information Processing, Vol. **50**, 2023, No 5, pp. 406-413. DOI: 10.3103/S0147688223050064.

37. Dimitrova, Z., V. Dimitrov, D. Borissova, I. Garvanov, M. Garvanova. Two-Stage Search-Based Approach for Determining and Sorting of Mountain Hiking Routes Using Directed Weighted Multigraph. – *Cybernetics and Information Technologies*, Vol. **20**, 2020, No 6, pp. 28-39.
38. Ry's, A., L. Lima, J. Exelmans, D. Janssens, H. Vangheluwe. Model Management to Support Systems Engineering Workflows Using Ontology-Based Knowledge Graphs. – *Journal of Industrial Information Integration*, Vol. **42**, 2024, 100720. DOI: 10.1016/j.jii.2024.100720.
39. Kong, F., S. Ahn. Use of Knowledge Graphs for Construction Safety Management: A Systematic Literature Review. – *Information*, Vol. **15**, 2024, 390. DOI: 10.3390/info15070390.
40. Wan, Y., C. Yuan, M. Zhan, L. Chen. Robust Graph Learning with Graph Convolutional Network. – *Information Processing & Management*, Vol. **59**, 2022, No 3, 102916. DOI: 10.1016/j.ipm.2022.102916.
41. Listl, F., D. Dittler, G. Hildebrandt, V. Stegmaier, N. Jazdi, M. Weyrich. Knowledge Graphs in the Digital Twin: A Systematic Literature Review about the Combination of Semantic Technologies and Simulation in Industrial Automation. – *IEEE Access*, Vol. **12**, 2024, pp. 187828-187843. DOI: 10.1109/ACCESS.2024.3514923.
42. Albagli-Kim, S., D. Beimel. Knowledge Graph-Based Framework for Decision Making Process with Limited Interaction. – *Mathematics*, Vol. **10**, 2022, No 21, 3981. DOI: 10.3390/math10213981.
43. Hegde, H., J. Vendetti, D. Goutte-Gattat, J. H. Caufield, J. B. Graybeal, N. L. Harris, N. Karam, C. Kindermann, N. Matentzoglou, J. A. Overton, M. A. Musen, C. J. Mungall. A Change Language for Ontologies and Knowledge Graphs. – *Database*, Vol. **2025**, 2025, baae133. DOI: 10.1093/database/baae133.
44. Fregnan, E., J. Fröhlich, D. Spadini, A. Bacchelli. Graph-Based Visualization of Merge Requests for Code Review. – *Journal of Systems and Software*, Vol. **195**, 2023, 111506. DOI: 10.1016/j.jss.2022.111506.
45. Wang, M., C. Tao, H. Guo. LCVD: Loop-Oriented Code Vulnerability Detection via Graph Neural Network. – *Journal of Systems and Software*, Vol. **202**, 2023, 111706. DOI: 10.1016/j.jss.2023.111706.
46. Gribova, V. V., 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*, Vol. **49**, 2015, No 5, pp. 172-181. DOI: 10.3103/S0005105515050052.
47. Moskalenko, P. M., V. A. Timchenko. Forming Knowledge Bases in Accordance with Ontological Agreements on an Intelligent Systems Development Platform. – *Pattern Recognition and Image Analysis*, Vol. **33**, 2023, No 3, pp. 432-445. DOI: 10.1134/S1054661823030331.
48. Gribova, V., A. Kleshchev, P. Moskalenko, V. Timchenko, L. Fedorischev, E. Shalfeeva, E. Zamburg. Technology for the Development of Intelligent Service Shells Based on Extended Generative Graph Grammars. – In: *Proc. of 3rd Russian-Pacific Conference on Computer Technology and Applications (RPC'18)*, Vladivostok, Russia, 2018, pp. 1-5. DOI: 10.1109/RPC.2018.8482129.

49. Gribova, V. V., P. M. Moskalenko, V. A. Timchenko, E. A. Shalfeyeva. Control Graph Based Development of Problem Solvers for Systems with Knowledge Bases. – Software Engineering, Vol. **12**, 2021, No 3, pp. 115-126. DOI: 10.17587/prin.12.115-126.
50. Gribova, V. V., L. A. Fedorischev, P. M. Moskalenko, V. A. Timchenko. Interaction of Cloud Services with External Software and Its Implementation on the IACPaaS Platform. – In: A. Srivastava, V. Gribova, Eds. Short Paper Proc. of VI International Conference on Information Technologies and High-Performance Computing (ITHPC'21), Khabarovsk, Russia, 14-16 September 2021, pp. 8-18.
51. Gribova, V. V., Y. N. Kul'chin, M. V. Petryaeva, D. B. Okun, R. I. Kovalev, E. A. Shalfeyeva. An Intelligent System for Medical Decision Support in Differential Diagnosis and Treatment of COVID-19. – Herald of the Russian Academy of Sciences, Vol. **92**, 2022, No 4, pp. 511-519. DOI: 10.1134/S1019331622040128.
52. Gribova, V. V., R. I. Kovalev, D. B. Okun. A Specialized Shell for Intelligent Systems of Prescribing Medication. – Scientific and Technical Information Processing, Vol. **48**, 2021, No 5, pp. 315-326. DOI: 10.3103/S0147688221050038.
53. Kul'chin, Y. N., V. V. Gribova, V. A. Timchenko, M. V. Polonik, D. S. Pivovarov, D. S. Yatsko, P. A. Nikiforov, A. I. Nikitin. Methodology for Designing a Laser-Based Additive Process for the Formation of a Lattice Periodic Structure on the Surface of an Aluminum Alloy Plate. – Automation and Remote Control, Vol. **83**, 2022, pp. 138-152. DOI: 10.1134/S0005117922010106.

Received: 11.11.2025; Revised Version: 27.03.2026; Accepted: 15.04.2026

© 2026 by the authors. Licensee IICT-BAS, Sofia, Bulgaria. This article is an open access article distributed under the terms and conditions of the **Creative Commons Attribution-NonCommercial-NoDerivatives** license.